Cryptographic Protocols

George Teşeleanu

"SIMION STOILOW" INSTITUTE OF MATHEMATICS OF THE ROMANIAN ACADEMY

Cryptographic Protocols

Supervisor:
Ferucio Laurenţiu Ţiplea

Author:
George Teşeleanu

Examiners:
Ioana Boureanu
Mirosław Kutyłowski
Mark Manulis

Defense date:
28 October 2021

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

Bucharest, August 2021

Declaration of Authorship

I, GEORGE TEŞELEANU, declare that this thesis titled, "CRYPTOGRAPHIC PROTOCOLS" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this Institute.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"The Ultimate Answer to Life, The Universe and Everything is...42!"

Douglas Adams, The Hitchhiker's Guide to the Galaxy

Contents

List of Figures

List of Tables

Abbreviations

CRT	The **C**hinese **R**emainder **T**heorem
GUZK	**U**nified **G**eneric **Z**ero **K**nowledge Protocol
HKE	**H**ashed **D**iffie-**H**ellman **K**ey **E**xchange
IBE	**I**dentity **B**ased **E**ncryption
i.i.d	**i**ndependent and **i**dentically **d**istributed
IoT	**I**nternet **o**f **T**hings
PKE	**P**ublic **K**ey **E**ncryption
PPT	**P**robabilistic **P**olynomial-**T**ime Algorithm
PRNG	**P**seudo-**R**andom **N**umbers **G**enerator
ROM	**R**andom **O**racle **M**odel
RNG	**R**andom **N**umbers **G**enerator
UDS	**U**nified **D**igital **S**ignature
u.i.i.d	**u**niformly **i**ndependent and **i**dentically **d**istributed
UZK	**U**nified **Z**ero **K**nowledge Protocol
ZKP	**Z**ero **K**nowledge **P**roofs

Notations

λ, κ	Security parameters
$x \xleftarrow{\$} X$, $x \in_R X$	The value x is chosen uniformly at random from a sample space X
$x \leftarrow y$	The assignment of value y to variable x
$vec \leftarrow \{val\}$	The initialization of all the entries of a vector vec with a value val
$[s, t]$	The subset $\{s, \ldots, t\} \subset \mathbb{N}$, where $s \leqslant t$. When s and t are real numbers by $[s, t]$ we understand the set of real numbers lying between s and t
$[s, t)$	$[s, t-1]$, where $s < t$
$M(\alpha, \beta, \mathbb{G})$	The set of matrices with α rows, β columns and entries from $\mathbb{G}$
$GL(\alpha, \mathbb{G})$	The set of invertible matrices with entries from $\mathbb{G}$ of size $\alpha \times \alpha$
$\mathcal{A}^{\times}$	The set of all strings over $\mathcal{A}$
$\{0,1\}^{\ell}$	The set of bit strings of length ℓ
A^T	The transpose of matrix A
$x \| y$	The string obtained by concatenating x and y in this order
$\|m\|$	The number of letters in a string m
$\|x\|$	The bit-length of variable x
$\|\mathbb{G}\|$	The cardinality of set $\mathbb{G}$
$NULL$	An empty variable
$\|$	Bitwise or operator
&	Bitwise and operator

$\oplus$	Bitwise xor operator
$\ll$	Left shift operator
$\gg_s$	Right shift operator for a signed integer
$==$	Equality testing operator
$+=$, $*=$	Compound assignment operators
$++$	Operator used for incrementing a variable
&	Reference to a variable
$size()$	Member function that returns the size of the object
$substring(pos, npos)$	Member function that returs a substring starting from *pos* and containing *npos* characters
$push\_back(val)$	Member function that adds *val* at the end of a vector
$sort$	Member function that sorts a vector in descending order
$MSB_\ell(Q)$	The most significant ℓ bits of Q[1]
$LSB_\ell(Q)$	The least significant ℓ bits of Q
$MID_{[\ell_1,\ell_2]}(Q)$	The bits of Q between position ℓ_1 and ℓ_2
b^α	α consecutive bits of b
$0^\alpha 1^\beta 0^\gamma$	An $(\alpha+\beta+\gamma)$-bit word that has α bits of 0, followed by β bits of 1 and γ trailing zeros
`0b`	The prefix for numeric constants represented in binary
`0x`	The prefix for numeric constants represented in hexadecimal
$\pi = \{a_0, a_1, \ldots, a_\ell\}$	A permutation π defined by $\pi(i) = a_i$ for all i values
Id	The identity permutation $Id = \{0, \ldots, \ell\}$
$Pr[E]$	The probability of the event E to happen
$gcd(a,b)$, (a,b)	The greatest common divisor of two integers a and b (the distinction between gcd and the utilization of parenthesis for pairing will be clear from context)
$a \equiv b \bmod n$, $a \equiv_n b$	The integers a and b are congruent modulo n, where n is an integer
a div n	The quotient of the integer division of a by n
$(a)_n$	The remainder of the integer division of a by n

[1] We use the big-endian convention

$SQRT_n(A)$	The set of all square roots $x \in \mathbb{Z}_n$ of integers $a \in A$
$SQRT_n(a_1, \ldots, a_m)$	$SQRT_n(A)$, if $A = \{a_1, \ldots, a_m\}$
$\left(\frac{a}{n}\right)$, $J_n(a)$	The Jacobi symbol[2] of an integer a modulo an integer n
$QR_n(A)$	The set of quadratic residues modulo n from A
QR_n	$QR_n(\mathbb{Z}_n^*)$
$QNR_n(A)$	The set of quadratic non-residues modulo n from A
QNR_n	$QNR_n(\mathbb{Z}_n^*)$
$J_n^+(A)$	The set of quadratic residues modulo n with Jacobi symbol 1
J_n^+, J_n	$J_n^+(\mathbb{Z}_n^*)$
$J_n^-(A)$	The set of quadratic residues modulo n with Jacobi symbol -1
J_n^-, $\bar{J}_n$	$J_n^-(\mathbb{Z}_n^*)$
$J_n^{\pm}(A)$	$J_n^{\pm}(A) = \{a \in A \mid J_p(a) = +1,\ J_q(a) = -1\}$, when $n = pq$ and $p < q$
$J_n^{\mp}(A)$	$J_n^{\mp}(A) = \{a \in A \mid J_p(a) = -1,\ J_q(a) = +1\}$, when $n = pq$ and $p < q$
$\mathbb{Z}_p$	The group of integers modulo p
$\mathbb{Z}_p^*$	The set $\{a \in \mathbb{Z}_p \mid gcd(a, p) = 1\}$
$\mathcal{Z}_p$	$\mathcal{Z}_p = \{-(p-1)/2, \ldots, -1, 0, 1, \ldots, (p-1)/2\}$
$v = \{v_i\}_{i \in A}$	The multidimensional vector $v = (v_0, \ldots, v_{\lvert A \rvert})$, where $A \subset \mathbb{N}$
$AES_k(m)$	Encryption of message m with key k using the AES algorithm[3]
C_k^n	The binomial coefficient "n choose k"
$B(p)$	A Bernoulli distribution, where p is the probability of obtaining a 1
ε	The bias $\varepsilon = p - 0.5$, where p is the probability of obtaining a 1
P_a	The probability of a random string being a
$Pr[A]$	For any $A \subseteq \mathbb{Z}_2^n$ we define $Pr[A] = \sum_{a \in A} P_a$

[2] For simplicity we will use the terminology of Jacobi symbol for both prime or composite moduli.
[3] We refer the reader to [85] for a description of AES.

$RSAgen(\lambda)$	A PPT algorithm that, given a security parameter λ, outputs a triple (n, p, q), where $n = pq$ is an RSA modulus

Dedicated to my mother

Chapter 1

Preface

Throughout history, the main role of cryptography has been to keep sensible information private, even in the presence of an adversary that has control over the communication channel. Even though privacy remains central to cryptography, the field has expanded and it incorporates other goals, such as data integrity and authenticity, access control or electronic payments.

Once used only by the military, cryptography is now in widespread use and people benefit from it daily, even without know it. For example, when buying an item online a secure channel is used to process the transaction and implicitly to ensure the privacy of your credit card. Or, when communicating through messaging apps our private conversations are protected using end-to-end encryption. With such a growing area of applicability, is not surprising that modern cryptography intertwines concepts from mathematics, computer science, engineering and physics.

Although a remarkable science, cryptography is also an art and a puzzling game. We have to think as an attacker would, while defending the system against threats; we have to juggle between speed, usability and security; we have to twist known concepts in order to make them fit our scope; we have to design high level concepts, while keeping in mind the low level ones etc. Influenced by the plethora of concepts a cryptographer has to manage, in this work we touch on different areas of cryptography and we either take the role of the designer or of the attacker. By presenting both sides of the same coin, we wish that the reader will start to appreciate the beauty of this puzzling science and will begin to see the relationships that arise between seemingly different concepts.

1.1 Outline

We further present a brief synopsis of the seven main chapters contained in this work. One of the most difficult things about structuring this work was the interdependency of some of the chapters. We have tried to present the material in this thesis in a logical and natural order. Without further ado, here is the thesis outline.

Chapter 2 tackles secret key cryptography and is split into three parts. The first part analyses the security of the (affine) Hill cipher and their corresponding modes of operation. Definitions and background information are presented in Section 2.1.1. The core of the first part consists of Sections 2.1.2 and 2.1.3 that contain several key ranking functions and ciphertext only attacks. Experimental results are provided in Section 2.1.4 and some possible research directions are given in Section 2.1.5. The letter frequencies and the Vigenère attack used in Section 2.1.4 are given in Appendices A and B. Some possible methods for increasing the brute-force complexity for the Grain family of stream ciphers are presented in the second part of this chapter. We introduce notations and give a quick reminder of the Grain family technical specifications in Section 2.2.1. Section 2.2.2 describes generic attacks against the Grain ciphers. In Section 2.2.3 we provide the reader with a security analysis of IV padding schemes for Grain ciphers. We underline various interesting ideas as future work in Section 2.2.4. We recall Grain v1 in Appendix C, Grain-128 in Appendix D and Grain-128a in Appendix E. We do not recall the corresponding parameters of Grain v0, even though the results presented in this section still hold in that case. In Appendices F and G we provide test values for our proposed algorithms. The last part of this chapter studies the effect of using quasigroups isotopic to groups when designing SPNs. Hence, prerequisites are given in Section 2.3.1. An SPN generalization is introduced in Section 2.3.2 and its security is studied in Section 2.3.3.

In Chapter 3 we discuss several public key protocols and some of their applications. The first part introduces several hardness assumptions necessary for proving the protocols' security. Zero-knowledge protocols are studied in the second part of this chapter. Therefore, we recall zero-knowledge concepts in Section 3.2.1. Inspired by Maurer's Unified-Zero Knowledge construction, in Section 3.2.2 we introduce a Unified Generic Zero-Knowledge protocol and prove it secure. We provide the reader with various special cases of UGZK in Section 3.2.3. A hash variant of our core protocol is tackled in Section 3.2.4 together with its security analysis. As a possible application for UGZK, in Section 3.2.5 we describe a lightweight authentication protocol, discuss security and complexity aspects and present implementation trade-offs which arise from small variations of the proposed result. In Section 3.2.6 we underline future work proposals. The third part of this chapter contains a signature scheme inspired by Maurer's UZK paradigm.

The necessary prerequisites are given in Section 3.3.1 and the exact details of the UDS signature are provided in Section 3.3.2. An application for UDS is given in the fourth part of this chapter. More precisely, after introducing preliminaries in Section 3.4.1, we introduce a co-signing protocol built on the legally fair contract signing protocol of Ferradi *et. al* in Section 3.4.2. We discuss some related open problems in Section 3.4.3. Two public key encryption schemes are presented in the fifth part. In Section 3.5.1 we introduce definitions, security assumptions and schemes used throughout the section. First we introduce in Section 3.5.2 a slight modification of the generalized ElGamal encryption scheme, that will be used in a subsequent chapter. Then, inspired by the Joye-Libert PKE scheme and aiming at obtaining a relevant generalization, in Section 3.5.3 we propose a new scheme based on 2^k residues, prove it secure in the standard model and analyze its performance compared to other related cryptosystems. Future work is presented in Section 3.5.3.5 and in Appendix H we present some optimized decryption algorithms for our proposed scheme. The final part of this chapter provides the reader with an application of our Joye-Libert based scheme to biometric authentication. Thus, definitions and security requirements are presented in Section 3.6.1, while our proposed authentication protocol is described in Section 3.6.2.

Some useful results for understanding the security of Cocks' identity based encryption and of certain variations of it are provided in Chapter 4. Basic notions and Cocks' scheme are presented in the first part of the chapter. The second part considers sets of the form $a + X = \{(a + x) \bmod n \mid x \in X\}$, where n is a prime or the product of two primes $n = pq$ and X is a subset of $\mathbb{Z}_n^*$ whose elements have some given Jacobi symbols modulo prime factors of n. The third part of the chapter points out two applications of the previously mentioned results. The first one provides the reader with a deep analysis of some distributions related to Cocks' IBE scheme and Galbraith's test, providing thus rigorous proofs for Galbraith's test. The second application discussed, relates to the computational indistinguishability of some distributions used for proving the security of certain variations of Cocks' IBE. We were able to prove statistical indistinguishability of those distributions without any hardness assumption. The chapter concludes with Section 4.4.

An unconventional method for backdooring cryptographic systems is studied in Chapter 5. The basic notions about kleptographic attacks are given in Section 5.1. The first part of this chapter deal with a threshold kleptographic attack that can be implemented in the generalized ElGamal signature. Thus, in Section 5.2.1 we describe a simplified attack on the generalized ElGamal signature and then extended it in Section 5.2.2. A series of signatures that support the implementation of our attack are provided in Section 5.2.3. Future work is presented in Section 5.2.4 and a two-party malicious signing

protocol is presented in Appendix I. We provide a supplementary kleptographic mechanism in Appendix J. A method for infecting Maurer's UZK protocol is studied in the second part of this chapter. In Sections 5.3.1 and 5.3.2 we present our new general kleptographic attacks and prove them secure. Instantiations of our attacks can be found in Section 5.3.3. Some possible research directions are given in Section 5.3.4. In the third part, we introduce a subscription based marketing model suitable for selling infected devices. Hence, some additional preliminaries are given in Section 5.4.1. Based on the ElGamal encryption algorithm, a series of kleptographic subscriptions that fit different scenarios are provided in Sections 5.4.2 to 5.4.4. We discuss some open problems in Section 5.4.5. Hash channels are tackled in the last part of the chapter. By adapting and improving Wu's mechanism we introduce new hash channels in Section 5.5.1. A series of experiments are conducted in Section 5.5.2, while several applications are provided in Section 5.5.3.

In Chapter 6 we study (pseudo-)random numbers generators. The first part of the chapter deals with Adobe Flash Player's[1] vulnerability in the pseudo-random number generator used for constant blinding. We introduce the necessary prerequisites in Section 6.1.1. The core of our seed recovering mechanism consists of Sections 6.1.2 and 6.1.3 and contains a series of algorithms for inverting a generalized version of the hash function used by the Flash Player. Experimental result are given in Section 6.1.4. Supplementary algorithms may be found in Appendix K. The second part contain an architecture that can be used to implement health tests for random numbers generators. Definitions and background information are presented in Section 6.2.1. Two classes of digital filters that amplify existing biases are described in Sections 6.2.2 and 6.2.3. Some possible applications are given in Section 6.2.4. In Section 6.2.5 we apply our proposed architecture to broken Bernoulli noise sources and present some experimental results. The theoretical model is provided in Section 6.2.6. Some finer measurements are provided in Section 6.2.7. In Section 6.2.8 we underline future work proposals.

Chapter 7 contains several protocols that fall in the category of recreational cryptography. Thus, in Section 7.1 we describe various schemes which aim at solving Yao's millionaires' problem and provide the reader with their corresponding security analyses. In Section 7.2 we present a set of protocols which act as solutions for comparing information without revealing it and discuss their security. In Section 7.3 we describe a public key cryptosystem constructed by means of an electrical scheme and tackle its security. In Appendix L we recall various physical cryptographic solutions which appeared in the literature, while in Appendix M we present a generic physical public key encryption scheme useful for introducing students to different properties of physical systems.

[1] versions 24.0.0.221 and earlier

1.2 Our Contributions

We further link our publish papers with some of the subsections presented in this work.

- Secret Key Cryptography
  - Sections 2.1.2 to 2.1.4 and Appendix B [244]
  - Sections 2.2.2 to 2.2.3 and Appendices F to G [172]
  - Sections 2.3.2 to 2.3.3 [245]
- Public Key Cryptography
  - Sections 3.2.2 to 3.2.5 [173]
  - Section 3.3.2 and Section 3.4.2 [171]
  - Section 3.5.2 [237]
  - Section 3.5.3, Section 3.6.2 and Appendix H [174]
- Identity Based Cryptography
  - Sections 4.2 to 4.3 [246, 247]
- Kleptographic Attacks
  - Sections 5.2.1 to 5.2.3 and Appendices I to J [237]
  - Sections 5.3.1 to 5.3.3 [239]
  - Sections 5.4.2 to 5.4.4 [240]
  - Sections 5.5.1 to 5.5.3 [242]
- (Pseudo-)Random Number Generators
  - Sections 6.1.2 to 6.1.4 and Appendix K [241]
  - Sections 6.2.2 to 6.2.7 [238, 243]
- Physical Cryptography
  - Sections 7.1 to 7.3 and Appendix M [80]

1.3 Published Papers

[P1] Mariana Costiuc, Diana Maimuţ, and George Teşeleanu. Physical Cryptography. In *SECITC 2019*, volume 12001 of *Lecture Notes in Computer Science*, pages 156–171. Springer, 2019.

[P2] Diana Maimuţ and George Teşeleanu. Secretly Embedding Trapdoors into Contract Signing Protocols. In *SECITC 2017*, volume 10543 of *Lecture Notes in Computer Science*, pages 166–186. Springer, 2017.

[P3] Diana Maimuţ and George Teşeleanu. A Unified Security Perspective on Legally Fair Contract Signing Protocols. In *SECITC 2018*, volume 11359 of *Lecture Notes in Computer Science*, pages 477–491. Springer, 2018.

[P4] Diana Maimuţ and George Teşeleanu. New Configurations of Grain Ciphers: Security Against Slide Attacks. In *BalkanCrypt 2018*, Communications in Computer and Information Science. Springer, 2018.

[P5] Diana Maimuţ and George Teşeleanu. A Generic View on the Unified Zero-Knowledge Protocol and its Applications. In *WISTP 2019*, volume 12024 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2019.

[P6] Diana Maimuţ and George Teşeleanu. A New Generalisation of the Goldwasser-Micali Cryptosystem Based on the Gap 2^k-Residuosity Assumption. In *SECITC 2020*, Lecture Notes in Computer Science. Springer, 2020.

[P7] George Teşeleanu. Threshold Kleptographic Attacks on Discrete Logarithm Based Signatures. In *LatinCrypt 2017*, volume 11368 of *Lecture Notes in Computer Science*, pages 401–414. Springer, 2017.

[P8] George Teşeleanu. Random Number Generators Can Be Fooled to Behave Badly. In *ICICS 2018*, volume 11149 of *Lecture Notes in Computer Science*, pages 124–141. Springer, 2018.

[P9] George Teşeleanu. Unifying Kleptographic Attacks. In *NordSec 2018*, volume 11252 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 2018.

[P10] George Teşeleanu. Managing Your Kleptographic Subscription Plan. In *C2SI 2019*, volume 11445 of *Lecture Notes in Computer Science*, pages 452–461. Springer, 2019.

[P11] George Teşeleanu. Reinterpreting and Improving the Cryptanalysis of the Flash Player PRNG. In *C2SI 2019*, volume 11445 of *Lecture Notes in Computer Science*, pages 92–104. Springer, 2019.

[P12] George Teşeleanu. Subliminal Hash Channels. In *A2C 2019*, volume 1133 of *Communications in Computer and Information Science*, pages 149–165. Springer, 2019.

[P13] George Teşeleanu. A Love Affair Between Bias Amplifiers and Broken Noise Sources. In *ICICS 2020*, Lecture Notes in Computer Science. Springer, 2020.

[P14] George Teşeleanu. Cracking Matrix Modes of Operation with Goodness-of-Fit Statistics. In *HistoCrypt 2020*, Linköping Electronic Conference Proceedings. Linköping University Electronic Press, 2020.

[P15] George Teşeleanu. Quasigroups and Substitution Permutation Networks: A Failed Experiment. *Cryptologia*, 2020.

[P16] Ferucio Laurentiu Tiplea, Sorin Iftene, George tese, and Anca-Maria Nica. On the Distribution of Quadratic Residues and Non-residues Modulo Composite Integers and Applications to Cryptography. *Appl. Math. Comput.*, 372, 2020.

[P17] Ferucio Laurenţiu Ţiplea, Sorin Iftene, George Teşeleanu, and Anca-Maria Nica. Security of Identity-Based Encryption Schemes from Quadratic Residues. In *SECITC 2016*, volume 10006 of *Lecture Notes in Computer Science*, pages 63–77, 2016.

Chapter 2

Secret Key Cryptography

The simplest and also the most common method for protecting the confidentiality of messages or authenticating a piece of information is to use a shared secret key between the sender and the receiver. This is called secret/symmetric key cryptography. In this scenario both participants use functions dependent on the same predetermined key. Usually, the shared key is randomly generated.

Symmetric key algorithm are assumed to maintain their security properties as long as adversaries cannot find the used key. This can mean three things: either the key is kept secure by the party using it or the key is large enough to avoid brute forcing it or the algorithm does not leak any information. In this chapter we will deal with two of the aforementioned aspects. More precisely, we will show how the (affine) Hill cipher and their corresponding modes of operation leak critical information through the ciphertext. Then we will describe a method for extending the life of Grain instantiations by increasing their corresponding brute force complexity. Finally, we provide the reader with equivalent instantiations of substitution permutation networks.

2.1 (Affine) Hill Cipher

Two classical ciphers based on linear algebra are the Hill cipher [144] and its affine version [145]. Both use invertible matrices over integers modulo a to encipher messages, where a is the size of the language alphabet $\mathcal{A}$. The first step of the encryption process is the encoding of each plaintext letter into a numerical equivalent. The simplest encoding is `"a"` $= 0$, `"b"` $= 1$ and so on. After encoding, the plaintext is divided into blocks of size k and, then, each block is multiplied with an invertible matrix of size k. In the affine case, a second matrix is added to the result. After each block is transformed, the result

is converted back into letters. To decipher messages, one must perform the above steps in reverse.

Although both ciphers are vulnerable to known plaintext attacks[1], efficient ciphertext only attacks have been developed only a decade ago [42] and only for the Hill cipher with small ks. Note that as k increases simple brute force attacks fail. For example, in the case of the Hill cipher with $a = 26$, we have around 2^{17} keys for $k = 2$, 2^{40} keys for $k = 3$ and 2^{73} keys for $k = 4$ [42]. According to [201, 43], given a and k the exact number of invertible matrices can be computed. Note that in the case of the affine Hill cipher the computational effort made to brute force the Hill cipher is multiplied with a^k.

In 2007, Bauer and Millward [42] introduced a ciphertext only attack for the Hill cipher[2], that was later improved in [266, 167, 178]. The attack was independently published by Khazaei and Ahmadi [154]. The main idea of these attacks is to do a brute force attack on the key rows, instead of the whole matrix, and then recover the decryption matrix.

In [157], Kiele suggests the usage of block-chaining procedures to complicate the algebraic cryptanalytic techniques developed for the Hill cipher. We will show in this section how to adapt the attacks described in [42, 266, 154] to different modes of operation (not only the block-chaining one) for both the Hill cipher and its affine version. Note that some modes do not require the key to be invertible, thus the attack presented in [167] does not work for all Hill based modes. For uniformity, we will only extend Yum and Lee's attack and leave as future work the extension of [167] to modes requiring invertible matrices. We stress that out of the three attacks [42, 266, 154] Yum and Lee's attack has the best performance to message recovery ratio.

Another paper that motivated this study is [41]. The authors of [41] conjecture that the fourth cryptogram of the Kryptos sculpture [9] is either encrypted using the affine Hill cipher or some other sort of cipher mode of operation. We provide the reader with a preliminary study of these conjectures. To prove or disprove these conjectures, one has to find a way to adapt all the presented ciphertext attacks to the secret encoding versions of the (affine) Hill cipher and their corresponding modes of operation. Various partial answers for the secret encoding Hill cipher are provided in [266].

2.1.1 Preliminaries

Conventions. To minimize repetitions, we employ the following system. When reading the attacks against the Hill based modes of operation we invite the reader to ignore

[1] *i.e.* after a number of known messages are encrypted, one can easily recover the encryption key(s) if he has access to the corresponding ciphertexts.

[2] Bauer and Millward's attack for $k = 3$ was previously and independently described online by Wutka [257].

red colored text, while in the case of the affine Hill based modes ignore the blue text. Also, when describing algorithms we prefer using verbose names for variables, while for mathematical descriptions we prefer notations. Additionally, when presenting algorithms we consider only lower case messages represented by ASCII codes (*i*.e. `"c"` − `"a"` = 99 − 97 = 2). The last convention used is to store constants in look-up tables when their size is small (*e.g.* letter frequencies) and in maps, otherwise (*e.g.* quadgraph frequencies).

2.1.1.1 Ciphers

A cipher consists of three probabilistic polynomial-time algorithms: *KeyGen*, *Encrypt* and *Decrypt*. The first one takes as input a security parameter and outputs the secret key. The secret key together with the *Encrypt* algorithm are used to encrypt a message m. The last algorithm decrypts any message encrypted using the known secret key.

Hill cipher. The Hill cipher is a poly-alphabetical cipher based on linear algebra introduced by Lester S. Hill in [144]. We briefly provide the algorithms for the Hill cipher. Note that before encrypting/decrypting a text, the corresponding letters are encoded/decoded as follows: `"a"` to/from 0, `"b"` to/from 1 and so on.

KeyGen(λ): Set an integer $k \geqslant \lambda$ and choose $K_1 \xleftarrow{\$} GL(k, \mathbb{Z}_a)$. Output the secret key $sk = K_1$.

Encrypt(sk, m): Pad message m until $|m| \equiv 0 \bmod k$[3]. Divide m into blocks $m = m_1 \| \dots \| m_\ell$, where $|m_i| = k$. Compute $c_i^T \leftarrow K_1 \cdot m_i^T$. Output the ciphertext $c = c_1 \| \dots \| c_\ell$.

Decrypt(sk, c): Divide c into ℓ blocks $c = c_1 \| \dots \| c_\ell$ and compute $m_i^T \leftarrow K_1^{-1} \cdot c_i^T$. Recover m by removing the padding.

Example 2.1. For clarity, we further provide the reader with an example from [42]. The message `"matrixencryptioniseasy"` is mapped into

$$12, 0, 19, 17, 8, 23, 4, 13, 2, 17, 24, 15, 19, 8, 14, 13, 8, 18, 4, 0, 18, 24.$$

If $K_1 \leftarrow \begin{pmatrix} 1 & 3 \\ 4 & 11 \end{pmatrix}$, then the first block is encrypted into $\begin{pmatrix} 1 & 3 \\ 4 & 11 \end{pmatrix} \cdot \begin{pmatrix} 12 \\ 0 \end{pmatrix} = \begin{pmatrix} 12 \\ 22 \end{pmatrix}$. Therefore, we obtain the ciphertext `"mwsdzzrdbnrbribrkweqmy"`.

[3] Usually a rarely used letter, such as `"x"`, is appended to m until we get the desired length.

Affine Hill cipher. An affine variation of the Hill cipher was introduced in [145]. We shortly provide the algorithms for the affine Hill cipher.

KeyGen(λ): Set an integer $k \geqslant \lambda$ and choose $K_1 \xleftarrow{\$} GL(k, \mathbb{Z}_a)$ and $K_2 \xleftarrow{\$} M(k, 1, \mathbb{Z}_a)$. Output the secret key $sk = (K_1, K_2)$.

Encrypt(sk, m): Pad message m until $|m| \equiv 0 \bmod k$. Divide m into blocks $m = m_1 \| \dots \| m_\ell$, where $|m_i| = k$. Compute $c_i^T \leftarrow K_1 \cdot m_i^T + K_2$. Output the ciphertext $c = c_1 \| \dots \| c_\ell$.

Decrypt(sk, c): Divide c into ℓ blocks $c = c_1 \| \dots \| c_\ell$ and compute $m_i^T \leftarrow K_1^{-1} \cdot (c_i^T - K_2)$. Recover m by removing the padding.

Other affine variations of the Hill cipher. In Table 2.1 we present all the possible affine variations of the Hill cipher. Note that $K_3 \xleftarrow{\$} M(k, 1, \mathbb{Z}_a)$. After performing some computations, we can see that for all variations we can recover m_i^T using $f(c_i) = K_1' \cdot c_i^T + K_2'$. Since we are interested only in recovering the encrypted messages and not the initial secret keys, all the presented attacks try to recover K_1' and K_2'. Thus, for the affine Hill cipher we only consider f for recovering m_i^T.

Encrypt	*Decrypt*	K_1'	K_2'
$c_i^T \leftarrow K_1 \cdot m_i^T + K_2$	$m_i^T \leftarrow K_1^{-1} \cdot (c_i^T - K_2)$	K_1^{-1}	$-K_1^{-1}K_2$
$c_i^T \leftarrow K_1 \cdot (m_i^T + K_2)$	$m_i^T \leftarrow K_1^{-1} \cdot c_i^T - K_2$	K_1^{-1}	$-K_2$
$c_i^T \leftarrow K_1 \cdot (m_i^T + K_2) + K_3$	$m_i^T \leftarrow K_1^{-1} \cdot (c_i^T - K_3) - K_2$	K_1^{-1}	$-K_1^{-1}K_3 - K_2$

TABLE 2.1: Affine variations of the Hill cipher.

2.1.1.2 Cipher Modes of Operation

When we encrypt messages block by block, usually called the ECB mode of operation, identical blocks are mapped into identical ciphertexts. Thus, block patterns are preserved. This is an information leakage that can lead to security breaches. To address this issue several modes of operation where introduced in [98], such as CBC, CTR, CFB and OFB.

In [27], the authors introduce a generalization of the CBC-MAC construction[4]. Based on Alagic *et* al.'s generalization, we present a possible adaptation of the CBC, CTR and CFB modes of operation to the (affine) Hill cipher.

[4] the XOR operation is replaced with a generic group operation

Let $E_k, D_k : M(k, k, \mathbb{Z}_a) \to M(k, k, \mathbb{Z}_a)$ be the matrix transformations of the (affine) Hill cipher's encryption and decryption. We further describe the encryption and decryption algorithms for CBC and CFB.

Encrypt(sk, m): Choose $iv \xleftarrow{\$} M(1, k, \mathbb{Z}_a)$ and pad message m until $|m| \equiv 0 \bmod k$. Divide m into blocks $m = m_1 \| \dots \| m_\ell$, where $|m_i| = k$. Let $m_0 \leftarrow iv$. For CBC compute $c_i \leftarrow E_k(c_{i-1} + m_i)$, while for CFB compute $c_i \leftarrow E_k(c_{i-1}) + m_i$. Let $c = c_1 \| \dots \| c_\ell$. The output is ciphertext (iv, c).

Decrypt(sk, iv, c): Divide c into ℓ blocks $c = c_1 \| \dots \| c_\ell$. For CBC compute $m_i \leftarrow D_k(c_i) - c_{i-1}$ and for CFB compute $m_i \leftarrow c_i - E_k(c_{i-1})$. Recover m by removing the padding.

In the case of CTR, the sender and the receiver each keep a state *ctr*. The initial value is chosen at random $ctr \xleftarrow{\$} M(1, k, \mathbb{Z}_a)$. Before each encryption *ctr* is updated as follows:

Update(ctr): Let $ctr = (\alpha_0, \dots, \alpha_{k-1})$ and $i \leftarrow k - 1$. Compute the following

1. $\alpha_i \leftarrow (\alpha_i + 1) \bmod a$,
2. If $\alpha_i == 0$, then $i \leftarrow (i - 1) \bmod k$ and go to step 1.

Now, the encryption and decryption algorithm for this mode of operation are:

Encrypt(sk, m): Pad message m until $|m| \equiv 0 \bmod k$. Divide m into blocks $m = m_1 \| \dots \| m_\ell$, where $|m_i| = k$. Compute $ctr \leftarrow$ *Update*(ctr) and $c_i \leftarrow E_k(ctr) + m_i$. The output is ciphertext $c = c_1 \| \dots \| c_\ell$.

Decrypt(sk, iv, c): Divide c into ℓ blocks $c = c_1 \| \dots \| c_\ell$. Compute $ctr \leftarrow$ *Update*(ctr) and $m_i \leftarrow c_i - E_k(ctr)$. Recover m by removing the padding.

Example 2.2. For clarity, we provide the reader with some examples for the *Update* function. Let $a = 26$ and $k = 2$. Then *Update*$((1, 2)) = (1, 3)$, *Update*$((1, 25)) = (2, 0)$ and *Update*$((25, 25)) = (0, 1)$.

Although our attacks do not apply to the OFB mode, for completeness we provide its description.

Encrypt(sk, m): Choose $iv \xleftarrow{\$} M(1, k, \mathbb{Z}_a)$ and pad message m until $|m| \equiv 0 \bmod k$. Divide m into blocks $m = m_1 \| \dots \| m_\ell$, where $|m_i| = k$. Let $x_0 \leftarrow iv$. Compute $x_i \leftarrow E_k(x_{i-1})$ and $c_i \leftarrow m_i + x_i$. Let $c = c_1 \| \dots \| c_\ell$. The output is ciphertext (iv, c).

Decrypt(sk, iv, c): Divide c into ℓ blocks $c = c_1 \| \ldots \| c_\ell$. Let $x_0 \leftarrow iv$. Compute $x_i \leftarrow E_k(x_{i-1})$ and $m_i \leftarrow c_i - x_i$. Recover m by removing the padding.

Remark 2.1. Note that the CFB, CTR and OFB modes do not require K_1 to be invertible.

2.1.1.3 Statistical Models

When brute forcing rows, we cannot tell immediately if the decrypted text is correct or not. But we can statistically analyze the letters of the resulting text and check if they are reasonable enough. Using the frequencies of the recovered letters and the frequencies of the characters in original language, we can rank the rows according to their relevance to the ciphertext.

In order to rank[5] all possible rows for the decryption key, Yum and Lee [266] introduce a goodness-of-fit score function. Compared to the score functions presented in [42, 154], Yum and Lee's function describes the exact probability of the recovered plaintext. We briefly describe the goodness-of-fit score function in Algorithm 1.

Let E_K and D_K be the encryption and, respectively, decryption function of a cipher. Also, let $c \leftarrow E_K(m)$ be the given cryptogram and K' the key we want to rank. The goodness-of-fit function takes as input the letter frequency table $letter\_freq$ associated with the language m is written in (see Appendix A for some examples) and the letter frequency table *occurence* observed in $D_{K'}(c)$.

Algorithm 1. The goodness-of-fit score function.

Input: A vector of letter occurrences *occurrence*.
Output: The vector's goodness-of-fit score *score*.

```
1 Function goodness_of_fit(letter_freq, occurrence):
2     score ← 1;
3     for i ∈ [0, alphabet_size) do
4         score *= letter_freq[i]^occurrence[i] / occurrence[i]!
5     end
6     return score;
```

To automatically separate meaningful messages from random texts, we use an approach similar with the ones described in [137, 170]. When testing a list of strings for meaning, we first score each of them using Algorithm 2 and then output the highest scoring message.

The first and second inputs of the score function are a string *in* and the block frequency map (in our case either a digraph di_freq or a quadgraph $quad\_freq$ frequency map)

[5] according to their relevance to a given cryptogram

associated with the language we are interested in. The fourth variable $num\_of\_letters$ controls if we are observing digraphs (*i.e.* $num\_of\_letters = 2$) or quadgraph (*i.e.* $num\_of\_letters = 4$). When computing block frequency maps, some blocks may be missing entirely from the training corpus. To avoid assigning a likelihood of zero to these blocks, we use the *ad hoc* method found in [170][6].

Algorithm 2. The score function.

Input: A string in, the bound $number\_of\_rows$.
Output: The string's score $score$.

```
1  Function score_function(in, block_freq, block_freq, num_of_letters):
2      score ← 0;
3      for i ∈ [0, in.size() − num_of_letters) do
4          temp ← in.substr(i, num_of_letters);
5          if temp ∈ block_freq then
6              score += block_freq[temp];
7          end
8          else
9              score += block_default;
10         end
11     end
12     return score;
```

Remark 2.2. To ease description, all frequency tables/maps will be implicit when presenting algorithms, unless otherwise specified.

2.1.2 Ranking Functions

The first step in attacking the (affine) Hill cipher and the associated modes of operation is to rank all possible rows according to their relevance to a given cryptogram. In this section we describe the ranking functions latter used in the attacks presented in Section 2.1.3.

2.1.2.1 (Affine) ECB

In [266], the authors describe a ranking algorithm for the Hill cipher. We choose to present it in this section (Algorithm 3, red text) because it is tightly linked with the affine version that we introduce (Algorithm 3, blue text).

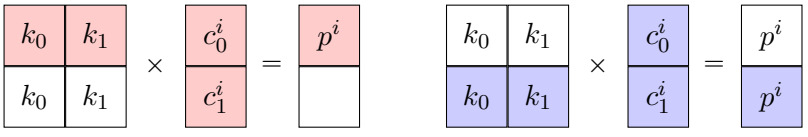
Let $matrix\_size = k = 2$ and let $enc = c$ be a Hill cipher cryptogram. We illustrate the influence of a given row on the decrypted plaintext p in Figure 2.1. We observe that if

[6] *i.e.* $block\_default \leftarrow \log_{10}(0.01/num\_of\_blocks)$, where the total number of blocks found in the training corpus is denoted by num_of_blocks

the first and second rows are equal we obtain the same letter p^i after decryption. Thus, is enough to decrypt the ciphertext using only the first row ($hill\_line\_decrypt$). Since we do not have duplicates, the resulting text msg is k times shorter than c. After decryption we compute the letter frequency observed in msg and use the $goodness\_of\_fit$ function to obtain the row's score. After all the rows have been ranked, we sort them in descending order according to their score. In the case of the affine Hill cipher the ranking algorithm is similar. The main difference is that instead of having to brute force k_0 and k_1, we also have to do an exhaustive search on k_2 (Figure 2.2). The algorithm for the generic case is given in Algorithm 3.

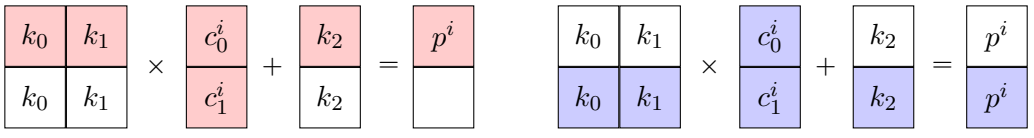
In some cases storing a vector of size a^k[7] might be troublesome. Thus, we further consider that $fit.size() = B$, where B is dependent on the available memory. Note that in this case fit must be sorted and when an element is inserted we first check if its score is higher than the lowest score from fit and if it is, the element replaces the lowest scoring element from fit.

We usually work with small values of $alphabet\_size$ and the $msg.size()$ and thus we consider the complexity of the $goodness\_of\_fit$ and of multiplication as $\mathcal{O}(1)$. Hence, the Hill version of Algorithm 3 performs $\mathcal{O}(a^k)$ $hill_line_decryption$s and sorts a vector of size B. So, it has a complexity of $\mathcal{O}(ka^k + B \log B)$. In the case of the affine Hill cipher, the only change is that we perform $\mathcal{O}(a^{k+1})$ $affine_hill_line_decryption$s. So, the complexity becomes $\mathcal{O}(ka^{k+1} + B \log B)$.



(A) Line 1. (B) Line 2.

FIGURE 2.1: Line propagation in ECB.



(A) Line 1. (B) Line 2.

FIGURE 2.2: Line propagation in affine ECB.

2.1.2.2 (Affine) CBC, CTR, CFB

Again, let $matrix\_size = 2$ and let enc be a Hill cipher cryptogram. The effect of a given row on the decrypted plaintext is shown in Figure 2.3 for CBC, in Figure 2.4 for

[7] a^{k+1} for the affine version

Algorithm 3. The algorithm for ranking all possible rows for (affine) ECB.

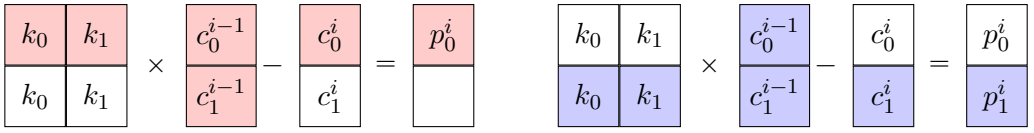
Input: The ciphertext enc.

Output: A vector fit containing all possible rows sorted by the goodness-of-fit score.

```
1  Function affine_hill_line_decrypt(conv, key_1, key_2):
2      msg_int[enc.size()/matrix_size] ← {0};
3      for i ∈ [0, conv.size()/matrix_size] do
4          for j ∈ [0, matrix_size] do
5              msg_int[i] ←
                (msg_int[i] + key_1[j] · conv[i · matrix_size + j]) mod alphabet_size;
6          end
7          msg_int[i] ← (msg_int[i] + key_2[i mod matrix_size]) mod alphabet_size;
8      end
9      return msg_int;
10 Function affine_ecb_rank(enc):
11     for key_1[0], ..., key_1[matrix_size − 1] ∈ [0, alphabet_size] do
12         for key_2 ∈ [0, alphabet_size] do
13             occurrence[alphabet_size] ← {0};
14             conv ← encode(enc);
15             msg_int ← hill_line_decrypt(enc, key_1);
16             msg_int ← affine_hill_line_decrypt(enc, key_1, key_2);
17             msg ← decode(msg_int)
18             for i ∈ [0, msg.size()] do
19                 occurrence[msg[i] − "a"]++;
20             end
21             occurrence.sort(); \\only for Algorithm 6;
22             score ← goodness_of_fit(letter_freq, occurrence);
23             fit.push_back((key_1, score));
24             fit.push_back((key_1, key_2, score));
25         end
26     end
27     fit.sort();
28     return fit;
```

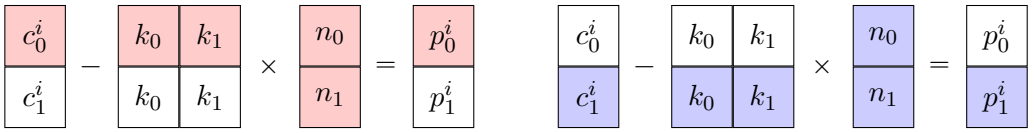
CTR and in Figure 2.5 for CFB. Compared to ECB, we can easily see that if the first and second row are identical the resulting letters are different. Thus, we need the full decryption of the Hill cipher to rank rows. After decryption, we break the resulting msg in two parts msg_0 and msg_1. The first part contains the letters in even positions and the second one the letters in odd positions. After we score each part, we store them in $fit[0]$ and, respectively, $fit[1]$. The last step is to sort the two vectors in descending order by score. The case of the affine Hill cipher is similar.

For the Hill modes attack, we perform $\mathcal{O}(a^k)$ decryptions, while for the affine version the number of decryptions is $\mathcal{O}(a^{k+1})$. Both algorithms sort k vectors of size B. Thus, the complexities are $\mathcal{O}(k^2a^k + kB\log B)$ and $\mathcal{O}(k^2a^{k+1} + kB\log B)$ for the Hill attack and, respectively, for the affine attack.



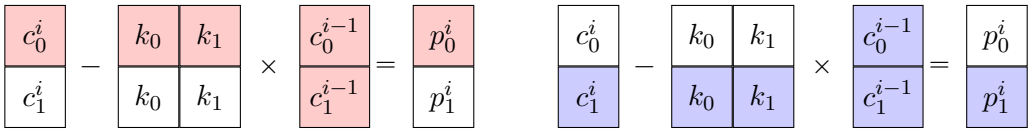
(A) Line 1. (B) Line 2.

FIGURE 2.3: Line propagation in CBC.



(A) Line 1. (B) Line 2.

FIGURE 2.4: Line propagation in CTR.



(A) Line 1. (B) Line 2.

FIGURE 2.5: Line propagation in CFB.

2.1.3 Message Recovering Attacks

After the ranking step is over, we can proceed to the recovering step. When searching for the original message a lot of random text is produced. To filter random messages from ones with meaning we use the *score_function* to score each message and we always output the highest scoring one.

2.1.3.1 (Affine) ECB

The authors of [42, 266] describe the message recovering algorithm for the Hill cipher, but they do not provide an automatic detection method for the original message. On the other hand, the authors of [154] trade-off success probability for an unique output. The gap is filled in [167]. We present the algorithm in this section (Algorithm 5, red text), instead of Section 2.1.1, because of its link to the affine version we introduce (Algorithm 5, blue text). Due to better results in practice, in Algorithm 5 we use a different scoring function[8] than the one from [167][9]. Also, compared to [167], we only output the highest scoring message without lowering the success probability.

[8] based on quadgraphs
[9] based on the index of coincidence

Algorithm 4. The algorithm for ranking all possible rows for (affine) CBC, CTR, CFB.

Input: The ciphertext enc and the initialization vector iv.

Output: A family of vectors fit containing all possible rows sorted by the goodness-of-fit score.

```
1  Function affine_mode_rank(enc, iv):
2      for a[0], ..., a[matrix_size − 1] ∈ [0, alphabet_size] do
3          for b ∈ [0, alphabet_size] do
4              occurrence[matrix_size][alphabet_size] ← {0};
5              for i ∈ [0, matrix_size] do
6                  for j ∈ [0, matrix_size] do
7                      key_1[i][j] ← a[j];
8                  end
9                  key_2[i] ← b;
10             end
11             conv ← encode(enc);
12             msg_int ← mode_decrypt(enc, iv, key_1);
13             msg_int ← affine_mode_decrypt(enc, iv, key_1, key_2);
14             msg ← decode(msg_int)
15             for i ∈ [0, msg.size()/matrix_size] do
16                 for j ∈ [0, matrix_size] do
17                     occurrence[j][msg[i · matrix_size + j] − "a"]++;
18                 end
19             end
20             for i ∈ [0, matrix_size] do
21                 occurrence[i].sort(); \\only for Algorithm 8;
22                 score ← goodness_of_fit(letter_freq, occurrence[i]);
23                 fit[i].push_back((a, score));
24                 fit[i].push_back((a, b, score));
25             end
26         end
27     end
28     for i ∈ [0, matrix_size] do
29         fit[i].sort();
30     end
31     return fit;
```

After ranking all possible rows, we need to find the decryption key's rows ($check\_variants$) and their order ($check\_variant$). Hence, Algorithm 5 checks all possible row combinations with index less than $number\_of\_rows = B$. Note that the success probability is dependent on $number\_of\_rows$[10]. After selecting k rows from fit, we test all possible row permutations[11], decrypt enc and rank the result. If one of the decrypted texts has a higher score than the stored message $global\_msg$, we overwrite $global\_msg$ and update $global\_score$. The main differences between the Hill cipher attack and the affine Hill

[10] see Section 2.1.4 for the experimental results

[11] σ_i denotes the ith permutation of length mat_size

cipher attack are: the call to the affine ranking algorithm, the creation of k_2 and the call to the affine decryption algorithm.

Algorithm 5. The algorithm for breaking (affine) ECB.

Input: The ciphertext *enc*, the bound $number\_of\_rows$.
Output: The best possible message $global\_msg$ and its associated score $global\_score$.

```
1  Function check_variant(enc, rows, &global_score, &global_msg):
2      best_score ← −∞;
3      for i ∈ [0, matrix_size!) do
4          for s ∈ [0, matrix_size) do
5              for t ∈ [0, matrix_size) do
6                  key_1[s][t] ← rows[σ_i[s]].key_1[t];
7              end
8              key_2[s] ← rows[σ_i[s]].key_2;
9          end
10         trial_msg ← hill_decrypt(enc, key_1);
11         trial_msg ← affine_hill_decrypt(enc, key_1, key_2);
12         trial_score ← score_function(trial_msg, quad_freq, quad_default, 4);
13         if trial_score > best_score then
14             best_score ← trial_score;
15             best_msg ← trial_msg;
16         end
17     end
18     if best_score > global_score then
19         global_score ← best_score;
20         global_msg ← best_msg;
21     end
22 Function check_variants(enc, fit, number_of_rows):
23     global_score ← −∞;
24     global_msg ← "";
25     for i_0 ∈ [0, number_of_rows) do
26         for i_1 ∈ [i_0 + 1, number_of_rows) do
27             ...
28             for i_matrix_size−1 ∈ [i_matrix_size−2 + 1, number_of_rows) do
29                 trial_rows ← ∅;
30                 for j ∈ [0, matrix_size) do
31                     trial_rows.push_back(fit[i_j]);
32                 end
33                 check_variant(enc, trial_rows, global_score, global_msg);
34             end
35         end
36     end
37     return (global_score, global_msg);
38 Function affine_ecb_attack(enc, number_of_rows):
39     fit ← affine_ecb_rank(enc);
40     return check_variants(enc, fit, number_of_rows);
```

For the same reasons as in Section 2.1.2.1, we further consider the complexity of the $score\_function$ as $\mathcal{O}(1)$. After the row ranking step, both message recovering algorithms perform $\mathcal{O}(B!/(B-k)!)$ decryptions. Thus, the complexities for the Hill attack and for the affine attack are $\mathcal{O}(ka^k + B\log B + k^2B!/(B-k)!)$ and, respectively, $\mathcal{O}(ka^{k+1} + B\log B + k^2B!/(B-k)!)$.

2.1.3.2 Affine ECB (Second Approach)

In [266], the authors propose a ranking method for the Hill cipher with unknown encoding and decoding functions. The basic idea is that encoding functions act as substitution ciphers and thus leave letter distributions intact. According to their method, to score a row one needs to sort in ascending order both $letter\_freq$ and $occurrence$ and then use Algorithm 1 to obtain the row's score. Note that Yum and Lee do not provide a message recovering algorithm.

The affine Hill cipher can be seen as the composition of a Hill cipher and a Vigenère cipher. Thus, we use Yum and Lee's ranking method to find K_1''s rows (ecb_rank), decrypt the cryptogram using the trial K_1' ($hill\_decrypt$) and then use a Vigenère message recovery algorithm ($break\_vigenere$) to find K_2'. This method is formally described in Algorithm 6. Note that $break\_vigenere$[12] returns the score of the $trial\_msg$. Unfortunately, we can not use only this score to filter messages. For example, when $k = 2$ the texts `easy` and `aeys` have the same $trial\_score$. Hence, we use a second scoring system based on quadgrams to differentiate between trial messages with the same score. Note that the only difference between $check\_variants\_2$ and $check\_variants$ is that the latter is using the $check\_variant\_2$ function.

We consider the complexity of $break\_vigenere$ as being $\mathcal{O}(1)$, since it is linear in the cryptogram's size. Then, the complexity of the second algorithm is $\mathcal{O}(ka^k + B\log B + k^2B!/(B-k)!)$.

2.1.3.3 (Affine) CBC, CTR, CFB

The main difference between ECB and the other modes is that after the ranking step is over, in the former case we know the exact position of the key rows. Thus, in Algorithm 7 we iterate over all rows ($check\_variants\_mode$), decrypt the cryptogram and then score the result ($check\_variant\_mode$).

[12] see Appendix B for a concrete algorithm

Algorithm 6. The algorithm for breaking affine ECB (second approach).

Input: The ciphertext enc, the bound $number\_of\_rows$.
Output: The best possible message $global\_msg$ and its associated score $global\_score$.

Function $check\_variant\_2(enc, rows, \&\, global\_score, \&\, global\_msg)$**:**
  $best\_score \leftarrow -\infty$;
  for $i \in [0, matrix\_size!)$ **do**
    for $s \in [0, matrix\_size)$ **do**
      for $t \in [0, matrix\_size)$ **do**
        $key_1[s][t] \leftarrow rows[\sigma_i[s]].key_1[t]$;
      end
    end
    $hill\_msg \leftarrow hill\_decrypt(enc, key_1)$;
    $(trial\_score, trial\_msg) \leftarrow break\_vigenere(hill\_msg)$;
    if $trial\_score > best\_score$ **then**
      $best\_score \leftarrow trial\_score$;
      $best\_msg \leftarrow trial\_msg$;
    end
    if $trial\_score == best\_score$ **then**
      $first\_quad\_score \leftarrow score\_function(best\_msg, quad\_freq, quad\_default, 4)$;
      $second\_quad\_score \leftarrow score\_function(trial\_msg, quad\_freq, quad\_default, 4)$;
      if $second\_quad\_score > first\_quad\_score$ **then**
        $best\_msg \leftarrow trial\_msg$;
      end
    end
  end
  if $best\_score > global\_score$ **then**
    $global\_score \leftarrow best\_score$;
    $global\_msg \leftarrow best\_msg$;
  end
Function $affine\_ecb\_attack\_2(enc, number\_of\_rows)$**:**
  $fit \leftarrow ecb\_rank(enc)$;
  return $check\_variants\_2(enc, fit, number\_of\_rows)$;

The $check\_variants\_mode$ function performs $\mathcal{O}(B^k)$ decryptions. Thus, Algorithm 7's complexity for the Hill based modes attack and for the affine versions is $\mathcal{O}(k^2a^k + kB\log B + k^2B^k)$ and, respectively, $\mathcal{O}(k^2a^{k+1} + kB\log B + k^2B^k)$.

2.1.3.4 Affine CBC, CTR, CFB (Second Approach)

As in the case of the affine Hill cipher, attacking a affine based mode can be interpreted as attacking a Hill-Vigenère cipher mode of operation. We present this complementary attack in Algorithm 8. Note that the only difference between $check\_variants\_mode$ and

Algorithm 7. The algorithm for breaking (affine) CBC, CTR, CFB.

Input: The ciphertext enc, the initialization vector iv, the bound $number\_of\_rows$.

Output: The best possible message $global\_msg$ and its associated score $global\_score$.

Function $check\_variant\_mode(enc, iv, rows, \&\, global\_score, \&\, global\_msg)$**:**
  for $s \in [0, matrix\_size)$ **do**
    for $t \in [0, matrix\_size)$ **do**
      $key_1[s][t] \leftarrow rows[s].a[t]$;
    end
    $key_2[s] \leftarrow rows[s].b$;
  end
  $trial\_msg \leftarrow mode\_decrypt(enc, iv, key_1)$;
  $trial\_msg \leftarrow affine\_mode\_decrypt(enc, iv, key_1, key_2)$;
  $trial\_score \leftarrow score\_function(trial\_msg, quad\_freq, quad\_default, 4)$;
  if $trial\_score > global\_score$ **then**
    $global\_score \leftarrow trial\_score$;
    $global\_msg \leftarrow trial\_msg$;
  end
Function $check\_variants\_mode(enc, fit, number\_of\_rows)$**:**
  $global\_score \leftarrow -\infty$;
  $global\_msg \leftarrow$ "";
  for $i_0 \in [0, number\_of\_rows)$ **do**
    for $i_1 \in [0, number\_of\_rows)$ **do**
      $\cdots$
      for $i_{matrix\_size-1} \in [0, number\_of\_rows)$ **do**
        $trial\_rows \leftarrow \varnothing$;
        for $j \in [0, matrix\_size)$ **do**
          $trial\_rows.push\_back(fit[j][i_j])$;
        end
        $check\_variant\_mode(enc, iv, trial\_rows, global\_score, global\_msg)$;
      end
    end
  end
  return $(global\_score, global\_msg)$;
Function $affine\_mode\_attack(enc, number\_of\_rows)$**:**
  $fit \leftarrow affine\_mode\_rank(enc, iv)$;
  return $check\_variants\_mode(enc, iv, fit, number\_of\_rows)$;

$check\_variants\_mode\_2$ is that the former uses the $check\_variant\_mode\_2$ function. The time complexity of Algorithm 8 is $\mathcal{O}(k^2a^k + kB\log B + k^2B^k)$.

Algorithm 8. The algorithm for breaking affine CBC, CTR, CFB (second approach).

Input: The ciphertext enc, the initialization vector iv, the bound $number\_of\_rows$.

Output: The best possible message $global\_msg$ and its associated score $global\_score$.

```
1  Function check_variant_mode_2(enc, iv, rows, &global_score, &global_msg):
2      for s ∈ [0, matrix_size) do
3          for t ∈ [0, matrix_size) do
4              key_1[s][t] ← rows[s].a[t];
5          end
6      end
7      hill_msg ← mode_decrypt(enc, iv, key_1);
8      (trial_score, trial_msg) ← break_vigenere(hill_msg);
9      if trial_score > global_score then
10         global_score ← trial_score;
11         global_msg ← trial_msg;
12     end
13 Function affine_mode_attack_2(enc, number_of_rows):
14     fit ← mode_rank(enc, iv);
15     return check_variants_mode_2(enc, iv, fit, number_of_rows);
```

2.1.4 Experimental Results

We implemented Algorithms 5 to 8 in order to see the relation between B and the algorithms' success probability[13]. To see the influence of the message's native language on the attack algorithms' recovery rate, we tested this type of relation for eight languages: Danish (DN), English (EN), Finnish (FN), French (FR), German (GE), Polish (PL), Spanish (SP) and Swedish (SW). We also computed the running time of Algorithms 5 to 8 for the English language and $k = 2$ (Section 2.1.4.5). Besides providing the reader with some benchmarks, we also wanted to have a precise comparison[14] between the two affine attack approaches.

In our implementations, frequency tables have $a = 26$ values and are derived from the frequencies provided in [170]. For completeness, we describe the tables in Appendix A. The quadgrams for the English language are downloaded from [170], while the digraph[15] frequencies are computed from the quadgraph map. The algorithm for breaking Vigenère is given in Appendix B.

For computing the success probability we used 100 texts with 100 letters (without diacritical marks) for each language. Each text was encrypted with a different key(s)/initialization vector/counter. The texts are taken from news items found in the Leipzig Corpora

[13] We refer the reader to Sections 2.1.4.2 to 2.1.4.4 for the results.

[14] that takes into account the hidden constants found in asymptotic notations

[15] If $abcd$ is a quadgraph, we consider ac as a digraph.

Collection [119]. The keys, initialization vectors and counters are generated using the default generator found in the GMP library [19]. When invertible keys were needed, we computed the inverse using the Armadillo library [217] and tested if the determinant is coprime with 26.

2.1.4.1 Unicity Distance of a Cipher

When analyzing the experimental results, the reader will observe different message recovery rates for different languages. These differences arise from distinct unicity distances[16] for distinct languages. The exact formula for the unicity distance when $a = 26$ is $\log_2 26^k/(\log_2 26 - H)$, where H is the language's entropy. Note that in our case the unicity distance is computed for one key row and we estimated the entropy from the frequency tables provided in Appendix A. The results for the unicity distance are provided in Table 2.2. We can see that in the case of the Polish language we need more letters per row than for the Finnish language. This gap will be more pronounced when determining the message recovery rates.

Language	$k = 2$	$k = 3$	$k = 4$
Danish	15.4323	23.1485	30.8647
English	18.2180	27.3270	36.4359
Finnish	12.0307	18.0460	24.0614
French	13.3713	20.0569	26.7425
German	15.6257	23.4386	31.2515
Polish	22.3918	33.5878	44.7837
Spanish	13.7891	20.6836	27.5781
Swedish	16.4837	24.7256	32.9674

TABLE 2.2: Unicity distance.

[16] The minimum ciphertext length required to determine the secret key almost uniquely.

2.1.4.2 Hill Modes of Operation Message Recovery Rates

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	2	94	93	100	96	95	84	96	95
	4	99	100	100	98	100	91	100	100
CBC	1	95	95	100	99	97	84	99	99
	2	99	99	100	100	100	90	100	100
CTR	1	96	93	100	96	98	87	100	98
	2	99	98	100	99	100	90	100	100
CFB	1	97	92	99	96	95	87	98	98
	2	100	99	100	100	99	91	100	100

TABLE 2.3: Number of recovered messages for the Hill modes of operation when $k = 2$.

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	8	88	59	97	90	71	22	87	80
	16	95	77	100	95	86	45	96	94
	32	97	87	100	98	94	68	99	99
CBC	4	86	57	99	92	71	18	91	78
	8	93	68	99	96	80	34	96	86
	16	96	80	100	96	89	55	97	96
CTR	4	64	40	84	65	46	11	68	45
	8	80	59	94	87	67	19	83	66
	16	91	75	97	93	80	48	92	77
CFB	4	85	53	99	90	73	12	89	78
	8	93	66	99	94	81	36	94	87
	16	96	79	100	97	91	52	96	96

TABLE 2.4: Number of recovered messages for the Hill modes of operation when $k = 3$.

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	512	78	48	97	89	72	10	85	74
	1024	88	65	98	91	89	19	94	86
	2048	95	80	99	95	94	39	95	93
CBC	32	78	50	97	89	69	13	88	72
	64	87	67	99	91	86	21	93	84
	128	93	78	99	95	94	45	95	93
CTR	32	71	37	91	77	55	6	80	64
	64	87	58	97	90	79	21	90	83
	128	93	75	100	95	94	40	99	88
CFB	32	78	48	97	88	69	14	86	73
	64	87	65	98	91	85	18	92	85
	128	93	75	99	95	95	45	94	95

TABLE 2.5: Number of recovered messages for the Hill modes of operation when $k = 4$.

2.1.4.3 Affine Hill Modes of Operation Message Recovery Rates (First Approach)

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	2	89	80	100	90	88	54	93	92
	4	97	94	100	98	99	79	98	99
	8	99	99	100	99	99	87	99	100
CBC	1	93	85	100	99	85	57	96	93
	2	97	88	100	99	93	68	98	100
	4	99	95	100	99	99	78	100	100
CTR	1	92	72	100	93	90	48	96	95
	2	97	88	100	96	98	68	99	99
	4	98	97	100	99	99	78	100	100
CFB	1	89	80	100	95	91	54	98	93
	2	97	92	100	98	97	69	100	99
	4	99	97	100	99	99	83	100	100

TABLE 2.6: Number of recovered messages for the affine Hill modes of operation when $k = 2$.

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	32	70	43	97	86	49	3	85	63
	64	84	50	99	91	62	11	87	75
	128	93	65	99	93	79	21	94	88
CBC	32	71	40	98	86	47	5	83	61
	64	82	50	99	93	65	11	90	74
	128	90	65	99	93	78	25	95	97
CTR	32	35	13	56	40	19	3	37	18
	64	58	28	85	63	36	6	60	45
	128	81	49	98	82	59	13	83	77
CFB	32	70	38	97	87	50	3	83	74
	64	84	49	99	93	64	8	89	86
	128	91	63	99	93	77	23	94	96

TABLE 2.7: Number of recovered messages for the affine Hill modes of operation when $k = 3$.

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	16384	82	53	98	90	79	14	89	79
	32768	92	69	99	93	93	26	94	88
	65536	96	83	100	95	95	54	96	94
CBC	16384	80	53	98	89	76	14	88	78
	32768	89	69	99	93	92	27	94	87
	65536	96	80	100	95	95	61	96	93
CTR	16384	77	46	95	86	63	11	86	74
	32768	87	66	98	92	89	26	92	85
	65536	95	79	100	97	95	53	96	92
CFB	16384	81	53	98	89	76	15	88	77
	32768	90	68	99	93	92	27	94	87
	65536	96	81	100	95	95	59	96	93

TABLE 2.8: Number of recovered messages for the affine Hill modes of operation when $k = 4$.

2.1.4.4 Affine Hill Modes of Operation Message Recovery Rates (Second Approach)

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	128	73	59	59	40	70	7	28	72
	256	92	83	98	97	90	32	98	89
	512	100	98	100	100	99	100	100	100
CBC	16	84	35	99	96	82	2	97	63
	32	95	57	100	97	92	4	100	87
	64	98	84	100	98	96	10	100	95
CTR	16	65	33	94	96	68	1	96	49
	32	92	58	100	97	87	5	100	82
	64	99	80	100	98	96	12	100	95
CFB	16	79	39	99	95	80	2	97	63
	32	94	60	100	98	91	6	100	86
	64	98	80	100	98	95	9	100	95

TABLE 2.9: Number of recovered messages for the secret coding affine Hill modes of operation when $k = 2$.

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	4096	24	25	63	71	44	0	71	32
	8192	53	54	97	98	71	4	94	64
	16384	99	93	100	100	98	89	100	97
CBC	4096	34	34	96	91	55	0	92	45
	8192	68	62	100	99	79	3	100	76
	16384	100	96	100	100	98	47	100	96
CTR	4096	30	36	88	88	54	0	91	44
	8192	68	60	100	99	81	3	100	77
	16384	100	96	100	100	98	41	100	100
CFB	4096	34	35	96	92	56	0	92	47
	8192	65	62	100	99	82	3	100	74
	16384	100	95	100	100	98	47	100	100

TABLE 2.10: Number of recovered messages for the secret coding affine Hill modes of operation when $k = 3$.

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	200000	46	32	86	76	30	0	82	53
	300000	73	64	96	98	71	2	95	78
	400000	93	86	100	100	91	16	100	94
CBC	200000	54	38	89	83	37	0	91	56
	300000	76	67	97	99	76	1	98	82
	400000	93	87	100	100	91	5	100	94
CTR	200000	54	41	90	84	38	0	85	56
	300000	76	67	96	95	73	2	98	82
	400000	95	86	100	100	94	4	100	95
CFB	200000	54	38	90	83	38	0	91	54
	300000	76	66	97	99	76	1	97	81
	400000	94	87	100	100	91	6	100	94

TABLE 2.11: Number of recovered messages for the secret coding affine Hill modes of operation when $k = 4$.

2.1.4.5 Running time

In this section we provide some benchmarks for Algorithms 5 to 8. The algorithms were run on a CPU Intel i7-4790 4.00 GHz and compiled with GCC with the O3 flag activated and the *omp_get_wtime()* function [15] was used to compute the running times. Due to resource constrains, we stopped the experiments at $k = 3$ for the Hill attacks and at $k = 2$ for the affine attacks. To obtain a fair comparison, when computing the running times, we used higher B values than the one presented in Sections 2.1.4.2 to 2.1.4.4. We present the exact margins in Table 2.12.

Mode	Hill ($k = 2$)	Afine Hill (1) ($k = 2$)	Afine Hill (2) ($k = 2$)	Hill ($k = 3$)
ECB	4 (100%)	8 (99%)	512 (98%)	128 (97%)
CBC	2 (99%)	4 (95%)	256 (96%)	128 (95%)
CTR	2 (98%)	4 (97%)	256 (98%)	128 (96%)
CFB	2 (99%)	4 (97%)	256 (98%)	128 (96%)

TABLE 2.12: The threshold B and the corresponding success probability for the English language.

In Table 2.13, the second and third columns contain the total time necessary to recover 100 independent texts, the fourth and fifth columns the total time necessary to recover 8 texts. It is clear from the presented results that the first approach (Affine Hill (1)) has significantly lower running times than the second approach (Affine Hill (2)). Note that in the case of the second approach the difference between the ECB attack and the rest

of the attacks is due to the extra *score_function* calls made when the *trial_score* is equal to the *best_score*.

Mode	Hill ($k = 2$)	Afine Hill (1) ($k = 2$)	Afine Hill (2) ($k = 2$)	Hill ($k = 3$)
ECB	0.94057	23.1658	1805.98	1415.60
CBC	1.75324	45.4769	379.762	1502.20
CTR	1.75827	45.9883	374.439	1423.39
CFB	1.75271	48.5864	360.428	1509.62

TABLE 2.13: Running times of Algorithms 5 to 8.

Let $k = 2$. To see if the chosen bounds have the same success rate for other texts, we encrypted 1000 independent texts[17] and then we run Algorithms 5 and 7. The number of plaintexts recovered is presented in Table 2.14. We can see that for the Hill based modes the success probabilities are almost the same, while for the affine versions the probabilities are a little lower than the initial estimations.

Cipher	ECB	CBC	CTR	CFB
Hill	995	987	982	982
Afine Hill (1)	970	956	945	953

TABLE 2.14: Success rates for Algorithms 5 and 7 when $k = 2$.

2.1.5 Future Work

The row ranking algorithms perform the same instructions for disjoint rows. Thus, an interesting implementation direction is to parallelize Algorithms 3 and 4. The recovering algorithms also perform the same instructions, but for independent keys. Hence, Algorithms 5 and 7 can also be parallelized.

Another possible speed-up is to parallelize the algorithm presented [167] for the Hill cipher. Note that this speed-up can also be applied to the Hill CBC mode. From a theoretical point of view, it would be interesting to see if the Leap *et.al.*'s algorithm can be tweaked to work for the affine Hill cipher. If it can be tweaked we might obtain faster decryption times for the affine Hill and the corresponding CBC mode.

A time-memory trade-off attack for the Hill cipher is presented in [178]. Thus, it might be interesting to see if this attack can be adapted to the affine version and to the (affine) modes of operation versions. From an implementation point of view, it might worth seeing if McDevitt *et.al.*'s attack can be parallelized.

In [266], the authors provide a ranking algorithm when the encoding and decoding functions are unknown, but they do not describe a message recovery algorithm. This cipher

[17] different from the 100 texts used for computing the bounds

can be seen as a composition of a substitution cipher, a Hill cipher and a second substitution cipher. Note that the two substitution ciphers do not necessarily have the same key. A generic version of the secret coding cipher can be obtained by combining a generic Vigenère cipher[18], a Hill cipher and a second generic Vigenère cipher. Note that in this case Yum and Lee's ranking algorithm still works. Hence, another possible research direction is to find message recovery algorithms[19] for this generic cipher.

In [145], Hill introduces a variation of the affine Hill cipher in which the elements of the key matrix are matrices. Thus, an interesting problem is to study the impact of the message recovering algorithms on the version presented in [145].

2.2 Grain Cipher Family

The Grain family of stream ciphers consists of four instantiations Grain v0 [140], Grain v1 [141], Grain-128 [139] and Grain-128a [211]. Grain v1 is a finalist of the hardware-based eSTREAM portfolio [4], a competition for choosing both hardware and software secure and efficient stream ciphers.

The design of the Grain family of ciphers includes an LFSR. The loading of the LFSR consists of an initialization vector (IV) and a certain string of bits P whose lengths and structures depend on the cipher's version. Following the terminology used in [39], we consider the IV as being padded with P. Thus, throughout this section, we use the term *padding* to denote P. Note that Grain v1 and Grain-128 make use of *periodic* IV padding and Grain-128a uses *aperiodic* IV padding.

A series of attacks against the Grain family padding techniques appeared in the literature [38, 39, 64, 162] during the last decade. In the light of these attacks, we propose the first security analysis[20] of generic IV padding schemes for Grain ciphers in the *periodic* as well as the *aperiodic* cases.

In this context, the concerns that arise are closely related to the security impact of various parameters of the padding, such as the position and structure of the padding block. Moreover, we consider both *compact* and *fragmented* padding blocks in our study. We refer to the original padding schemes of the Grain ciphers as being compact (*i.e.* a single padding block is used). We denote as fragmented padding the division of the padding block into smaller blocks of equal length[21].

[18] By a generic Vigenère cipher we understand a Vigenère cipher with random alphabets.
[19] that might use Yum and Lee's ranking algorithm
[20] against slide attacks
[21] we consider these smaller blocks as being spread among the linear feedback register's data

By examining the structure of the padding and analyzing its compact and especially fragmented versions, we actually study the idea of extending the key's life. The latter could be achieved by introducing a variable padding according to suitable constraints. Hence, the general question that arises is the following: *what is to be loaded in the LFSRs of Grain ciphers in order to obtain secure settings?*. Note that our study is preliminary, taking into account only slide attacks. We consider other types of attacks as future work.

We stress that finding better attacks than the ones already presented in the literature is outside the scope of this section, as our main goal is to establish sound personalized versions of the Grain cipher. Hence, our work does not have any immediate implication towards breaking any cipher of the Grain family. Nevertheless, our observations become meaningful either in the lightweight cryptography scenario or in the case of an enhanced security context (e.g. secure government applications).

Lightweight cryptography lies at the crossroad between cryptography, computer science and electrical engineering. Thus, trade-offs between performance, security and cost must be considered. Given such constraints and the fact that embedded devices operate in hostile environments, there is an increasing need for new and varied security solutions, mainly constructed in view of the current ubiquitous computing tendency. As the Grain family lies precisely within the lightweight primitives' category, we believe that the study presented in the current section is of interest for the industry and, especially, government organizations.

When dealing with security devices for which the transmission and processing of the IV is neither so costly nor hard to handle (e.g. the corresponding communication protocols easily allow the transmission), shrinking the padding up to complete removal might be considered. More precisely, we suggest the use of a longer IV in such a context in order to increase security. Moreover, many Grain-type configurations could be obtained if our proposed padding schemes are used. Such configurations could be considered as personalizations of the main algorithm and, if the associated parameters are kept secret, the key's life can be extended.

2.2.1 Preliminaries

Conventions. During the following, capital letters will denote padding blocks and small letters will refer to certain bits of the padding.

Grain is a hardware-oriented stream cipher initially proposed by Hell, Johansson and Meier [140] and whose main building blocks are an n bit *linear feedback shift register* (LFSR), an n bit *non-linear feedback shift register* (NFSR) and an *output function*.

Because of a weakness in the output function, a key recovery attack [52] and a distinguishing attack [155] on Grain v0 were proposed. To solve these security issues, Grain v1 [141] was introduced. Also, Grain-128 [139] was proposed as a variant of Grain v1. Grain-128 uses 128-bit keys instead of 80-bit keys. Grain 128a [211] was designed to address cryptanalysis results [34, 92, 91, 159, 231] against the previous version. Grain 128a offers optional authentication. We stress that, in this paper, we do not address the authentication feature of Grain-128a.

Let $X_i = [x_i, x_{i+1}, \ldots, x_{i+n-1}]$ denote the state of the NFSR at time i and let $g(x)$ be the nonlinear feedback polynomial of the NFSR. $g(X_i)$ represents the corresponding update function of the NFSR. In the case of the LFSR, let $Y_i = [y_i, y_{i+1}, \ldots, y_{i+n-1}]$ be its state, $f(x)$ the linear feedback polynomial and $f(Y_i)$ the corresponding update function. The filter function $h(X_i, Y_i)$ takes inputs from both the states X_i and Y_i.

We shortly describe the generic algorithms KLA, KSA and PRGA below. As KSA is invertible, a state $S_i = X_i \| Y_i$ can be rolled back one clock to S_{i-1}. We further refer to the transition function from S_i to S_{i-1} as KSA^{-1}.

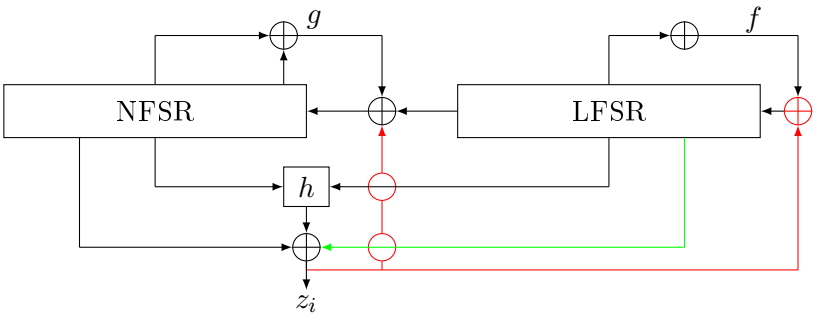


FIGURE 2.6: Output generator and key initialization of Grain ciphers.

Key Loading Algorithm (KLA). The Grain family uses an n-bit key K, an m-bit initialization vector IV with $m < n$ and some fixed padding $P \in \{0,1\}^{\alpha}$, where $\alpha = n - m$. The key is loaded in the NFSR, while the pair (IV, P) is loaded in the LFSR using a one-to-one function further denoted as $\texttt{Load}_{IV}(IV, P)$.

Key Scheduling Algorithm (KSA). After running KLA, the output[22] z_i is XOR-ed to both the LFSR and NFSR update functions, *i.e.*, during one clock the LFSR and the NFSR bits are updated as $y_{i+n} = z_i + f(Y_i)$, $x_{i+n} = y_i + z_i + g(X_i)$.

[22] during one clock

Pseudorandom Keystream Generation Algorithm (PRGA). After performing KSA routine for $2n$ clocks, z_i is no longer XOR-ed to the LFSR and NFSR update functions, but it is used as the output keystream bit. During this phase, the LFSR and NFSR are updated as $y_{i+n} = f(Y_i)$, $x_{i+n} = y_i + g(X_i)$.

Figure 2.6 depicts an overview of KSA and PRGA. Common features are depicted in black. In the case of Grain v1, the pseudorandom keystream generation algorithm does not include the green path. The red paths correspond to the key scheduling algorithm.

The corresponding parameters of Grain v1 are described in Appendix C, while Grain-128 is tackled in Appendix D and Grain-128a in Appendix E. The appendices also include the $\texttt{Load}_{\texttt{IV}}$ functions and the KSA^{-1} algorithms for all versions.

Security Model. In the *Chosen IV - Related Key* setting (according to [39, Section 2.1]), an adversary is able to query an encryption oracle (which has access to the key K) in order to obtain valid ciphertexts. For each query i, the adversary can choose the oracle's parameters: an initialization vector IV_i, a function $\mathcal{F}_i : \{0,1\}^n \to \{0,1\}^n$ and a message m_i. The oracle encrypts m_i using the Key-IV pair $(\mathcal{F}_i(K), IV_i)$. The adversary's task is to distinguish the keystream output from a random stream.

Assumptions. Based on the results of the experiments we conducted, we further assume that the output of KSA, KSA^{-1} and PRGA is independently and uniformly distributed. More precisely, all previous algorithms were statistically tested applying the NIST Test Suite [13]. During our experiments we used the following setup:

1. X_i is a randomly generated n-bit state using the GMP library [19];
2. Y_i'' is either $0^{2\alpha}$ or $1^{2\alpha}$;
3. $Y_i = Y_i' \| Y_i''$, where Y_i' is a randomly generated $(m-\alpha)$-bit state using the GMP library.

2.2.2 Generic Grain Attacks

As already mentioned in Section 2.2.1, the Grain family uses an NFSR and a nonlinear filter (which takes input from both shift registers) to introduce nonlinearity. If after the initialization process, the LFSR is in an all zero state, only the NFSR is actively participating to the output. As already shown in the literature, NFSRs are vulnerable to distinguishing attacks [52, 267, 159].

Weak Key-IV pair. If the LFSR reaches the all zero state after $2n$ clocks we say that the pair (K, IV) is a *weak Key-IV pair*. An algorithm which produces weak Key-IV pairs for Grain v1 is presented in [267]. We refer the reader to Algorithm 9 for a generalization of this algorithm to any of the Grain ciphers.

Given a state V, we define it as `valid` if there exists an $IV \in \{0,1\}^m$ such that $\texttt{Load}_{IV}(IV, P) = V$, where P is the fixed padding. We further use a function $\texttt{Extract}_{IV}(V)$ which is the inverse of $\texttt{Load}_{IV}(\cdot, P)$. The probability to obtain a weak Key-IV pair by running Algorithm 9 is $1/2^{\alpha}$.

Algorithm 9. Generic Weak Key-IV Attack.

Output: A Key-IV pair (K', IV')

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^n$ and let $V \in \{0,1\}^n$ be the zero LFSR state $(0, ..., 0)$
  Run $\text{KSA}^{-1}(K \| V)$ routine for $2n$ clocks and produce state $S' = K' \| V'$
  if V' is `valid` **then**
    Set $s \leftarrow 1$ and $IV' \leftarrow \texttt{Extract}_{IV}(V')$
    return (K', IV')
  end
end

A refined version of the attack from [267] is discussed in [38] and generalized in Algorithm 10. The authors of [38] give precise differences between keystreams generated using the output of Algorithm 10 for Grain v1 (see Theorem 2.1), Grain-128 (see Theorem 2.2) and Grain-128a (see Theorem 2.3).

Theorem 2.1. For Grain v1, two initial states S_0 and $S_{0,\Delta}$ which differ only in the 79^{th} position of the LFSR, produce identical output bits in 75 specific positions among the initial 96 key stream bits obtained during the PRGA.

Remark 2.3. More precisely, the 75 positions are the following ones:

$$k \in [0, 95] \backslash \{15, 33, 44, 51, 54, 57, 62, 69, 72, 73, 75, 76, 80, 82, 83, 87, 90, 91, 93 - 95\}.$$

Theorem 2.2. For Grain-128, two initial states S_0 and $S_{0,\Delta}$ which differ only in the 127^{th} position of the LFSR, produce identical output bits in 112 specific positions among the initial 160 key stream bits obtained during the PRGA.

Remark 2.4. More precisely, the 112 positions are the following ones:

$$k \in [0, 159] \backslash \{32, 34, 48, 64, 66, 67, 79 - 81, 85, 90, 92, 95, 96, 98, 99, 106, 107, 112, 114, 117, 119, 122, 124 - 126, 128, 130 - 132, 138, 139, 142 - 146, 148 - 151, 153 - 159\}.$$

Theorem 2.3. For Grain-128a, two initial states S_0 and $S_{0,\Delta}$ which differ only in the 127^{th} position of the LFSR, produce identical output bits in 115 specific positions among the initial 160 key stream bits obtained during the PRGA.

Remark 2.5. More precisely, the 115 positions are the following ones:

$$k \in [0, 159] \backslash \{33, 34, 48, 65 - 67, 80, 81, 85, 91, 92, 95, 97 - 99, 106, 107, 112, 114, 117, 119, 123 - 125, 127 - 132, 138, 139, 142 - 146, 149 - 151, 154 - 157, 159\}.$$

Algorithm 10. Search for Key-IV pairs that produce almost similar initial keystream.

Input: An integer $r \in \{0, 2n\}$
Output: Key-IV pairs (K, IV) and (K', IV')

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^n$ and $IV \in_R \{0,1\}^m$
  Run KSA$(K \| IV)$ routine for $2n$ clocks to obtain an initial state $S_0 \in \{0,1\}^{2n}$
  Construct $S_{0,\Delta}$ from S_0 by flipping the bit on position r
  Run KSA$^{-1}(S_{0,\Delta})$ routine for $2n$ clocks and produce state $S' = K' \| V'$
  if V' is `valid` **then**
    Set $s \leftarrow 1$ and $IV' \leftarrow \texttt{Extract}_{IV}(V')$
    return (K, IV) and (K', IV')
  end
end

We further present an algorithm that checks which keystream positions produced by the states S and S_Δ are identical (introduced in Algorithm 10). Note that if we run Algorithm 11 we obtain less positions than claimed in Theorems 2.1 to 2.3, as shown in Appendix F. This is due to the fact that Algorithm 11 is prone to producing internal collisions and, thus, eliminate certain positions that are identical in both keystreams. Note that Theorem 2.4 is a refined version of Remarks 2.3 to 2.5 in the sense that it represents an automatic tool for finding identical keystream positions.

Modified Pseudorandom Keystream Generation Algorithm (PRGA′). To obtain our modified PRGA we replace + (XOR) and · (AND) operations in the original PRGA with | (OR) operations.

Theorem 2.4. Let r be a position of Grain's internal state, q_1 the number of desired identical positions in the keystream and q_2 the maximum number of search trials. Then, Algorithm 11 finds at most q_1 identical positions in a maximum of q_2 trials.

Proof. We note that in Algorithm 11 the bit b_r on position r is set. If b_r is taken into consideration while computing the output bit of PRGA then the output of PRGA$'$ is also set due to the replacement of the original operations ($+$ and $\cdot$) with $|$ operations. The same argument is valid if a bit of Grain's internal state is influenced by b_r.

The above statements remain true for each internal state bit that becomes set during the execution of Algorithm 11. □

Algorithm 11. Search for identical keystream position in Grain.

Input: Integers $r \in \{0, 2n\}$ and $q_1, q_2 > 0$
Output: Keystream positions φ

```
Set s ← 0 and φ ← ∅
Let S ∈ {0,1}^{2n} be the zero state (0, ..., 0)
Construct S_Δ from S by flipping the bit on position r
while |φ| ⩽ q_1 and s < q_2 do
    Set b ← PRGA'(S_Δ) and update state S_Δ with the current state
    if b = 0 then
        Update φ ← φ ∪ {s}
    end
    Set s ← s + 1
end
return φ
```

2.2.3 Proposed Ideas

2.2.3.1 Compact Padding

Attacks that exploit the periodic padding used in Grain-128 where first presented in [64, 162] and further improved in [38]. We generalize and simplify these attacks below.

Setup. Let $Y_1 = [y_0, \ldots, y_{d_1-1}]$, where $|Y_1| = d_1$, let $Y_2 = [y_{d_1+\alpha}, \ldots, y_{n-1}]$, where $|Y_2| = d_2$ and let $IV = Y_1 \| Y_2$. We define

$$\texttt{Load}_{IV}(IV, P) = Y_1 \| P \| Y_2.$$

Let $S = [s_0, \ldots, s_{n-1}]$ be a state of the LFSR, then we define

$$\texttt{Extract}_{IV}(S) = s_0 \| \ldots \| s_{d_1-1} \| \ldots \| s_{d_1+\alpha} \| \ldots \| s_{n-1}.$$

Padding. Let $\alpha = \lambda\omega$ and $|P_0| = \ldots = |P_{\omega-1}| = \lambda$, then we define $P = P_0 \| \ldots \| P_{\omega-1}$. We say that P is a *periodic padding of order* λ if λ is the smallest integer such that $P_0 = \ldots = P_{\omega-1}$.

Periodic padding of order α is further referred to as *aperiodic padding*.

Theorem 2.5. Let P be a periodic padding of order λ and let $i = 1, 2$ denote an index. For each (set of) condition(s) presented in Column 2 of Table 2.15 there exists an attack whose corresponding success probability is presented in Column 3 of Table 2.15.

	Conditions	Success Probability
1.	$d_1 \geqslant \lambda$ or $d_2 \geqslant \lambda$	$1/2^{\lambda}$
2.	$d_1 \geqslant \lambda$ and $d_2 \geqslant \lambda$	$1/2^{\lambda-1}$
3.	$d_i < \lambda$	$1/2^{2\lambda-d_i}$

TABLE 2.15: Attack parameters for Theorem 2.5.

Proof.

1. The proof follows directly from Algorithms 13 and 15. Given the assumptions in Section 2.2.1, the probability that the first λ keystream bits are zero is $1/2^{\lambda}$.

2. The proof is a direct consequence of Item 1.

3. The proof is straightforward in the light of Algorithms 16 and 17. Given the assumptions in Section 2.2.1, the probability that $V_1' = P_0$ is $1/2^{\lambda-d_1}$ and the probability that $V_2' = P_{\omega-1}$ is $1/2^{\lambda-d_2}$. Also, the probability that the first λ keystream bits are zero is $1/2^{\lambda}$. Since the two events are independent, we obtain the desired success probability.

Algorithm 12. $\texttt{Pair}_1(\sigma, S)$.

Input: Number of clocks σ and a state S.
Output: A Key-IV pair (K', IV') or $\perp$

Run $\text{KSA}^{-1}(S)$ routine for σ clocks and produce state $S' = (K' \| V_1' \| P \| P_{\omega-1} \| V_2')$, where $|V_1'| = d_1$ and $|V_2'| = d_2 - \lambda$
Set $IV' \leftarrow V_1' \| P_{\omega-1} \| V_2'$
if (K', IV') produces all zero keystream bits in the first λ PRGA rounds **then**
  return (K', IV')
end
return $\perp$

□

Algorithm 13. Constructing Key-IV pairs that generate λ bit shifted keystream.

Output: Key-IV pairs (K', IV') and (K, IV)

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^n, V_1 \in_R \{0,1\}^{d_1-\lambda}$ and $V_2 \in_R \{0,1\}^{d_2}$
  Set $IV \leftarrow V_1 \| P_0 \| V_2$, $S \leftarrow K \| V_1 \| P_0 \| P \| V_2$ and $output \leftarrow \texttt{Pair}_1(\lambda, S)$
  if $output \neq \perp$ **then**
    Set $s \leftarrow 1$
    return (K, IV) and $output$
  end
end

Algorithm 14. $\texttt{Pair}_2(\sigma, S)$.

Input: Number of clocks σ and a state S.

Output: A Key-IV pair (K', IV').

Run KSA(S) routine for σ clocks and produce state $S' = (K' \| V_1' \| P_0 \| P \| V_2')$, where $|V_1'| = d_1 - \lambda$ and $|V_2'| = d_2$
Set $IV' \leftarrow V_1' \| P_0 \| V_2'$
return (K', IV')

Algorithm 15. Constructing Key-IV pairs that generate λ bit shifted keystream.

Output: Key-IV pairs (K', IV') and (K, IV)

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^n, V_1 \in_R \{0,1\}^{d_1}$ and $V_2 \in_R \{0,1\}^{d_2-\lambda}$
  Set $IV \leftarrow V_1 \| P_{\omega-1} \| V_2$
  if (K, IV) produces all zero keystream bits in the first λ PRGA rounds **then**
    Set $s \leftarrow 1$ and $S \leftarrow (K \| V_1 \| P \| P_{\omega-1} \| V_2)$
    return (K, IV) and $\texttt{Pair}_2(\lambda, S)$
  end
end

Remark 2.6. Let $d_2 = 0, \lambda = 1, P_0 = 1$. If $\alpha = 16$, then the attack described in [162] is the same as the attack we detail in Algorithm 17. The same is true for [64] if $\alpha = 32$. Also, if $\alpha = 32$ then Algorithm 13 is a simplified version of the attack presented in [38].

Remark 2.7. To minimize the impact of Theorem 2.5, one must choose a padding value such that $\lambda = \alpha$ and either $d_1 < \alpha$ or $d_2 < \alpha$. In this case, because of the generic attacks described in Section 2.2.2, the success probability can not drop below $1/2^{\alpha}$. The designers of Grain-128a have chosen $d_2 = 0$ and $P = \texttt{0xfffffffe}$. In [39], the authors introduce an attack for Grain-128a, which is a special case of the attack we detail in Algorithm 13.

Algorithm 16. Constructing Key-IV pairs that generate λ bit shifted keystream.

Output: Key-IV pairs (K'', IV'') and (K, IV)

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^n$ and $V_2 \in_R \{0,1\}^{d_2}$
  Set $IV \leftarrow LSB_{d_1}(P_0) \| V_2$
  Run $\text{KSA}^{-1}(K \| LSB_{d_1}(P_0) \| P \| V_2)$ routine for $\lambda - d_1$ clocks and produce state $S' = (K' \| V_1' \| P \| V_2')$, where $|V_1'| = \lambda$ and $|V_2'| = d_2 - \lambda + d_1$
  if $V_1' = p_0$ **then**
    Set $S \leftarrow K' \| P_0 \| P \| V_2'$ and *output* $\leftarrow \texttt{Pair}_1(d_1, S)$
    if *output* $\neq \perp$ **then**
      Set $s \leftarrow 1$
      return (K, IV) and *output*
    end
  end
end

Algorithm 17. Constructing Key-IV pairs that generate λ bit shifted keystream.

Output: Key-IV pairs (K'', IV'') and (K, IV)

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^n$ and $V_1 \in_R \{0,1\}^{d_1}$
  Set $IV \leftarrow V_1 \| MSB_{d_2}(P_{\omega-1})$
  if K, IV produces all zero keystream bits in the first λ PRGA rounds **then**
    Run $\text{KSA}(K \| V_1 \| P \| MSB_{d_2}(P_{\omega-1}))$ routine for $\lambda - d_2$ clocks and produce state $S' = (K' \| V_1' \| P \| V_2')$, where $|V_1'| = d_1 - \lambda + d_2$ and $|V_2'| = \lambda$
    if $V_2' = P_{\omega-1}$ **then**
      Set $s \leftarrow 1$ and $S \leftarrow (K' \| V_1' \| P \| P_{\omega-1})$
      return (K, IV) and $\texttt{Pair}_2(d_2, S)$
    end
  end
end

Theorem 2.6. Let P be an aperiodic padding, $1 \leqslant \gamma < \alpha/2$ and $d_2 < \alpha$. Also, let $i = 1, 2$ denote an index. If $LSB_\gamma(P) = MSB_\gamma(P)$, then for each condition presented in Column 2 of Table 2.16 there exists an attack whose corresponding success probability is presented in Column 3 of Table 2.16.

	Condition	Success Probability
1.	$d_i \geqslant \alpha - \gamma$	$1/2^{\alpha-\gamma}$
2.	$d_i < \alpha - \gamma$	$1/2^{2\alpha-2\gamma-d_i}$

TABLE 2.16: Attack parameters for Theorem 2.6.

Proof. 1. The first part of proof follows from Algorithm 13 with the following changes:

(a) λ is replaced by $\alpha - \gamma$;

(b) P_0 is replaced by $MSB_{\alpha-\gamma}(P)$;

(c) $P_{\omega-1}$ is replaced by $LSB_{\alpha-\gamma}(P)$.

Therefore, the probability that the first $\alpha - \gamma$ keystream bits are zero is $1/2^{\alpha-\gamma}$. Similarly, the second part follows from Algorithm 15.

2. To prove the first part, we use the above changes on Algorithm 16, except that instead of replacing $P_{\omega-1}$ we replace $LSB_{d_1}(P_0)$ with $MID_{[\gamma+d_1-1,\gamma]}(P)$. Thus, we obtain the probability $1/2^{\alpha-\gamma}$. Similarly, for the second part we use Algorithm 17.

□

Remark 2.8. To prevent the attacks presented in the proof of Theorem 2.6, the padding must be chosen such that $MSB_\gamma(P) \neq LSB_\gamma(P)$, $\forall\ 1 \leqslant \gamma < \alpha/2$. Grain 128a uses such a padding $P =$ `0xfffffffe`. Another example was suggested in [64] to counter their proposed attacks: $P =$ `0x00000001`.

Constraints. Taking into account all the previous remarks, we may conclude that *good*[23] compact padding schemes are aperiodic and, in particular, satisfy $MSB_\gamma(P) \neq LSB_\gamma(P)$, $\forall\ 1 \leqslant \gamma < \alpha/2$. Also, another constraint is the position of the padding, *i.e.* $d_1 < \alpha$ or $d_2 < \alpha$ must be satisfied.

Remark 2.9. In the compact padding case, the number of padding schemes that verify the security restrictions represent 26% of the total 2^α. The previous percentage and the values we mention below were determined experimentally.

For $\alpha = 16$ and $0 \leqslant d_1, d_2 < 16$ we obtain $17622 \simeq 2^{14}$ compact padding schemes resistant to previous attacks. Thus, the complexity of a brute-force attack increases with 2^{19}.

For $\alpha = 32$ and $0 \leqslant d_1, d_2 < 32$ we obtain $1150153322 \simeq 2^{30}$ compact padding schemes resistant to previous attacks. Thus, the complexity of a brute-force attack increases with 2^{36}.

2.2.3.2 Fragmented Padding

Setup. Let $\alpha = c{\cdot}\beta$, where $c > 1$. Also, let $IV = B_0\|B_1\|\ldots\|B_c$ and $P = P_0\|P_1\|\ldots\|P_{c-1}$, where $|B_0| = d_1$, $|P_0| = \ldots = |P_{c-1}| = |B_1| = \ldots = |B_{c-1}| = \beta$ and $|B_c| = d_2$. In this

[23] resistant to the aforementioned attacks

case, we define

$$\texttt{Load}_{IV}(IV, P) = B_0 \| P_0 \| B_1 \| P_1 \| \ldots \| B_{c-1} \| P_{c-1} \| B_c.$$

Let $S = S_0 \| \ldots \| S_{2c}$ be a state of the LFSR, such that $|S_0| = d_1$, $|S_1| = \ldots = |S_{2c-1}| = \beta$ and $|S_{2c}| = d_2$. Then we define

$$\texttt{Extract}_{IV}(S) = S_0 \| S_2 \| \ldots \| S_{2c}.$$

Theorem 2.7. Let $i = 1, 2$ denote an index. In the previously mentioned setting, for each (set of) condition(s) presented in Column 2 of Table 2.17 there exists an attack whose corresponding success probability is presented in Column 3 of Table 2.17.

	Conditions	Success Probability
1.	$d_1 \geqslant \beta$ or $d_2 \geqslant \beta$	$1/2^{\beta}$
2.	$d_1 \geqslant \beta$ and $d_2 \geqslant \beta$	$1/2^{\beta-1}$
3.	$d_i < \beta$	$1/2^{2\beta-d_i}$

TABLE 2.17: Attack parameters for Theorem 2.7.

Proof. 1. We only prove the case $i = 1$ as the case $i = 2$ is similar in the light of Algorithm 15. The proof follows directly from Algorithm 20. Given the assumptions in Section 2.2.1, the probability that the first β keystream bits are zero is $1/2^{\beta}$.

Algorithm 18. $\texttt{Update}_1()$.

Output: Variable *value*

```
Set value ← P_0
for i = 1 to c − 1 do
    Update value ← value‖P_i‖P_i
end
return value
```

2. The proof is a direct consequence of Item 1.

3. Again, we only prove the case $i = 1$. The proof is straightforward in the light of Algorithm 21. Given the assumptions in Section 2.2.1, the probability that $V_1' = P_0$ is $1/2^{\beta-d_1}$. Also, the probability that the first β keystream bits are zero is $1/2^{\beta}$. Since the two events are independent, we obtain the desired success probability.

□

Algorithm 19. $\texttt{Pair}_3(\sigma, S)$.

Input: Number of clocks σ and a state S.
Output: A Key-IV pair (K', IV') or $\perp$

Run $\text{KSA}^{-1}(S)$ routine for σ clocks and produce state $S' = (K'\|V_1'\|value\|V_2')$, where $|V_1'| = d_1$ and $|V_2'| = d_2 - \beta$
Set $IV' \leftarrow V_1'\|P\|V_2'$
if (K', IV') produces all zero keystream bits in the first β PRGA rounds **then**
  return (K', IV')
end
return $\perp$

Algorithm 20. Constructing Key-IV pairs that generate β bit shifted keystream.

Output: Key-IV pairs (K', IV') and (K, IV)

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^n, V_1 \in_R \{0,1\}^{d_1-\beta}$ and $V_2 \in_R \{0,1\}^{d_2}$
  Set $value \leftarrow P_0\|\texttt{Update}_1()$, $IV \leftarrow V_1\|P\|V_2$, $S \leftarrow K\|V_1\|value\|V_2$ and $output \leftarrow \texttt{Pair}_3(\beta, S)$
  if $output \neq \perp$ **then**
    Set $s \leftarrow 1$
    return (K, IV) and $output$
  end
end

Algorithm 21. Constructing Key-IV pairs that generate β bit shifted keystream.

Output: Key-IV pairs (K', IV') and (K, IV)

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^n$ and $V_2 \in_R \{0,1\}^{d_2}$
  Set $value \leftarrow \texttt{Update}_1()$ and $IV \leftarrow LSB_{\alpha-\beta+d_1}(P)\|V_2$
  Run $\text{KSA}^{-1}(K\|LSB_{d_1}(P_0)\|value\|V_2)$ routine for $\beta - d_1$ clocks and produce state $S' = (K'\|V_1'\|value\|V_2')$, where $|V_1'| = \beta$ and $|V_2'| = d_2 - \beta + d_1$
  if $V_1' = P_0$ **then**
    Set $S \leftarrow K'\|P_0\|value\|V_2'$ and $output \leftarrow \texttt{Pair}_3(d_1, S)$
    if $output \neq \perp$ **then**
      Set $s \leftarrow 1$
      return (K, IV) and $output$
    end
  end
end

Remark 2.10. Let $\delta < \beta$ and $\beta > 1$. To prevent the attacks presented in Theorem 2.7, we have to slightly modify the structure of the IV. We need at least one block $|B_i| = \delta$, where $1 \leqslant i \leqslant c-1$. We further consider that $|B_i| = \delta$, $\forall\ 1 \leqslant i \leqslant c-1$.

Theorem 2.8. Let $|B_i| = \delta$, $\forall\ 1 \leqslant i \leqslant c-1$. Also, let $1 \leqslant \gamma \leqslant \beta$, $1 \leqslant t \leqslant c$ and

$0 \leqslant j \leqslant t-1$. If $LSB_\gamma(P_{c-1-j}) = MSB_\gamma(P_{t-1-j})\ \forall j$ then for each (set of) condition(s) presented in Column 2 of Table 2.18 there exists an attack whose corresponding success probability is presented in Column 3 of Table 2.18.

	Conditions	Success Probability
1.	$d_1 \geqslant \beta - \gamma + (\beta+\delta)(c-t),\ \delta \geqslant \beta - \gamma$	$1/2^{\beta-\gamma+(\beta+\delta)(c-t)}$
2.	$d_1 \geqslant \beta - \gamma + (\beta+\delta)(c-t),\ \delta < \beta - \gamma,$ $MSB_{\beta-\gamma-\delta}(P_{c-1-j}) = LSB_{\beta-\gamma-\delta}(P_{t-2-j})\ \forall j$	$1/2^{\beta-\gamma+(\beta+\delta)(c-t)}$
3.	$d_1 < \beta - \gamma + (\beta+\delta)(c-t),\ \delta \geqslant \beta - \gamma$	$1/2^{2\beta-2\gamma+2(\beta+\delta)(c-t)-d_1}$
4.	$d_1 < \beta - \gamma + (\beta+\delta)(c-t),\ \delta < \beta - \gamma,$ $MSB_{\beta-\gamma-\delta}(P_{c-1-j}) = LSB_{\beta-\gamma-\delta}(P_{t-2-j})\ \forall j$	$1/2^{2\beta-2\gamma+2(\beta+\delta)(c-t)-d_1}$

TABLE 2.18: Attack parameters for Theorem 2.8.

Proof. 1. The proof follows directly from Algorithm 24. Given the assumptions in Section 2.2.1, the probability that the first $\beta - \gamma + (\beta+\delta)(c-t)$ keystream bits are zero is $1/2^{\beta-\gamma+(\beta+\delta)(c-t)}$.

Algorithm 22. $\texttt{Update}_2(start, stop)$.

Input: Indexes *start* and *stop*
Output: Variable *value*

```
Set value ← NULL
for i = start to stop do
    Choose C_i ∈_R {0,1}^δ
    Update value ← value‖C_i‖P_i
end
return value
```

Algorithm 23. $\texttt{Update}_3(value_1, value_2)$.

Input: Variables $value_1$ and $value_2$
Output: Variable *value*

```
for i = t to c − 1 do
    Choose B_i ∈_R {0,1}^δ
    Update value_1 ← value_1‖B_i‖P_i and value_2 ← value_2‖B_i
end
Set value ← value_1‖value_2
return value
```

The proofs for the remaining cases presented in Table 2.18 follow directly from previous results. Thus, we omit them. □

Algorithm 24. Constructing Key-IV pairs that generate $\beta - \gamma + (\beta + \delta)(c - t)$ bit shifted keystream.

Output: Key-IV pairs (K', IV') and (K, IV)

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^n, V_1 \in_R \{0,1\}^{d_1 - \beta + \gamma - (\beta+\delta)(c-t)}$ and $V_2 \in_R \{0,1\}^{d_2}$
  Set $value_1 \leftarrow P_0 \| \texttt{Update}_2(0, c-t-2) \| C_{c-t-1} \| MSB_{\beta-\gamma}(P_{c-t})$ and $value_2 \leftarrow value_1$
  Update $value_1 \leftarrow value_1 \| P_0$
  for $i = 1$ to $t - 1$ **do**
    Choose $B_i \in_R \{0,1\}^{\delta - \beta + \gamma}$
    Update $value_1 \leftarrow value_1 \| B_i \| MSB_{\beta-\gamma}(P_{c-t+i}) \| P_i$ and $value_2 \leftarrow value_2 \| B_i \| MSB_{\beta-\gamma}(P_{c-t+i})$
  end
  Set $value_1 \| value_2 \leftarrow \texttt{Update}_3(value_1, value_2)$ and $IV \leftarrow V_1 \| value_2 \| V_2$
  Run $\text{KSA}^{-1}(K \| V_1 \| value_1 \| V_2)$ routine for $\beta - \gamma + (\beta + \delta)(c - t)$ clocks and produce state $S' = (K' \| V_1' \| value_1 \| V_2')$, where $|V_1'| = d_1$ and $|V_2'| = d_2 - \beta + \gamma - (\beta + \delta)(c - t)$
  Set $IV' \leftarrow V_1' \| value_1 \| V_2'$
  if (K', IV') produces all zero keystream bits in the first $\beta - \gamma + (\beta + \delta)(\text{c} - \text{t})$ PRGA rounds **then**
    Set $s \leftarrow 1$
    return (K, IV) and (K', IV')
  end
end

Theorem 2.9. Let $|B_i| = \delta$, $\forall\ 1 \leqslant i \leqslant c - 1$. Also, let $1 \leqslant \gamma \leqslant \beta$, $1 \leqslant t \leqslant c$ and $0 \leqslant j \leqslant t - 2$. If $\delta \geqslant \beta - \gamma$ then for each (set of) condition(s) presented in Column 2 of Table 2.19 there exists an attack whose corresponding success probability is presented in Column 3 of Table 2.19.

	Conditions	Success Probability
1.	$d_1 \geqslant \delta - \beta + \gamma + \beta(c - t + 1) + \delta(c - t)$, $MSB_\gamma(P_{c-1-j}) = LSB_\gamma(P_{t-2-j}) \forall j$	$1/2^{\delta - \beta + \gamma + \beta(c-t+1) + \delta(c-t)}$
2.	$d_1 < \delta - \beta + \gamma + \beta(c - t + 1) + \delta(c - t)$, $MSB_\gamma(P_{c-1-j}) = LSB_\gamma(P_{t-2-j}) \forall j$	$1/2^{2\delta - 2\beta + 2\gamma + 2\beta(c-t+1) + 2\delta(c-t) - d_1}$

TABLE 2.19: Attack parameters for Theorem 2.9.

Proof. 1. The proof follows directly from Algorithm 25. Given the assumptions in Section 2.2.1, the probability that the first $\delta - \beta + \gamma + \beta(c - t + 1) + \delta(c - t)$ keystream bits are zero is $1/2^{\delta - \beta + \gamma + \beta(c-t+1) + \delta(c-t)}$.

2. The proof is similar to the proof of Theorem 2.7, Item 3.

□

Algorithm 25. Constructing Key-IV pairs that generate $\delta-\beta+\gamma+\beta(c-t+1)+\delta(c-t)$ bit shifted keystream.

Output: Key-IV pairs (K', IV') and (K, IV)

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^n, V_1 \in_R \{0,1\}^{d_1-\delta+\beta-\gamma-\beta(c-t+1)-\delta(c-t)}, V_2 \in_R \{0,1\}^{d_2}$ and $C_{c-t+1} \in_R \{0,1\}^{\delta-\beta+\gamma}$
  Set $value_1 \leftarrow P_0 \| \texttt{Update}_2(1, c-t) \| C_{c-t+1}$ and $value_2 \leftarrow value_1$
  Update $value_1 \leftarrow value_1 \| P_0$
  for $i = 1$ to $t-1$ **do**
    Choose $B_i \in_R \{0,1\}^{\delta-\beta+\gamma}$
    Update $value_1 = value_1 \| LSB_{\beta-\gamma}(P_{c-t+i}) \| B_i \| P_i$ and $value_2 = value_2 \| LSB_{\beta-\gamma}(P_{c-t+i}) \| B_i$
  end
  Set $value_1 \| value_2 \leftarrow \texttt{Update}_3(value_1, value_2)$ and $IV \leftarrow V_1 \| value_2 \| V_2$
  Run $\text{KSA}^{-1}(K \| V_1 \| value_1 \| V_2)$ routine for $\delta - \beta + \gamma + \beta(c-t+1) + \delta(c-t)$ clocks and produce state $S' = (K' \| V_1' \| value_1 \| V_2')$, where $|V_1'| = d_1$ and $|V_2'| = d_2 - \delta + \beta - \gamma - \beta(c-t+1) - \delta(c-t)$
  Set $IV' \leftarrow V_1' \| value_1 \| V_2'$
  if (K', IV') produces all zero keystream bits in the first $\delta - \beta + \gamma + \beta(\text{c} - \text{t} + 1) + \delta(\text{c} - \text{t})$ PRGA rounds **then**
    Set $s \leftarrow 1$
    return (K, IV) and (K', IV')
  end
end

Remark 2.11. Taking into account the generic attacks described in Section 2.2.2, any probability bigger than $1/2^\alpha$ is superfluous. As an example, when $\alpha = 32$ we obtain a good padding scheme for the following parameters $d_2 = 0, \beta = 16, \delta = 14, P_0 = \texttt{0x8000}, P_1 = \texttt{0x7fff}$.

Remark 2.12. Let $c = 2, \delta \leqslant \beta - 2, \gamma < \beta$ and $P_0 \neq P_1$. The best success probability of a slide attack when the following conditions are met:

$$\begin{aligned} \gamma > 1: \quad & LSB_\gamma(P_1) \neq MSB_\gamma(P_0) \\ & LSB_\gamma(P_0) \neq MSB_\gamma(P_1), \\ \gamma > 0: \quad & LSB_\gamma(P_1) \neq MSB_\gamma(P_1) \\ & LSB_\gamma(P_0) \neq MSB_\gamma(P_0), \end{aligned}$$

is $1/2^{\alpha-1+\delta} \geqslant 1/2^\alpha$. The number of padding schemes that verify the security restrictions represent 2% of the total 2^α. The previous percentage and the values we mention below were determined experimentally.

For $\alpha = 16, \beta = 8, 1 \leqslant \delta \leqslant 6, \gamma < 8$ and $d_1 = d_2 = 0$ we obtain $1840 \simeq 2^{10}$ fragmented padding schemes resistant to previous attacks. Thus, the complexity of a brute-force attack increases with 2^{14}.

For $\alpha = 32, \beta = 16, 1 \leqslant \delta \leqslant 14, \gamma < 16$ and $d_1 = d_2 = 0$ we obtain $117113488 \simeq 2^{23}$ fragmented padding schemes resistant to previous attacks. Thus, the complexity of a brute-force attack increases with 2^{28}.

2.2.4 Future Work

A closely related study which naturally arises is analyzing the security of breaking the padding into aperiodic blocks. Another idea would be to study how the proposed padding techniques interfere with the security of the authentication feature of Grain-128a. A question that arises is if the occurrence of slide pairs may somehow be converted into a distinguishing or key recovery attack. Another interesting point would be to investigate what would happen to the security of the Grain family with respect to differential, linear or cube attacks in the various padding scenarios we outlined. One more future work idea could be to analyze various methods of preventing the all zero state of Grain's LFSR.

2.3 Quasigroup Substitution Permutation Networks

In its most basic form, differential cryptanalysis [55] predicts how certain changes in the plaintext propagate through a cipher. When considering an ideally randomizing cipher, the probability of predicting these changes is $1/2^n$, where n is the number of input bits. Thus, in the ideal case, it is infeasible for an attacker to use these predictions when n is, for example, 128. Unfortunately, designers use theoretical estimates based on certain assumptions that do not always hold in practice. Hence, differential cryptanalysis is often the most effective tool against symmetric key cryptographic algorithms [188].

Quasigroups are group-like structures that, unlike groups, are not required to be associative and to possess an identity element. The usage of quasigroups as building blocks for cryptographic primitives is not very common. Regardless of that, various such cryptosystems can be found in the literature [164, 117, 116, 35, 90, 160].

In this paper we introduce a straightforward generalization of substitution-permutation networks (SPN) and study its security. By replacing the group operation $\star$ between keys and (intermediary) plaintexts with a quasigroup operation $\otimes$ we aimed at extending the usage of quasigroups. Unfortunately, by means of differential cryptanalysis we prove that

in the case of quasigroups isotopic with a group[24] the problem of breaking an SPN using $\otimes$ reduces to breaking an SPN using $\star$ and a substitution box (s-box) different from the initial one. Thus, if we initialize the SPN with a random secret s-box, replacing $\star$ with $\otimes$ brings no extra security[25]. In the case of static s-boxes, changing $\star$ with $\otimes$ might even affect the SPN's security.

Although the design presented in this paper is not a successful one, we think that its usefulness is twofold. ① Most scientific reports and papers published appear as sanitized accounts[26] and this gives people a distorted view of scientific research [179, 146, 235, 255]. This leads to a view that implies that failure, serendipity and unexpected results are not a normal part of science [146, 220]. Hence, this report provides students with an indication of the real processes of experimentation. ② Negative results and false directions are rarely reported [146, 248] and, thus, people are bound to repeat the same mistakes. By presenting our results, we hope to provide an opportunity for others to learn where this path leads. Hence, preventing them to make the same mistakes[27].

2.3.1 Preliminaries

2.3.1.1 Quasigroups

In this section we introduce a few basic notions about quasigroups. We base our exposition on [230].

Definition 2.1. A quasigroup $(\mathbb{G}, \otimes)$ is a set $\mathbb{G}$ equipped with a binary operation $\otimes : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}$, in which specification of any two of the values x, y, z in the equation $x \otimes y = z$ determines the third uniquely.

Definition 2.2. For a quasigroup $(\mathbb{G}, \otimes)$ we define the left division $x \backslash z = y$ as the unique solution y to $x \otimes y = z$. Similarly, we define the right division $z / y = x$ as the unique solution x to $x \otimes y = z$.

Lemma 2.1. The following identities hold

$$y \backslash (y \otimes x) = x, \qquad (x \otimes y) / y = x,$$
$$y \otimes (y \backslash x) = x, \qquad (x / y) \otimes y = x.$$

[24] Note that this is the most popular method for generating quasigroups.
[25] *i.e.* we simply obtain another instantiation of the SPN
[26] Authors present their results as if they achieved them in a straightforward manner and not through a messy process.
[27] In [236], the author advises people to write down their mistakes so that they avoid making them again in the future.

Definition 2.3. Let $(\mathbb{G}, \otimes)$, $(\mathbb{H}, \star)$ be two quasigroups. An ordered triple of bijections π, ρ, ω of a set $\mathbb{G}$ onto the set $\mathbb{H}$ is called an isotopy of $(\mathbb{G}, \otimes)$ to $(\mathbb{H}, \star)$ if for any $x, y \in \mathbb{G}$ $\pi(x) \star \rho(y) = \omega(x \otimes y)$. If such an isotopism exists, then $(\mathbb{G}, \otimes)$, $(\mathbb{H}, \star)$ are called isotopic.

A popular method for constructing quasigroups [116, 117, 160, 251] is the following. Choose a group $(\mathbb{G}, \star)$ (*e.g.* $(\mathbb{Z}_{2^n}, \oplus)$ or $(\mathbb{Z}_{2^n}, +)$) and three random permutations $\pi, \rho, \omega : \mathbb{G} \to \mathbb{G}$. Then, define the quasigroup operation as $x \otimes y = \omega^{-1}(\pi(x) \star \rho(y))$. To see why this leads to a quasigroup, we note that x, y and z are mapped uniquely to $\pi(x)$, $\rho(y)$ and $\omega(z)$ and, thus, any equation of the form $\pi(x) \star \rho(y) = \omega(z)$ is in fact uniquely resolved in the base group $\mathbb{G}$ given any of $\pi(x)$, $\rho(y)$ and $\omega(z)$.

Example 2.3. Let $(\mathbb{G}, \star) = (\mathbb{Z}_4, \oplus), \omega^{-1} = \{2, 1, 0, 3\}$, $\pi = \{2, 1, 3, 0\}$ and $\rho = \{2, 0, 3, 1\}$. The corresponding quasigroup operations for $(\mathbb{Z}_4, \otimes)$ can be found in Table 2.20.

$\otimes$	0	1	2	3
0	2	0	1	3
1	3	1	0	2
2	1	3	2	0
3	0	2	3	1

$\oslash$	0	1	2	3
0	1	2	0	3
1	2	1	3	0
2	3	0	2	1
3	0	3	1	2

$\oslash$	0	1	2	3
0	3	0	1	2
1	2	1	0	3
2	0	3	2	1
3	1	2	3	0

TABLE 2.20: Quasigroup operations.

Example 2.4. Let $(\mathbb{G}, \star) = (\mathbb{Z}_n, -)$. Then $\mathbb{G}$ is isotopic with $(\mathbb{Z}_n, +)$, where $\omega, \pi = Id$ and $\rho(i) = n - i \bmod n$ [251].

2.3.1.2 Group Differential Cryptanalysis

Differential cryptanalysis was initially introduced in [55] for $(\mathbb{Z}_{2^n}, \oplus)$ and was extended to abelian groups in [165]. We further extend the notion to non-commutative groups.

Definition 2.4. Let $\mathbb{G}$ be a set equipped with a binary operation $\bullet : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$. The difference between two elements $X, X' \in (\mathbb{G}, \bullet)$ is defined as $\Delta_\bullet(X, X') = X \bullet X'$.

Definition 2.5. Let $(\mathbb{G}, \star)$ be a group. We define the group differential probabilities

$$LDP_\star(\sigma, \alpha, \beta) = \frac{1}{|\mathbb{G}|} \sum_{\substack{X, X' \in \mathbb{G} \\ \Delta_\star(X^{-1}, X') = \alpha}} [\Delta_\star(\sigma(X)^{-1}, \sigma(X')) = \beta]$$

$$RDP_\star(\sigma, \alpha, \beta) = \frac{1}{|\mathbb{G}|} \sum_{\substack{X, X' \in \mathbb{G} \\ \Delta_\star(X, X'^{-1}) = \alpha}} [\Delta_\star(\sigma(X), \sigma(X')^{-1}) = \beta].$$

where $\sigma : \mathbb{G} \to \mathbb{G}$ is a permutation and $\alpha, \beta \in \mathbb{G}$.

Differential cryptanalysis exploits the high probability of certain occurrences of plaintext differences and differences into the last round of the cipher [143]. Thus, an attacker first computes the values of a round's LDPs (RDPs). Note that in the case of groups LDPs are dependent only on the round's non-linear layer. Hence, in the case of SPNs only the s-box's LDP values are needed. Once the LDPs are computed, the attacker examines likely differential characteristics. By a differential characteristic χ we understand a sequence of input and output differences such that the output difference of a round is the input difference of the next round. Using the most likely differential characteristic[28] an attacker exploits information coming into the last round of the cipher to derive parts of the last layer's subkey. More precisely, he partially decrypts the last round for each pair of ciphertexts[29] for all possible partial subkeys. When the difference for the input to the last round corresponds to the value expected from χ a counter incremented. The partial subkey value with the highest counter is assumed to be the correct partial subkey. For a concrete example of the whole process, we refer the reader to [143].

Example 2.5. Let $(\mathbb{G}, \star) = (\mathbb{Z}_8, \oplus)$ and $\sigma = \{5, 1, 0, 3, 4, 2, 6, 7\}$. The difference distribution table for the $\oplus$ operation and the σ s-box can be found in Table 2.21. For simplicity, we multiplied all the $LDP_{\oplus}(\sigma, \alpha, \beta)$ values by $|\mathbb{G}|$. Note that in this case $LDP_{\oplus} = RDP_{\oplus}$.

α \ β	0	1	2	3	4	5	6	7
0	8	0	0	0	0	0	0	0
1	0	2	0	2	2	0	2	0
2	0	0	4	0	0	4	0	0
3	0	2	0	2	2	0	2	0
4	0	2	0	2	2	0	2	0
5	0	0	0	0	0	4	0	4
6	0	2	0	2	2	0	2	0
7	0	0	4	0	0	0	0	4

TABLE 2.21: Difference distribution table for $\oplus$ and σ.

2.3.2 Quasigroup Substitution Permutation Network

Let n be a positive integer and $(\mathbb{G}, \otimes)$ a quasigroup. An SPN (see Figure 2.7) is an iterated structure that processes a plaintext for r rounds. Each round consist of a substitution layer $(S_1, \ldots, S_n)$, a permutation layer (P_i) and a key mixing operation. Also, the SPN has an initial round that consists only of a key mixing operation. Note

[28] When constructing differential trails we ignore the case $\alpha, \beta = e$, where e is the identity element of group $\mathbb{G}$.

[29] corresponding to the pairs of plaintexts used to generate χ

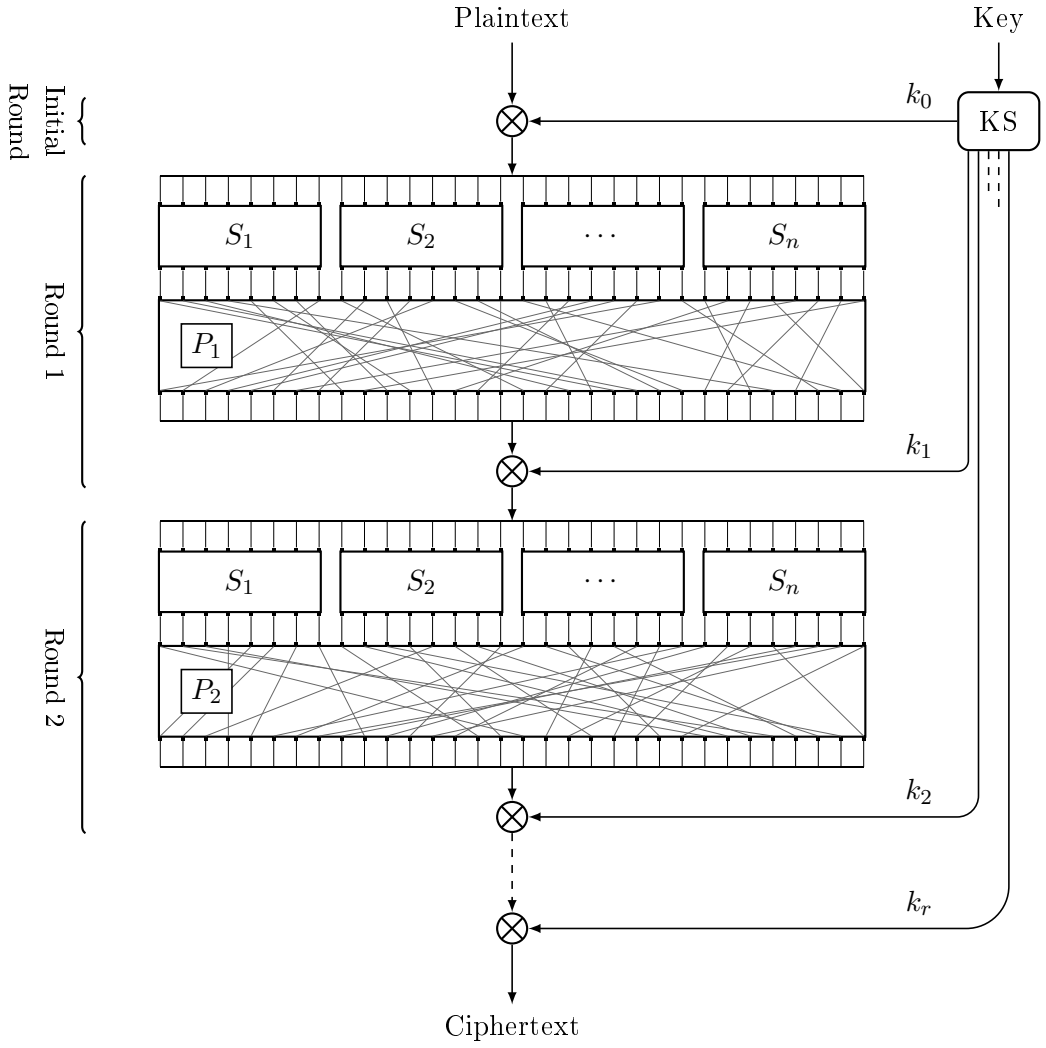


FIGURE 2.7: Quasigroup substitution permutation network.

that for each round i the key schedule algorithm (KS) derives the subkey k_i from the initial key.

Let $p_i = p_i^1 \| \dots \| p_i^n$ and $k_i = k_i^1 \| \dots \| k_i^n$ be the intermediary plaintext and, respectively, the subkey for round i[30]. Then, a left quasigroup SPN has as a key mixing operation $k_i \otimes p_i = k_i^1 \otimes p_i^1 \| \dots \| k_i^n \otimes p_i^n$, while a right quasigroup SPN has $p_i \otimes k_i = p_i^1 \otimes k_i^1 \| \dots \| p_i^n \otimes k_i^n$.

Remark 2.13. Let S_i be randomly chosen for all i values. When $(\mathbb{G}, \otimes) = (\mathbb{Z}_{2^n}, \oplus)$, the distribution of LDP values is studied in [198, 199]. These results are extended in [138], where the authors consider a generic abelian group $(\mathbb{G}, \otimes)$. When all the s-boxes are static[31], the distribution of LDPs for $(\mathbb{Z}_{2^n}, \oplus)$ is studied for example in [196, 65, 93].

[30] Note that $p_i^j, k_i^j \in \mathbb{G}$ for all j values.
[31] *i.e* are fixed and public for all of the SPN's implementations

2.3.3 Quasigroup Differential Cryptanalysis

In this section we extend the notion of differential cryptanalysis to quasigroup SPNs. After showing that our generalisation is correct, we use it to study the security of SPNs based on quasigroups isotopic to a group.

Definition 2.6. Let K be a key, $(\mathbb{G}, \otimes)$ a quasigroup and $\bullet \in \{\oslash, \oslash\}$. We define the quasigroup differential probabilities

$$DP_{\bullet}(\sigma, \alpha, \beta) = \frac{1}{|\mathbb{G}|} \sum_{\substack{X, X' \in \mathbb{G} \\ \Delta_{\bullet}(X, X') = \alpha}} [\Delta_{\bullet}(\sigma(X), \sigma(X')) = \beta],$$

$$KDP_{\oslash}(\sigma, \alpha, \beta, K) = \frac{1}{|\mathbb{G}|} \sum_{\substack{X, X' \in \mathbb{G} \\ \Delta_{\oslash}(X, X') = \alpha}} [\Delta_{\oslash}(\sigma(K \otimes X), \sigma(K \otimes X')) = \beta],$$

$$KDP_{\oslash}(\sigma, \alpha, \beta, K) = \frac{1}{|\mathbb{G}|} \sum_{\substack{X, X' \in \mathbb{G} \\ \Delta_{\oslash}(X, X') = \alpha}} [\Delta_{\oslash}(\sigma(X \otimes K), \sigma(X' \otimes K)) = \beta],$$

where $\sigma : \mathbb{G} \to \mathbb{G}$ is a permutation and $\alpha, \beta \in \mathbb{G}$.

Example 2.6. Let $\omega^{-1} = \{4, 7, 0, 5, 1, 2, 3, 6\}$, $\pi = \{6, 1, 5, 2, 3, 0, 4, 7\}$ and $\rho = \{5, 1, 2, 6, 4, 0, 7, 3\}$. Using Example 2.5 as a starting point, in Table 2.22 we present the difference distribution tables for $\otimes$ and σ. To see that in general DP is different from KDP, we also computed the keyed distribution tables for $K = 0$. The results are presented in Table 2.23.[32]

When $\mathbb{G}$ is an associative quasigroup[33], we managed to prove (Lemma 2.2) that key bits K have no influence on the input difference value $\Delta_{\bullet}$, where $\bullet \in \{\oslash, \oslash\}$, and, thus, can be ignored. In other words, a keyed s-box has the same difference distribution table as an unkeyed s-box (Corollary 2.1).

Lemma 2.2. If $\otimes$ is associative, then the following identities hold

$$\Delta_{\oslash}(K \otimes X, K \otimes X') = \Delta_{\oslash}(X, X')$$
$$\Delta_{\oslash}(X \otimes K, X' \otimes K) = \Delta_{\oslash}(X, X').$$

Proof. Using Lemma 2.1 we obtain

$$X \otimes \Delta_{\oslash}(X, X') = X \otimes (X \oslash X') = X',$$

[32] The code used to generate Tables 2.21 to 2.24 can be found at https://github.com/teseleanu/quasigroup_differential_4_bit.

[33] The need for associativity was pointed out to the author by one of the anonymous reviewers.

α \ β	0	1	2	3	4	5	6	7
0	5	0	0	1	1	0	1	0
1	0	2	1	1	1	1	2	0
2	1	1	3	1	2	0	0	0
3	0	1	0	3	0	1	1	2
4	0	1	1	0	3	0	1	2
5	1	1	0	2	1	3	0	0
6	1	2	1	0	0	1	3	0
7	0	0	2	0	0	2	0	4

(A) $|G| \cdot DP_{\oslash}(\sigma, \alpha, \beta)$

α \ β	0	1	2	3	4	5	6	7
0	3	2	1	0	0	0	1	1
1	1	3	0	0	1	2	1	0
2	0	0	3	1	1	2	1	0
3	0	1	2	3	0	0	1	1
4	1	0	0	1	3	0	2	1
5	2	0	0	2	0	4	0	0
6	1	1	1	1	2	0	2	0
7	0	1	1	0	1	0	0	5

(B) $|G| \cdot DP_{\oslash}(\sigma, \alpha, \beta)$

TABLE 2.22: Difference distribution tables for $\otimes$ and σ.

that leads to

$$\begin{aligned}\Delta_{\oslash}(K \otimes X, K \otimes X') &= (K \otimes X) \oslash (K \otimes X') \\ &= (K \otimes X) \oslash [K \otimes (X \otimes \Delta_{\oslash}(X, X'))] \\ &= (K \otimes X) \oslash [(K \otimes X) \otimes \Delta_{\oslash}(X, X')] \\ &= \Delta_{\oslash}(X, X').\end{aligned}$$

Similarly we prove the second equation. □

Corollary 2.1. If $\otimes$ is associative, then the following identities hold

$$KDP_{\oslash}(\sigma, \alpha, \beta, K) = DP_{\oslash}(\sigma, \alpha, \beta),$$
$$KDP_{\oslash}(\sigma, \alpha, \beta, K) = DP_{\oslash}(\sigma, \alpha, \beta).$$

α \ β	0	1	2	3	4	5	6	7
0	1	0	2	1	0	1	0	3
1	2	0	0	2	1	1	1	1
2	0	0	4	0	0	2	2	0
3	1	0	2	1	3	0	1	0
4	1	1	0	0	1	3	0	2
5	2	1	0	3	0	1	1	0
6	1	1	0	0	2	0	3	1
7	0	5	0	1	1	0	0	1

(A) $|G| \cdot KDP_{\oslash}(\sigma, \alpha, \beta, K)$

α \ β	0	1	2	3	4	5	6	7
0	0	1	0	1	1	5	0	0
1	5	0	1	0	0	0	1	1
2	1	0	5	0	0	0	1	1
3	0	1	0	1	5	1	0	0
4	1	0	1	0	0	0	1	5
5	0	1	0	5	1	1	0	0
6	0	5	0	1	1	1	0	0
7	1	0	1	0	0	0	5	1

(B) $|G| \cdot KDP_{\oslash}(\sigma, \alpha, \beta, K)$

TABLE 2.23: Keyed difference distribution tables for $\otimes$ and σ.

Proof. According to Definition 2.1, given X and K there exists an unique element Y such that $X = K \otimes Y$. Thus, we have

$$\begin{aligned}
DP_{\oslash}(\sigma, \alpha, \beta) &= \frac{1}{|\mathbb{G}|} \sum_{\substack{X, X' \in \mathbb{G} \\ \Delta_{\oslash}(X, X') = \alpha}} [\Delta_{\oslash}(\sigma(X), \sigma(X')) = \beta] \\
&= \frac{1}{|\mathbb{G}|} \sum_{\substack{K \otimes Y, K \otimes Y' \in \mathbb{G} \\ \Delta_{\oslash}(K \otimes Y, K \otimes Y') = \alpha}} [\Delta_{\oslash}(\sigma(K \otimes Y), \sigma(K \otimes Y')) = \beta] \\
&= \frac{1}{|\mathbb{G}|} \sum_{\substack{K \otimes Y, K \otimes Y' \in \mathbb{G} \\ \Delta_{\oslash}(Y, Y') = \alpha}} [\Delta_{\oslash}(\sigma(K \otimes Y), \sigma(K \otimes Y')) = \beta] \\
&= \frac{1}{|\mathbb{G}|} \sum_{\substack{Y, Y' \in \mathbb{G} \\ \Delta_{\oslash}(Y, Y') = \alpha}} [\Delta_{\oslash}(\sigma(K \otimes Y), \sigma(K \otimes Y')) = \beta] \\
&= KDP_{\oslash}(\sigma, \alpha, \beta, K),
\end{aligned}$$

where for the third equality we use Lemma 2.2. Similarly, we prove the second equation. □

To see if our definition is a generalization for the group differential probability, we must

recover LDP and RDP when $(\mathbb{G}, \otimes)$ is a group. We prove this in Corollary 2.2. Note that any group is associative and, according to Corollary 2.1, equivalence to DP suffices.

Lemma 2.3. If $(\mathbb{G}, \otimes)$ forms a group then the following identities hold

$$\Delta_{\oslash}(X, X') = \Delta_{\otimes}(X^{-1}, X'),$$
$$\Delta_{\oslash}(X, X') = \Delta_{\otimes}(X', X^{-1}).$$

Proof. Note that

$$\begin{aligned}\Delta_{\oslash}(X, X') = \alpha &\iff X \otimes \alpha = X' \\ &\iff X^{-1} \otimes X' = \alpha \iff \Delta_{\otimes}(X^{-1}, X') = \alpha.\end{aligned}$$

Similarly, we prove the second equation. □

Corollary 2.2. If $(\mathbb{G}, \otimes)$ forms a group then $DP_{\oslash}(\sigma, \alpha, \beta) = LDP_{\otimes}(\sigma, \alpha, \beta)$ and $DP_{\oslash}(\sigma, \alpha, \beta) = RDP_{\otimes}(\sigma, \alpha, \beta)$.

Proof. Note that

$$\begin{aligned}DP_{\oslash}(\sigma, \alpha, \beta) &= \frac{1}{|\mathbb{G}|} \sum_{\substack{X, X' \in \mathbb{G} \\ \Delta_{\oslash}(X, X') = \alpha}} [\Delta_{\oslash}(\sigma(X), \sigma(X')) = \beta] \\ &= \frac{1}{|\mathbb{G}|} \sum_{\substack{X, X' \in \mathbb{G} \\ \Delta_{\otimes}(X^{-1}, X') = \alpha}} [\Delta_{\otimes}(\sigma(X)^{-1}, \sigma(X')) = \beta] \\ &= LDP_{\otimes}(\sigma, \alpha, \beta).\end{aligned}$$

Similarly we prove the second equation. □

The action of deriving $\otimes$ from $\star$ gives rise to a natural question: what happens if we derive a new quasigroup operation $\hat{\otimes}$ from $\otimes$? Unfortunately, according to Lemma 2.4 we end up with another isotopy of $\star$. Thus, the problem of studying KDP for a chain of isotopies is reduced to studying KDP for an isotopy of the base operation $\star$.

Lemma 2.4. We define $x \hat{\otimes} y = \hat{\omega}^{-1}(\hat{\pi}(x) \otimes \hat{\rho}(y))$. Then there exist ω', π', ρ' such that $x \hat{\otimes} y = \omega'^{-1}(\pi'(x) \star \rho'(y))$.

Proof. Remark that

$$\begin{aligned} x \hat{\otimes} y &= \hat{\omega}^{-1}(\hat{\pi}(x) \otimes \hat{\rho}(y)) \\ &= \hat{\omega}^{-1}(\omega^{-1}(\pi(\hat{\pi}(x)) \star \rho(\hat{\rho}(y)))) \\ &= \omega'^{-1}(\pi'(x) \star \rho'(y)), \end{aligned}$$

where $\omega' = \hat{\omega} \circ \omega$, $\pi' = \hat{\pi} \circ \pi$ and $\rho' = \hat{\rho} \circ \rho$. □

When the base group $(\mathbb{G}, \star)$ is commutative we observe (Lemma 2.5) that taking into consideration both $\oslash$ and $\odot$ for designing an SPN does not make sense.

Lemma 2.5. We define $x \bar{\otimes} y = \omega^{-1}(\rho(x) \star \pi(y)) = z$, $x \bar{\oslash} z = y$ and $z \bar{\odot} y = x$. If $\star$ is commutative then the following identities hold

$$\begin{aligned} KDP_{\oslash}(\sigma, \alpha, \beta, K) &= KDP_{\bar{\odot}}(\sigma, \alpha, \beta, K), \\ KDP_{\odot}(\sigma, \alpha, \beta, K) &= KDP_{\bar{\oslash}}(\sigma, \alpha, \beta, K). \end{aligned}$$

Proof. The lemma's hypothesis implies that

$$\begin{aligned} x \otimes y &= \omega^{-1}(\pi(x) \star \rho(y)) \\ &= \omega^{-1}(\rho(x) \star \pi(y)) \\ &= y \bar{\otimes} x. \end{aligned}$$

Thus, $\Delta_{\oslash}(x, y) = \Delta_{\bar{\odot}}(y, x)$ for any $x, y \in \mathbb{G}$. Hence, $KDP_{\oslash}(\sigma, \alpha, \beta, K) = KDP_{\bar{\odot}}(\sigma, \alpha, \beta, K)$. The second statement is proven similarly. □

Corollary 2.3. If $\star$ is commutative and $\pi = \rho$ then we have $KDP_{\oslash}(\sigma, \alpha, \beta, K) = KDP_{\odot}(\sigma, \alpha, \beta, K)$.

We further study the impact of the ω, π, ρ permutations on KDP.

Lemma 2.6. Let $\pi' = \omega^{-1} \circ \pi$, $\rho' = \omega^{-1} \circ \rho$, $\sigma' = \omega^{-1} \circ \sigma \circ \omega$. We define $x * y = \pi'(x) \star \rho'(y) = z$, $x \backslash z = y$ and $z / y = x$. Then the following identities hold

$$\begin{aligned} KDP_{\oslash}(\sigma, \alpha, \beta, K) &= KDP_{\backslash}(\sigma', \omega(\alpha), \omega(\beta), \omega(K)) \\ KDP_{\odot}(\sigma, \alpha, \beta, K) &= KDP_{/}(\sigma', \omega(\alpha), \omega(\beta), \omega(K)). \end{aligned}$$

Proof. First we rewrite

$$KDP_{\oslash}(\sigma, \alpha, \beta, K) = \frac{1}{|\mathbb{G}|} \sum_{\substack{X \in \mathbb{G} \\ \Delta_{\otimes}(X, \alpha) = X'}} [\Delta_{\otimes}(\sigma(K \otimes X), \beta) = \sigma(K \otimes X')].$$

Let $\omega(X) = Y$, $\omega(X') = Y'$ and $\omega(\alpha) = A$. Then

$$\begin{aligned} X \otimes \alpha = X' &\iff \pi(X) \star \rho(\alpha) = \omega(X') \\ &\iff \pi'(\omega(X)) \star \rho'(\omega(\alpha)) = \omega(X') \\ &\iff \pi'(Y) \star \rho'(A) = Y' \\ &\iff Y * A = Y'. \end{aligned} \tag{2.1}$$

Let $\omega(K) = K'$. Then we obtain

$$\begin{aligned} \sigma(K \otimes X) &= \sigma(\omega^{-1}(\pi(K) \star \rho(X)) \\ &= \sigma(\omega^{-1}(\pi'(\omega(K)) \star \rho'(\omega(X)))) \\ &= \omega^{-1}(\sigma'(\pi'(K') \star \rho'(Y))) \\ &= \omega^{-1}(\sigma'(K' * Y)) \end{aligned} \tag{2.2}$$

and similarly

$$\sigma(K \otimes X') = \omega^{-1}(\sigma'(K' * Y')). \tag{2.3}$$

Let $\omega(\beta) = B$. Using Equations (2.2) and (2.3) we obtain

$$\begin{aligned} \sigma(K \otimes X) \otimes \beta = \sigma(K \otimes X') &\iff \omega^{-1}(\sigma'(K' * Y)) \otimes \beta = \omega^{-1}(\sigma'(K' * Y')) \\ &\iff \pi'(\sigma'(K' * Y)) \star \rho(\beta) = \sigma'(K' * Y') \\ &\iff \pi'(\sigma'(K' * Y)) \star \rho'(\omega(\beta)) = \sigma'(K' * Y') \\ &\iff \sigma'(K' * Y) * B = \sigma'(K' * Y'). \end{aligned} \tag{2.4}$$

Using Equations (2.1) and (2.4) we obtain

$$\begin{aligned} KDP_{\oslash}(\sigma, \alpha, \beta, K) &= \frac{1}{|\mathbb{G}|} \sum_{\substack{X \in \mathbb{G} \\ \Delta_{\otimes}(X,\alpha) = X'}} [\Delta_{\otimes}(\sigma(K \otimes X), \beta) = \sigma(K \otimes X')] \\ &= \frac{1}{|\mathbb{G}|} \sum_{\substack{Y \in \mathbb{G} \\ \Delta_{*}(Y,A) = Y'}} [\Delta_{*}(\sigma'(K' * Y), B) = \sigma'(K' * Y')] \\ &= \frac{1}{|\mathbb{G}|} \sum_{\substack{Y, Y' \in \mathbb{G} \\ \Delta_{\setminus}(Y,Y') = A}} [\Delta_{\setminus}(\sigma'(K' * Y), \sigma'(K' * Y')) = B] \\ &= KDP_{\setminus}(\sigma', A, B, K'). \end{aligned}$$

Similarly, we obtain $KDP_{\oslash}(\sigma, \alpha, \beta, K) = KDP_{/}(\sigma', A, B, K)$. □

Lemma 2.6 tells us that it is irrelevant from a differential point of view[34] if we define the quasigroup operation with $\omega \neq Id$ or $\omega = Id$. Thus, we further restrict our study[35] to the quasigroup operation $x \otimes y = \pi(x) \star \rho(y)$.

Lemma 2.7. Let $\pi' = \rho^{-1} \circ \pi$, $\sigma' = \rho^{-1} \circ \sigma \circ \rho$. We define $x *_1 y = \rho(\pi'(x) \star y) = z$, $x \backslash_1 z = y$ and $z /_1 y = x$. Then the following identity holds

$$KDP_{\oslash}(\sigma, \alpha, \beta, K) = KDP_{\backslash_1}(\sigma', \rho(\alpha), \rho(\beta), \rho(K)).$$

Proof. Let $\rho(X) = Y$, $\rho(X') = Y'$ and $\rho(\alpha) = A$. Then

$$\begin{aligned} X \otimes \alpha = X' &\iff \pi(X) \star \rho(\alpha) = X' \\ &\iff \rho(\pi'(\rho(X)) \star A) = \rho(X') \\ &\iff \rho(\pi'(Y) \star A) = Y' \\ &\iff Y *_1 A = Y'. \end{aligned} \tag{2.5}$$

Let $\rho(K) = K'$. Then we obtain

$$\begin{aligned} \sigma(K \otimes X) &= \sigma(\pi(K) \star \rho(X)) \\ &= \sigma(\pi'(\rho(K)) \star Y) \\ &= \rho^{-1}(\sigma'(\rho(\pi'(K') \star Y))) \\ &= \rho^{-1}(\sigma'(K' *_1 Y)) \end{aligned} \tag{2.6}$$

and similarly

$$\sigma(K \otimes X') = \rho^{-1}(\sigma'(K' *_1 Y')). \tag{2.7}$$

Let $\omega(\beta) = B$. Using Equations (2.6) and (2.7) we obtain

$$\begin{aligned} \sigma(K \otimes X) \otimes \beta = \sigma(K \otimes X') &\iff \rho^{-1}(\sigma'(K' *_1 Y)) \otimes \beta = \rho^{-1}(\sigma'(K' *_1 Y')) \\ &\iff \pi'(\sigma'(K' *_1 Y)) \star \rho(\beta) = \rho^{-1}(\sigma'(K' *_1 Y')) \\ &\iff \rho(\pi'(\sigma'(K' *_1 Y)) \star B) = \sigma'(K' *_1 Y') \\ &\iff \sigma'(K' *_1 Y) *_1 B = \sigma'(K' *_1 Y'). \end{aligned} \tag{2.8}$$

[34] *e.g.* we obtain the same differential probability KDP

[35] without loss of generality

Using Equations (2.5) and (2.8) we obtain

$$\begin{aligned} KDP_{\oslash}(\sigma,\alpha,\beta,K) &= \frac{1}{|\mathbb{G}|}\sum_{\substack{X\in\mathbb{G}\\ \Delta_{\otimes}(X,\alpha)=X'}}[\Delta_{\otimes}(\sigma(K\otimes X),\beta)=\sigma(K\otimes X')] \\ &= \frac{1}{|\mathbb{G}|}\sum_{\substack{Y\in\mathbb{G}\\ \Delta_{*_1}(Y,A)=Y'}}[\Delta_{*_1}(\sigma'(K'*_1 Y),B)=\sigma'(K'*_1 Y')] \\ &= KDP_{\setminus_1}(\sigma',A,B,K'). \end{aligned}$$

□

Lemma 2.8. Let $\rho' = \pi^{-1}\circ\rho$, $\sigma' = \pi^{-1}\circ\sigma\circ\pi$. We define $x *_2 y = \pi(x\star\rho'(y)) = z$, $x\setminus_2 z = y$ and $z/_2 y = x$. Then the following identity holds

$$KDP_{\oslash}(\sigma,\alpha,\beta,K) = KDP_{/_2}(\sigma',\pi(\alpha),\pi(\beta),\pi(K)).$$

Lemma 2.8 is proven similarly to Lemma 2.7 and, thus, its proof is omitted. Remark that our scope is to see how certain differences in the input affect the output of the non-linear layer. But our non-linear layer has either the form $\sigma(\rho(\pi(x)\star y))$ or the form $\sigma(\pi(x\star\rho(y)))$. Thus, a simpler strategy would be to study directly $\sigma_1 = \rho\circ\sigma$ and $\sigma_2\pi\circ\sigma$ instead of σ. Taking into account the previous remark, we further restrict our study to $x\otimes_1 y = \pi(x)\star y$ and $x\otimes_2 y = x\star\rho(y)$.

Example 2.7. Using Examples 2.5 and 2.6 as starting points, in Table 2.24 we present the difference distribution tables for $\otimes_1$ and $\otimes_2$.

Example 2.8. Let $\mathbb{G} = \mathbb{Z}_{256}$. To see how the maximum values for $LDP_{\oplus}$, $KDP_{\oslash_1}$ and $KDP_{\oslash_2}$ are distributed, we run the following experiment 10000 times[36]. We randomly generated π, ρ and then we computed the maximum values of $256\cdot LDP_{\oplus}$[37]. Then we generated 1000 keys and for each π and ρ we computed the mean value of the maximum values of $256\cdot KDP_{\oslash_1}$ and $256\cdot KDP_{\oslash_2}$. After gathering data from these experiments we computed the expected value $E[x]$ and the median absolute deviation MAD for each differential probability. The results are presented in Table 2.25.

We can see from Examples 2.5 and 2.7 that the difference distribution tables for $\oplus$, $\otimes_1$ and $\otimes_2$ have nothing in common. Also, Example 2.8 tells us that the average probability of success for a differential attack is lower in the case of $\otimes_1$ and $\otimes_2$ than in the case of $\oplus$. Thus, it might seem that we discovered a new method for improving SPNs.

[36] The associated code can be found at https://github.com/teseleanu/quasigroup_differential_8_bit.

[37] In this case we excluded the value 256.

α \ β	0	1	2	3	4	5	6	7
0	2	2	0	0	0	0	2	2
1	0	2	0	2	2	0	0	2
2	0	0	2	2	0	0	2	2
3	2	0	2	0	2	0	0	2
4	2	0	0	2	2	0	2	0
5	0	0	0	0	0	8	0	0
6	2	2	2	2	0	0	0	0
7	0	2	2	0	2	0	2	0

(A) $|G| \cdot KDP_{\oslash_1}(\sigma, \alpha, \beta, K)$

α \ β	0	1	2	3	4	5	6	7
0	2	2	2	2	0	0	0	0
1	2	0	0	2	2	0	0	2
2	0	0	2	2	2	0	2	0
3	0	2	0	2	0	0	2	2
4	2	2	0	0	2	0	2	0
5	0	0	0	0	0	8	0	0
6	2	0	2	0	0	0	2	2
7	0	2	2	0	2	0	0	2

(B) $|G| \cdot KDP_{\oslash_2}(\sigma, \alpha, \beta, K)$

TABLE 2.24: Difference distribution tables for $\otimes_1$ and $\otimes_2$.

	$LDP_{\oplus}$	$KDP_{\oslash_1}$	$KDP_{\oslash_2}$
$E[x]$	11.3550	7.56167	7.56204
MAD	1.067740	0.036824	0.036817

TABLE 2.25: Distribution of maximal differential probabilities.

Unfortunately, this is not the case. Let's review what we want to do. We want to study how input differences affect the output differences of a keyed s-box σ_K. Since K and, for example, π are generated as a pair, for a differential attack to work we do not really need to know K. The value $\pi(K)$ suffices. Thus, another way of studying the output differences of S_K is by using $\Delta_\star$. According to Lemma 2.9 the resulting differential probability is independent of $\pi(K)$. Hence, the choice for the permutation that acts on the key is irrelevant. This leads to the fact that using an isotopy is identical[38] to using the base operation.

Lemma 2.9. The following identities hold

$$\Delta_\star((\pi(K) \star X)^{-1}, \pi(K) \star X') = \Delta_\star(X^{-1}, X'),$$
$$\Delta_\star(X \star \pi(K), (X' \star \pi(K))^{-1}) = \Delta_\star(X, X'^{-1}).$$

[38] from a differential point of view

Proof. We simply remark that

$$\begin{aligned}\Delta_\star((\pi(K) \star X)^{-1}, \pi(K) \star X') &= X^{-1} \star \pi(K)^{-1} \star \pi(K) \star X' \\ &= X^{-1} \star X' = \Delta_\star(X^{-1}, X').\end{aligned}$$

Similarly we obtain the second equation. □

To summarise all the lemmas and observations we provide the reader with Proposition 2.1.

Proposition 2.1. A quasigroup SPN derived from a group SPN using an isotopy has the same differential security as the same group SPN instantiated with a different s-box.

Chapter 3

Public Key Cryptography

One of the problems associated with secret key cryptography is key distribution. An elegant solution for this inconvenience is provided by public/asymmetric key cryptography. In an asymmetric setting a participant possesses a pair of keys: a public key and an associated secret key. The public key is known by everybody and is bound to the participant's identity. Using the public key, any party can send messages to the owner, while he can read them using his secret key. Compared to secret key systems[1], in the public key setting there is no need for a secure channel in order to disseminate the participants' public keys. Another attractive property of asymmetric algorithms is that their security can, in most cases, be reduced to difficult computational problems.

Although initially developed for solving the key distribution problem, public key cryptography has expanded and incorporates other application such as encryption schemes, digital signatures or zero-knowledge protocols. In this chapter we develop various examples for the previously mentioned applications and relate their security to some well known hardness assumptions.

Conventions. For simplicity, public parameters will be implicit when describing an algorithm.

3.1 Hardness Assumptions

The building blocks of public key cryptographic schemes are based on intractable computational problems. By themselves, these computational problems do not solve any

[1] where a secure channel is needed to distribute the communication key to the participants

cryptographic problem relevant to a user's security goal. But, through careful manipulations, cryptographers manage to design primitives useful to this end. However, these primitives are secure only if we assume PPT adversaries[2].

In this section we provide the reader with two main classes of assumptions. The first class relates to the discrete logarithm problem over cyclic groups and the second to the factoring of composite integers.

3.1.1 Diffie-Hellman Assumptions

Definition 3.1 (Computational Diffie-Hellman - CDH)**.** Let $\mathbb{G}$ be a cyclic group of order q, g a generator of $\mathbb{G}$ and let A be a PPT algorithm that returns an element from $\mathbb{G}$ on input (g^x, g^y). We define

$$ADV^{\text{CDH}}_{\mathbb{G},g}(A) = Pr[A(g^x, g^y) = g^{xy} | x, y \xleftarrow{\$} \mathbb{Z}_q^*].$$

If $ADV^{\text{CDH}}_{\mathbb{G},g}(A)$ is negligible for any PPT algorithm A, we say that the *Computational Diffie-Hellman problem* is hard in $\mathbb{G}$.

Definition 3.2 (Decisional Diffie-Hellman - DDH)**.** Let $\mathbb{G}$ be a cyclic group of order q, g a generator of $\mathbb{G}$. Let A be a PPT algorithm which returns 1 on input (g^x, g^y, g^z) if $g^{xy} = g^z$. We define

$$\begin{aligned} ADV^{\text{DDH}}_{\mathbb{G},g}(A) = |&Pr[A(g^x, g^y, g^z) = 1 | x, y \xleftarrow{\$} \mathbb{Z}_q^*, z \leftarrow xy] \\ &- Pr[A(g^x, g^y, g^z) = 1 | x, y, z \xleftarrow{\$} \mathbb{Z}_q^*]|. \end{aligned}$$

If $ADV^{\text{DDH}}_{\mathbb{G},g}(A)$ is negligible for any PPT algorithm A, we say that the *Decisional Diffie-Hellman problem* is hard in $\mathbb{G}$.

Definition 3.3 (Hash Diffie-Hellman - HDH)**.** Let $\mathbb{G}$ be a cyclic group of order q, g a generator of $\mathbb{G}$ and $H : \mathbb{G} \rightarrow \mathbb{Z}_q^*$ a hash function. Let A be a PPT algorithm which returns 1 on input (g^x, g^y, z) if $H(g^{xy}) = z$. We define

$$\begin{aligned} ADV^{\text{HDH}}_{\mathbb{G},g,H}(A) = |&Pr[A(g^x, g^y, H(g^{xy})) = 1 | x, y \xleftarrow{\$} \mathbb{Z}_q^*] \\ &- Pr[A(g^x, g^y, z) = 1 | x, y, z \xleftarrow{\$} \mathbb{Z}_q^*]|. \end{aligned}$$

If $ADV^{\text{HDH}}_{\mathbb{G},g,H}(A)$ is negligible for any PPT algorithm A, we say that the *Hash Diffie-Hellman problem* is hard in $\mathbb{G}$.

[2] In practice all adversaries are computationally bounded.

Definition 3.4 (Entropy Smoothing - ES)**.** Let $\mathbb{G}$ be a cyclic group of order q, $\mathcal{K}$ the key space and $\mathcal{H} = \{h_i\}_{i \in \mathcal{K}}$ a family of keyed hash functions, where each h_i maps $\mathbb{G}$ to $\mathbb{Z}_q^*$. Let A be a PPT algorithm which returns 1 on input (i, y) if $y = h_i(z)$, where z is chosen at random from $\mathbb{G}$. Also, let We define

$$ADV_{\mathcal{H}}^{\text{ES}}(A) = |Pr[A(i, h_i(z)) = 1 | i \xleftarrow{\$} \mathcal{K}, z \xleftarrow{\$} \mathbb{G}] - Pr[A(i, h) = 1 | i \xleftarrow{\$} \mathcal{K}, h \xleftarrow{\$} \mathbb{Z}_q^*]|.$$

If $ADV_{\mathcal{H}}^{\text{ES}}(A)$ is negligible for any PPT algorithm A, we say that $\mathcal{H}$ is *Entropy Smoothing*. The action of choosing a random element from an entropy smoothing family $\mathcal{H}$ is further referred to as "H is ES".

Remark 3.1. In [95], the authors prove that CBC-MAC, HMAC and Merkle-Damgård constructions satisfy the above definition, as long as the underlying primitives satisfy some security properties.

Remark 3.2. The HDH assumption was formally introduced in [24, 25], although it was informally introduced as a composite assumption in [270, 48]. According to [48], the HDH assumption is equivalent with the CDH assumption in ROM. If the DDH assumption is hard in $\mathbb{G}$ and H is ES, then the HDH assumption is hard in $\mathbb{G}$ [24, 193, 223]. In [112], the authors show that the HDH assumption holds, even if the DDH assumption is relaxed to the following assumption: $\mathbb{G}$ contains a large enough group in which DDH holds. One particular interesting group is $\mathbb{Z}_p^*$, where p is a "large"[3] prime. According to [112], it is conjectured that if $\mathbb{G}$ is generated by an element $g \in \mathbb{Z}_p^*$ of order q, where q is a "large"[4] prime that divides $p - 1$, then the DDH assumption holds. The analysis conducted in [112] provides the reader with solid arguments to support the hypothesis that HDH holds in the subgroup $\mathbb{G} \subset \mathbb{Z}_p^*$.

Hashed Diffie-Hellman Key Exchange (HKE). Based on the HDH assumption we describe a key exchange protocol[5] in Figure 3.1. A formal analysis of this design can be found in [24, 25, 95].

[3] at least 2048 bits, better 3072 bits
[4] at least 192 bits, better 256 bits
[5] a high level description of the IKE protocols [136, 153]

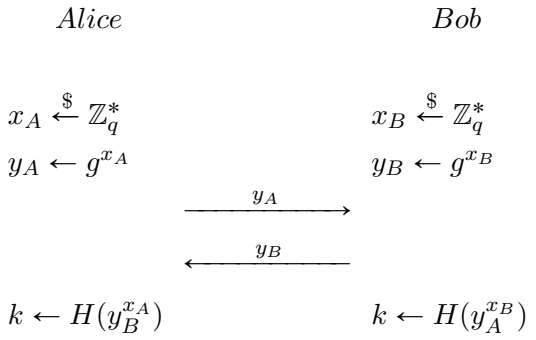


FIGURE 3.1: The Hashed Diffie-Hellman key exchange protocol.

3.1.2 2^k-th power residue

There are several ways to generalize the Legendre symbol to higher powers. We further consider the 2^k-th power residue symbol as presented in [259]. The classical Legendre symbol is obtained when $k = 1$.

Definition 3.5. Let p be an odd prime such that $2^k | p-1$. Then the symbol

$$\left(\frac{a}{p}\right)_{2^k} = a^{\frac{p-1}{2^k}} \bmod p$$

is called the 2^k-th power residue symbol modulo p, where $a^{\frac{p-1}{2^k}} \in \mathcal{Z}_p$.

Properties. The 2^k-th power residue symbol satisfies the following properties

1. If $a \equiv b \bmod p$, then $\left(\frac{a}{p}\right)_{2^k} = \left(\frac{b}{p}\right)_{2^k}$;
2. $\left(\frac{a^{2^k}}{p}\right)_{2^k} = 1$;
3. $\left(\frac{ab}{p}\right)_{2^k} = \left(\frac{a}{p}\right)_{2^k}\left(\frac{b}{p}\right)_{2^k} \bmod p$;
4. $\left(\frac{1}{p}\right)_{2^k} = 1$ and $\left(\frac{-1}{p}\right)_{2^k} = (-1)^{(p-1)/2^k}$.

Remark 3.3. The Jacobi symbol extends the Legendre symbol to composite moduli. If $n = p_1^{e_1} \cdots p_m^{e_m}$ is the prime factorization of the positive integer n, then the Jacobi symbol of a modulo n is

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \cdots \left(\frac{a}{p_m}\right)^{e_m}.$$

3.1.3 Modular Security Assumptions

Definition 3.6 (Quadratic Residuosity - QR, Squared Jacobi Symbol - SJS and Gap 2^k-Residuosity - GR)**.** Choose two large prime numbers $p, q \geqslant 2^\lambda$ and compute $n = pq$. Let A be a probabilistic polynomial-time (PPT) algorithm that returns 1 on input (x, n) or (x^2, n) or (x, k, n) if $x \in QR_n$ or J_n or $J_n \backslash QR_n$. We define

$$ADV_A^{\text{QR}}(\lambda) = \left| Pr[A(x,n) = 1 | x \xleftarrow{\$} QR_n] - Pr[A(x,n) = 1 | x \xleftarrow{\$} J_n \backslash QR_n] \right|,$$
$$ADV_A^{\text{SJS}}(\lambda) = \left| Pr[A(x^2,n) = 1 | x \xleftarrow{\$} J_n] - Pr[A(x^2,n) = 1 | x \xleftarrow{\$} \bar{J}_n] \right|,$$
$$ADV_{A,k}^{\text{GR}}(\lambda) = \left| Pr[A(x,k,n) = 1 | x \xleftarrow{\$} J_n \backslash QR_n] - Pr[A(x^{2^k},k,n) = 1 | x \xleftarrow{\$} \mathbb{Z}_n^*] \right|.$$

The *Quadratic Residuosity* assumption states that for any PPT algorithm A the advantage $ADV_A^{\text{QR}}(\lambda)$ is negligible.

If $p, q \equiv 1 \bmod 4$, then the *Squared Jacobi Symbol* assumption states that for any PPT algorithm A the advantage $ADV_A^{\text{SJS}}(\lambda)$ is negligible.

Let $p, q \equiv 1 \bmod 2^k$. The *Gap 2^k-Residuosity* assumption states that for any PPT algorithm A the advantage $ADV_A^{\text{GR}}(\lambda)$ is negligible.

Remark 3.4. In [51], the authors investigate the relation between the assumptions presented in Definition 3.6. They prove that for any PPT adversary A against the GR assumption, we have two efficient PPT algorithms B_1 and B_2 such that

$$ADV_{A,k}^{\text{GR}}(\lambda) \leqslant \frac{3}{2}\left((k - \frac{1}{3}) \cdot ADV_{B_1}^{\text{QR}}(\lambda) + (k-1) \cdot ADV_{B_2}^{\text{SJS}}(\lambda)\right).$$

3.2 Zero-Knowledge Protocols

The main issue addressed by ZKP is represented by *identification schemes* (entity authentication). Thus, building on the most important goal that a ZKP can achieve one may find elegant solutions to various problems that arise in different areas: digital cash, auctioning, IoT, password authentication and so on.

A typical zero knowledge protocol involves a prover *Peggy* which possesses a piece of secret information x associated with her identity and a verifier *Victor* whose job is to check that *Peggy* really owns x. Two classical examples of such protocols (proposed for smartcards) are the Schnorr protocol [219] and the Guillou-Quisquater protocol [131]. Working in an abstract framework, Maurer shows in [176] that the previously mentioned protocols are actually instantiations of the same one.

Building on Maurer's result, we considered of great interest providing the reader with a generalized perspective of the Unified Zero-Knowledge (UZK) protocol as well as a hash variant of it. An important consequence of our generic approach is the unification of Maurer's [176], Feige-Fiat-Shamir's [103] and Chaum-Everste-Van De Graaf's [68] protocols. Moreover, a special case of our protocol's hash version is the *h-variant* of the Fiat-Shamir scheme [108, 115].

As the IoT paradigm arised, lightweight devices[6] became more and more popular. Due to the open and distributed nature of the IoT, proper security is needed for the entire network to operate accordingly. Now let us consider the case of online wireless sensor networks (WSNs). The lightweight nature of sensor nodes heavily restricts cryptographic operations. Thus, the need for specific cryptographic solutions becomes obvious. The Fiat-Shamir-like distributed authentication protocol presented in [78] represents such an example. Based on this previous construction we propose a unified generic zero-knowledge protocol. Just as the result described in [78], our protocol can be applied for securing WSNs and, more generally, IoT-related solutions. Nonetheless, our construction offers flexibility when choosing the assumptions on which its security relies. A secondary feature of our scheme is the possibility of reusing existing certificates when implementing the distributed authentication protocol.

3.2.1 Preliminaries

3.2.1.1 Groups

Let $(\mathbb{G}, \star)$ and $(\mathbb{H}, \otimes)$ be two groups. We assume that the group operations $\star$ and $\otimes$ are efficiently computable.

Definition 3.7 (Homomorphism)**.** Let $f : \mathbb{G} \to \mathbb{H}$ be a function (not necessarily one-to-one). We say that f is a homomorphism if $f(x \star y) = f(x) \otimes f(y)$.

Throughout the rest of the paper we consider f to be a homomorphism as well as a one-way function[7]. To be consistent with [176], we denote the value $f(x)$ by $[x]$. Note that given $[x]$ and $[y]$ we can efficiently compute $[x \star y] = [x] \otimes [y]$, due to the fact that f is a homomorphism.

[6] low-cost devices with limited resources, be it computational or physical

[7] meaning that it is infeasible to compute x from $f(x)$

3.2.1.2 Zero-Knowledge Protocols

Let $Q : \{0,1\}^* \times \{0,1\}^* \to \{\texttt{true}, \texttt{false}\}$ be a predicate. Given a value y, Peggy will try to convince Victor that she knows a value x such that $Q(y,x) = \texttt{true}$. We further recall some definitions from [103, 176, 125, 176].

Definition 3.8 (Proof of Knowledge Protocol)**.** An interactive protocol (P,V) is a proof of knowledge protocol for predicate Q if the following properties hold

- Completeness: V accepts the proof when P has as input an x with $Q(y,x) = \texttt{true}$;
- Soundness: there is an efficient program K (called knowledge extractor) such that for any $\hat{P}$ (possibly dishonest) with non-negligible probability of making V accept the proof, K can interact with $\hat{P}$ and output (with overwhelming probability) an x such that $Q(y,x) = \texttt{true}$.

Definition 3.9 (Zero Knowledge Protocol)**.** A protocol (P,V) is zero-knowledge if for every efficient program $\bar{V}$ there exists an efficient program S, the simulator, such that the output of S is indistinguishable from a transcript of the protocol execution between P and $\bar{V}$. If the indistinguishability is perfect[8], then the protocol is called perfect zero-knowledge.

Definition 3.10 (2-Extractable)**.** Let Q be a predicate for a proof of knowledge. A 3-move protocol[9] with challenge space $\mathcal{C}$ is 2-extractable if from any two triplets (r,c,s) and (r,c',s'), with distinct $c,c' \in \mathcal{C}$ accepted by $Victor$, one can efficiently compute an x such that $Q(y,x) = \texttt{true}$.

According to [176], UZK (Figure 3.2) is a zero-knowledge protocol if the conditions from Theorem 3.1 are satisfied. If the challenge space $\mathcal{C}$ is small, then one needs several 3-move rounds to make the soundness error negligible. We further assume that UZK satisfies the conditions stated in Theorem 3.1.

Theorem 3.1. If values $\ell \in \mathbb{Z}$ and $u \in \mathbb{G}$ are known such that $\gcd(c_0 - c_1, \ell) = 1$ for all $c_0, c_1 \in \mathcal{C}$ with $c_0 \neq c_1$ and $[u] = y^\ell$, then the protocol described in Figure 3.2 is 2-extractable. Moreover, a protocol consisting of α rounds is a proof of knowledge if $1/|\mathcal{C}|^\alpha$ is negligible, and it is a zero-knowledge protocol if $|\mathcal{C}|$ is polynomially bounded.

[8] i.e. the probability distribution of the simulated and the actual transcript are identical
[9] in which *Peggy* sends r, *Victor* sends c, *Peggy* sends s

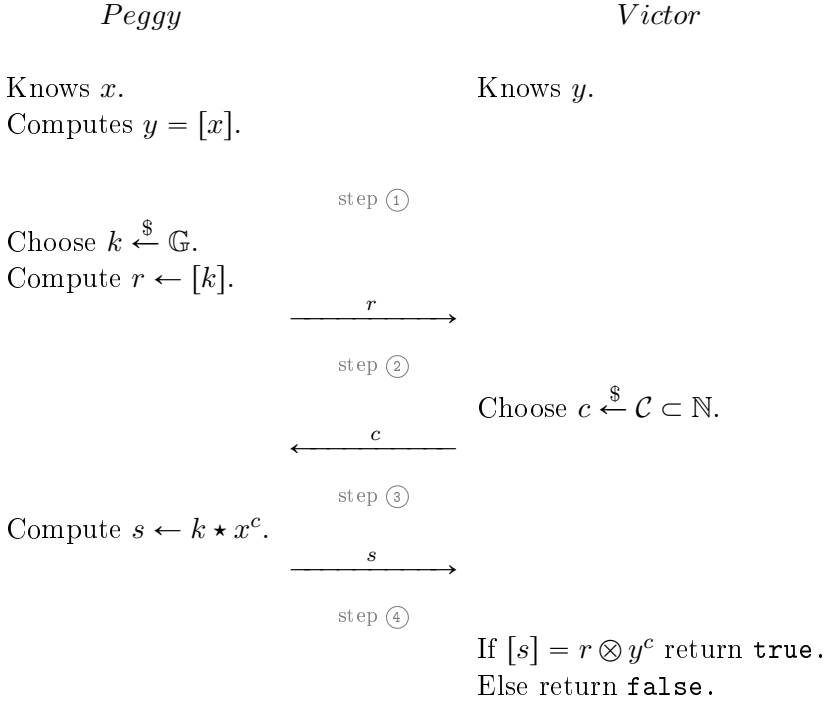


FIGURE 3.2: Maurer's Unified Zero-Knowledge (UZK) Protocol.

3.2.1.3 Hash Functions

In the following, we consider the definitions from [115]. These concepts are further applied in Section 3.2.4 within the security proof of our proposed generalization of the *h-variant* protocol [108].

Definition 3.11. Let $\delta \geqslant 2$ be an integer. An δ-collision for a hash function h is an δ-tuple $\{m_i\}_{i\in[1,\delta]}$ such that $h(m_1) = h(m_2) = \ldots = h(m_\delta)$.

Definition 3.12. Let $\delta \geqslant 2$ be an integer. A hash function is δ-collision resistant if it is computationally infeasible to find a δ-collision.

3.2.2 The Main Protocol

Inspired by Maurer's UZK protocol [176], we describe a UGZK protocol (Figure 3.3). Note that the UZK scheme is a special case of the UGZK construction. We also prove the security of our proposed construction in a Feige-Fiat-Shamir manner [103].

3.2.2.1 Description

Let n be a positive integer and let $i \in [1, n]$. For a given vector $\{z_i\}_{i\in[1,n]}$, the protocol in Figure 3.3 is a proof of knowledge[10] of a vector $\{[x_i]\}_{i\in[1,n]}$ such that $z_i = [x_i]$. The challenge spaces $\mathcal{C}_i$ for the elements c_i are chosen as arbitrary subsets of $\mathbb{N}$, for all $i \in [1, n]$. For the sake of uniformity, we assume that all the challenge spaces $\mathcal{C}_i$ are equal and we denote them by $\mathcal{C}$. If $|\mathcal{C}|$ is chosen to be small, then several rounds are needed in order to reduce the soundness error up to the point of being negligible.

When $n = 1$ we obtain the UZK protocol introduced in [176]. Note that in this case $\mathbb{G}$ and $\mathbb{H}$ need not be commutative.

Peggy		*Victor*
Knows $\{x_i\}_{i\in[1,n]}$. Computes $\{z_i\}_{i\in[1,n]} = \{[x_i]\}_{i\in[1,n]}$.		Knows $\{z_i\}_{i\in[1,n]}$.
Choose $k \xleftarrow{\$} \mathbb{G}$. Compute $t \leftarrow [k]$.		
	$\xrightarrow{t}$	
		Choose $c = \{c_i\}_{i\in[1,n]} \xleftarrow{\$} \mathcal{C}^n \subset \mathbb{N}^n$.
	$\xleftarrow{c}$	
Compute $r \leftarrow k \star (\star_{i=1}^{n} x_i^{c_i})$.		
	$\xrightarrow{r}$	
		If $[r] = t \otimes (\otimes_{i=1}^{n} z_i^{c_i})$ return `true`. Else return `false`.

FIGURE 3.3: A Unified Generic Zero-Knowledge (UGZK) Protocol.

3.2.2.2 Security Analysis

Theorem 3.2. Let $\mathbb{H}$ be a commutative group and let $j \in [1, n]$. If values $\ell_j \in \mathbb{Z}$ and $u_j \in \mathbb{G}$ are known such that

- $\gcd(c_j'' - c_j', \ell_j) = 1$ for all $c_j', c_j'' \in \mathcal{C}$ with $c_j' \neq c_j''$,
- $[u_j] = z_j^{\ell_j}$,

then by running the protocol described in Figure 3.3 for m rounds we obtain a proof of knowledge protocol if $1/|\mathcal{C}|^{nm}$ is negligible, and a zero-knowledge protocol if $|\mathcal{C}|^n$ is polynomially bounded.

[10] provided that the conditions of Theorem 3.2 are satisfied

Proof. Let $s = |\mathcal{C}|$. To prove that P's proof always convinces V, we evaluate the verification condition:

$$[r] = [k \star (\star_{i=1}^{n} x_i^{c_i})] = [k] \otimes (\otimes_{i=1}^{n} [x_i]^{c_i}) = t \otimes (\otimes_{i=1}^{n} z_i^{c_i}) .$$

Note that a corrupt $\bar{P}$ can cheat V with a negligible probability s^{-nm} per iteration by guessing the $\{c_i\}_{i\in[1,n]}$ vector, preparing $t = [k] \otimes \left(\otimes_{i=1}^{n} z_i^{-c_i}\right)$ in the first step, and providing $r = k$ in the last step.

Next, we show that whenever V accepts $\bar{P}$'s proof with non-negligible probability, there exists a knowledge extractor K that can print out all the x_is with overwhelming probability. Let T be the truncated execution tree of $(\bar{P}, V)$ for input I and random tape RA. As in [103, Theorem 3], the algorithm we construct explores this tree by repeatedly resetting $\bar{P}$ to the root, providing the necessary steering requests and verifying which one of the s sons of each explored vertex corresponds to a correct answer. V may ask s^n possible questions at each stage and, thus, the vertices in T may have polynomially many sons in terms of $|I|$. A vertex is called *heavy* if its degree is larger than s^{n-1} (i.e. if more than s^{n-1} executions of $(\bar{P}, V)$ at this state are successful). Our goal in this part of the proof is to show that all the x_is can be computed from the sons of a heavy vertex and that a PPT K can find a heavy vertex in T with overwhelming probability.

Let H be any heavy vertex in T and let Q be the set of queries in the form of vectors $\{c_i\}_{i\in[1,n]}$ which are properly answered by $\bar{P}$. It is easy to show that for any $1 \leqslant j \leqslant n$ a set Q of more than s^{n-1} vectors (having the length n) must contain two vectors $\{c'_i\}_{i\in[1,n]}$ and $\{c''_i\}_{i\in[1,n]}$ in which $c'_j \neq c''_j$ and $c'_i = c''_i$ for all $i \neq j$. Since both queries were properly answered, the two verification conditions imply

$$[r'_j] = t'_j \otimes \left(\otimes_{i=1}^{n} z_i^{c'_i}\right) \text{ and } [r''_j] = t''_j \otimes \left(\otimes_{i=1}^{n} z_i^{c''_i}\right) .$$

However, $\bar{P}$ must choose t before he obtains V's query and, thus, $t'_j = t''_j$. From r'_j and r''_j we can obtain $\tilde{x}_j$ such that $[\tilde{x}_j] = z_j$, as

$$\tilde{x}_j = u_j^{a_j} \star (r''^{-1}_j \star r'_j)^{b_j},$$

where a_j and b_j are computed using Euclid's extended gcd algorithm such that $\ell_j a_j + (c''_j - c'_j) b_j = 1$.

By rewriting the equations we get

$$\begin{aligned}[r_j''^{-1} \star r_j'] &= [r_j''^{-1}] \otimes [r_j'] \\ &= \left(\otimes_{i=n}^{1} z_i^{-c_i''}\right) \otimes t_j''^{-1} \otimes t_j' \otimes \left(\otimes_{i=1}^{n} z_i^{c_i'}\right) \\ &= \left(\otimes_{i=n}^{j} z_i^{-c_i''}\right) \otimes \left(\otimes_{i=j}^{n} z_i^{c_i'}\right) \\ &= z_j^{c_j'-c_j''},\end{aligned}$$

where for obtaining the last equality we used the commutative property of $\mathbb{H}$. Thus,

$$\begin{aligned}[\tilde{x}_j] &= [u_j^{a_j} \star (r_j''^{-1} \star r_j')^{b_j}] \\ &= [u_j]^{a_j} \otimes ([r_j''^{-1} \star r_j'])^{b_j} \\ &= (z_j^{\ell_j})^{a_j} \otimes (z_j^{c_j'-c_j''})^{b_j} \\ &= z_j^{\ell_j a_j + (c_j'-c_j'')b_j} \\ &= z_j.\end{aligned}$$

Now we show that at least half the vertices in at least one of the levels in T must be heavy. Let α_i be the ratio between the number of vertices at level $i+1$ and the number of vertices at level i in T. If $\alpha_i \leqslant (1/2s)s^n$ for all $1 \leqslant i \leqslant m$, then the total number of leaves in T (which is the product of all these α_i) is bounded by $(1/2s)^m s^{nm}$, which is a negligible fraction of the s^{nm} possible leaves. Since we assume that this fraction is polynomial, $\alpha_i > (1/2s)s^n$ for at least one i, and thus at least half the vertices at this level must contain more than s^n/s sons.

To find a heavy vertex in T, K chooses polynomially many random vertices at each level, and determines their degrees by repeated resets and executions of $\bar{P}$. To ensure a uniform probability distribution in spite of the uneven degrees of the vertices, M should explore random paths in the untruncated tree, and restart from the root whenever the path encounters an improperly answered query. Since a non-negligible fraction of the leaves is assumed to survive the truncation, this blind exploration of T can be carried out in polynomial time.

The last part of the proof deals with the zero-knowledge aspect of the protocol. By using resettable simulation in the sense of [125], the simulator S described in Algorithm 26 can mimic the communication in $(P, \bar{V})$ with an indistinguishable probability distribution in $O(ms^n)$ expected time, which is polynomial by our assumptions on s^n.

□

Algorithm 26. The simulator S.

Input: The public key $\{z_i\}_{i\in[1,n]}$
Output: A transcript $\mathcal{L}$

foreach $j \in [1, m]$ **do**
  Choose $c = \{c_i\}_{i\in[1,n]}$ at random from C^n
  Select a random number $r \xleftarrow{\$} \mathbb{G}$
  Compute $t \leftarrow [r] \otimes \left(\otimes_{i=n}^{1} z_i^{-c_i}\right)$
  Call $\bar{V}$ with input t and obtain a challenge c'
  if $c = c'$ **then**
    $L \leftarrow L \cup \{(t, c, r)\}$
  end
  else
    Reset $\bar{V}$'s state and repeat this round with new random choices
  end
end
return $\mathcal{L}$

3.2.3 Special Cases of the UGZK protocol

In this section we describe a number of protocols as instantiations of our main UGZK construction. Note that when $n = 1$ we obtain the UZK protocol from [176]. Thus, some schemes described in [176] are further reconsidered, while some examples are specific to our UGZK protocol. Although in the original paper [176] Maurer shows how to use UZK to prove the knowledge of a vector of secrets, our protocol UGZK is better in terms of transcript size.

3.2.3.1 Proofs of Knowledge of a Multiple Discrete Logarithm

Let $p = 2q + 1$ be a prime number such that q is also prime. Select an element $h \in \mathbb{H}_p$ of order q in some multiplicative group of order $p - 1$. The multiple discrete logarithm of a vector $\{z_i\}_{i\in[1,n]} \in \mathbb{H}_p^n$ is a vector of exponents $\{x_i\}_{i\in[1,n]}$ such that $z_i = h^{x_i}$, for all $i \in [1, n]$. We further describe a protocol for proving the knowledge of a multiple discrete logarithm.

A protocol for proving knowledge of a multiple discrete logarithm can be obtained as a special case of UGZK where $(\mathbb{G}, \star) = (\mathbb{Z}_q, +)$ and $\mathbb{H} = \langle h \rangle$. The one-way group homomorphism is defined by $[x] = h^x$, while the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, q - 1]$. The conditions of Theorem 3.2 are satisfied for $\ell_j = q$ and $u_j = 0$, where $j \in [1, n]$. When $n = 1$ we obtain the Schnorr protocol [219][11]. In the case $n \geqslant 1$ and $\mathcal{C} = \{0, 1\}$ we obtain the multiple logarithm protocol described in [68].

[11] This proof can be seen as a more efficient version of a proposal made by Chaum *et al.* [68].

Next we discuss a variation[11] of the previously presented protocol. Let $p = 2fp' + 1$ and $q = 2fq' + 1$ be prime numbers such that f, p' and q' are distinct primes. Select an element $h \in \mathbb{Z}_N^*$ of order f, where $N = pq$. Note that p and q are secret.

Using the UGZK notations we have $(\mathbb{G}, \star) = (\mathbb{Z}_f, +)$ and $\mathbb{H} = \langle h \rangle$. The one-way group homomorphism is defined by $[x] = h^x$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, f-1]$. We can observe that the conditions of Theorem 3.2 are satisfied for $\ell_j = f$ and $u_j = 0$, where $j \in [1, n]$. When $n = 1$ we obtain the Girault protocol [113].

3.2.3.2 Proofs of Knowledge of a Multiple e^{th}-root

Let p and q be two large prime numbers. Compute $N = pq$ and choose a prime e such that $\gcd(e, \varphi(N)) = 1$. A multiple e^{th}-root of a vector $\{z_i\}_{i\in[1,n]} \in (\mathbb{Z}_N^*)^n$ is a base vector $\{x_i\}_{i\in[1,n]}$ such that $z_i \equiv x_i^e \bmod N$. Note that the multiple e^{th}-root is not unique. We further describe a protocol for proving the knowledge of a multiple e^{th}-root.

Such a protocol can be obtained from UGZK with $(\mathbb{G}, \star) = (\mathbb{H}, \otimes) = (\mathbb{Z}_N^*, \cdot)$. The one-way group homomorphism is defined by $[x] = x^e \bmod N$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, e-1]$. The conditions of Theorem 3.2 are satisfied for $\ell_j = e$ and $u_j = z$, where $j \in [1, n]$. We stress that when $e = 2$ we obtain the protocol introduced by Feige, Fiat and Shamir [103]. In the case $n = 1$ we obtain the Guillou-Quisquater protocol [131][12].

3.2.3.3 Proofs of Knowledge of a Multiple Discrete Logarithm Representation

Let $p = 2q + 1$ be a prime number such that q is also prime. Select α elements $\{h_j\}_{j\in[1,\alpha]} \in \mathbb{H}_p^\alpha$ of order q in some multiplicative group of order $p-1$. A multiple discrete logarithm representation of a vector $\{z_i\}_{i\in[1,n]} \in (\langle h_1, \ldots, h_\alpha \rangle)^n$ is a vector of exponent vectors $(\{x_{1,j}\}_{j\in[1,\alpha]}, \ldots, \{x_{n,j}\}_{j\in[1,\alpha]})$ such that $z_i = h_1^{x_{i,1}} \ldots h_\alpha^{x_{i,\alpha}}$, for all $i \in [1, n]$. Note that multiple discrete logarithm representations are not unique. We further describe a protocol for proving the knowledge of a multiple discrete logarithm representation.

A protocol for proving the knowledge of a multiple representation can be instantiated from UGZK by setting $\mathbb{G} = \mathbb{Z}_q^\alpha$ with $\star$ defined as a component-wise addition operation and $\mathbb{H} = \langle h_1, \ldots, h_\alpha \rangle$. The one-way group homomorphism is defined by $[(x_1, \ldots, x_\alpha)] = h_1^{x_1} \ldots h_\alpha^{x_\alpha}$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, q-1]$. The conditions of Theorem 3.2 are satisfied for $\ell_j = q$ and $u_j = (0, \ldots, 0)$, where $j \in [1, n]$.

[12] This proof is a generalization of a protocol introduced by Fiat and Shamir [108].

When $n = 1$ we obtain a protocol proposed by Maurer in [176] which is a generalization of the protocols presented by Okamoto in [200] and Chaum *et.al.* in [68].

Chaum *et al.* [68] also provide a protocol variant for a composite n. Thus, by adapting the protocol presented in Section 3.2.3.1 and tweaking the previously described one, we can obtain a similar version for composite numbers. Using the notations from the protocol in Section 3.2.3.1, we set $\mathbb{G} = \mathbb{Z}_f^{\alpha}$ and $\mathbb{H} = \langle h_1, \ldots, h_m \rangle$, where $h_1, \ldots, h_\alpha \in \mathbb{Z}_n^*$ are elements of order f. The one-way group homomorphism is defined by $[(x_1, \ldots, x_\alpha)] = h_1^{x_1} \ldots h_\alpha^{x_\alpha}$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $\mathbb{Z}_f$. It is easy to see that $\ell_j = f$ and $u_j = (0, \ldots, 0)$, where $j \in [1, n]$.

3.2.3.4 Proofs of Knowledge of a Multiple e^{th}-root Representation

Let p and q be two large prime numbers. Compute $N = pq$ and choose primes $e_1, \ldots, e_\alpha$ such that $\gcd(e_i, \varphi(N)) = 1$, for $i \in [1, \alpha]$. A multiple e^{th}-root representation of a vector $\{z_i\}_{i \in [1,n]} \in (Z_N^*)^n$ is a vector of bases vector $(\{x_{1,j}\}_{j \in [1,\alpha]}, \ldots, \{x_{n,j}\}_{j \in [1,\alpha]})$ such that $z_i \equiv x_{i,1}^{e_1} \ldots x_{i,\alpha}^{e_\alpha} \bmod N$, for all $i \in [1, n]$. Note that multiple e^{th}-root representations are not unique. We further describe a protocol for proving the knowledge of a multiple e^{th}-root representation.

A protocol for proving the knowledge of a multiple e^{th}-root representation can be obtained from UGZK if we set $\mathbb{G} = (\mathbb{Z}_N^*)^\alpha$ with $\star$ defined as multiplication applied component-wise and $(\mathbb{H}, \otimes) = (\mathbb{Z}_N^*, \cdot)$. The one-way group homomorphism is defined by $[(x_1, \ldots, x_\alpha)] = x_1^{e_1} \ldots x_\alpha^{e_\alpha} \bmod N$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, e-1]$, where e is a prime such that $\gcd(e, \phi(N)) = 1$. Since all e_i are coprime then there exist β_is such that $\beta_1 e_1 + \ldots + \beta_\alpha e_\alpha = 1$. Then, it is easy to see that $\ell_j = 1$ and $u_j = (z_j^{\beta_1}, \ldots, z_j^{\beta_\alpha})$, where $j \in [1, n]$. When $n = 1$ we obtain a protocol introduced in [239].

3.2.4 Hash Protocol Variant

In order to decrease the number of communication bits, *Peggy* can hash t and send *Victor* the result. This method was proposed by Fiat and Shamir [108] and later analyzed in [115]. We employ the same technique for the protocol presented in Figure 3.3 and analyze its security.

3.2.4.1 Description

Let H be a hash function that maps elements from $\mathbb{H}$ into bit streams. The hash variant of the protocol works as follows: in the first step *Peggy* sends $H(t)$ to *Victor* (instead of t) and the last step becomes

> If $H(t) = H\left([r] \otimes \left(\otimes_{i=1}^{n} z_i^{-c_i}\right)\right)$ return `true`.
> Else return `false`.

3.2.4.2 Security Analysis

Theorem 3.3. Let $s = |\mathcal{C}|$. If there exists a PPT algorithm $\bar{P}$ such that the probability that $\bar{P}$ is accepted by an honest verifier is greater than $(\delta - 1)|\mathcal{C}|^{-n} + \varepsilon$, where $\varepsilon > 0$, then there exists a PPT algorithm $\tilde{P}$ which, with overwhelming probability, either inverts $[\cdot]$ or finds a δ-collision for h.

Proof. Let Ω be the set of $\tilde{p}$ elements in which $\tilde{P}$ picks its random values and E be the set $\mathcal{C}^n$, both of them characterized by the uniform distribution. For each value $(\omega, e) \in \Omega \times E$, $\tilde{P}$ passes the protocol (and we say it is a success) or not. Let S be the subset $\Omega \times E$ composed of all possible successes. Our assumption is that

$$\frac{|S|}{|\Omega \times E|} > (r-1)|\mathcal{C}|^{-n} + \varepsilon$$

with $\varepsilon > 0$ and $|\Omega \times E| = \tilde{p} \cdot s^n$.

Let $E_r = \{e \in E \mid (\omega, e) \text{ is a success}\}$ and $\Omega_r = \{\omega \in \Omega \mid |E_r| \geqslant r\}$. We have that

$$|S| \leqslant |\Omega_r| \cdot s^n + (r-1) \cdot (\tilde{p} - |\Omega_r|).$$

Thus,

$$\frac{|S|}{|\Omega \times E|} \leqslant \left[\frac{|\Omega_r|}{|\Omega|} + (r-1) \cdot \left(s^{-n} - \frac{|\Omega_r|}{|\Omega \times E|}\right)\right] \leqslant \frac{|\Omega_r|}{|\Omega|} + (r-1) \cdot s^{-n}$$

which implies

$$\frac{|\Omega_r|}{|\Omega|} \geqslant \varepsilon.$$

Let $\hat{P}$ be the PPT algorithm obtained by resetting $\tilde{P}$ ε^{-1} times. With constant probability, $\hat{P}$ picks ω in Ω_r and the probability can be made close to 1 by repeating the

execution of $\hat{P}$. At the end, δ values $\{r_i\}_{i\in[1,\delta]}$ are found such that, for distinct challenges $\{c_i\}_{i\in[1,\delta]} \in (C^n)^\delta$

$$H\left([r_1]\otimes\left(\otimes_{i=1}^n z_i^{-c_{i,1}}\right)\right) = H\left([r_2]\otimes\left(\otimes_{i=1}^n z_i^{-c_{i,2}}\right)\right) = \ldots = H\left([r_\delta]\otimes\left(\otimes_{i=1}^n z_i^{-c_{i,\delta}}\right)\right).$$

Now, we have two possibilities. In the first case, two of the values, say $[r_1]\otimes\left(\otimes_{i=1}^n z_i^{-c_{i,1}}\right)$ and $[r_2]\otimes\left(\otimes_{i=1}^n z_i^{-c_{i,2}}\right)$, are equal before hashing. Let $\mathcal{C}^- = \{-c \mid c \in \mathcal{C}\}$. Then, $[r_1 r_2^{-1}] = \left(\otimes_{i=1}^n z_i^{c'_i}\right)$, where $c' \in \mathcal{C}^- \cup \mathcal{C}$. This contradicts the intractability of $[\cdot]$. In the second case, all these values are pairwise distinct and a δ-collision for H has been found. This contradicts our assumption regarding H. □

Remark 3.5. This result suggests the use of hash-functions which are only resistant to δ-collisions (with $\delta > 2$), such that the hash values computed in the first pass can be made much shorter. Indeed, the decrease of the security level can be balanced by sending a slightly larger value of c in the second pass. More precisely, if $\delta = s^{n'}$, we choose $c \in \mathcal{C}^{n+n'}$ instead of $c \in \mathcal{C}^n$.

3.2.5 A Distributed Unified Protocol

A Fiat-Shamir-like distributed authentication protocol was proposed in [78]. Given our UGZK construction, we describe a generic collective authentication protocol which can be seen as a natural follow up of the main result in [78].

3.2.5.1 Description

Let us consider an n-node network consisting of $\mathcal{N}_1, ..., \mathcal{N}_n$. The nodes $\mathcal{N}_i$ can be seen as users and the base station $\mathcal{T}$ as a trusted center. To achieve the authentication of the entire network, we propose a unified Fiat-Shamir-like construction which we detail next.

1. Let x_i be a secret piece of information given to node $\mathcal{N}_i$. First, the network topology has to converge and a spanning tree needs to be constructed (*e.g.* with an algorithm similar with the one presented in [185]). Then, $\mathcal{T}$ sends an authentication request message to all the $\mathcal{N}_i$ directly connected to it, a message which contains a commitment to c (see 3.) to ensure the protocol's zero-knowledge property even against dishonest verifiers.

2. After receiving an authentication request message:

   - Each $\mathcal{N}_i$ generates a private k_i and computes $t_i \leftarrow [k_i]$;

- The $\mathcal{N}_i$s send authentication messages to all their (existing) children;
- After the children respond, nodes $\mathcal{N}_i$ compute $t_i \leftarrow t_i \otimes (\otimes_j t_j)$ and send the result up to their parents. Note that the t_js are sent by the nodes' children.

Such a construction permits the network to compute the $\otimes$ operation of all the t_is and send the result t_c to the top of the tree in d steps, where d represents the degree of the spanning tree. We refer the reader to Figure 3.4 for a toy example of this step.

3. $\mathcal{T}$ sends a random $c \in \mathcal{C}^n$ as an authentication challenge to the $\mathcal{N}_i$ directly connected to it.

4. After receiving an authentication challenge c:

   - Each $\mathcal{N}_i$ computes $r_i \leftarrow k_i \star x_i^{c_i}$;
   - The $\mathcal{N}_i$s then send the authentication challenge to all their (existing) children;
   - After the children respond, the $\mathcal{N}_i$s compute $r_i \leftarrow r_i \star (\star_j r_j)$ and send the result to their parents. Note that the r_js are sent by the nodes' children.

   The network therefore computes collectively the $\star$ operation of all the r_i's and transmits the result r_c to $\mathcal{T}$. Again, we refer the reader to Figure 3.4 for a toy example of this step.

5. After receiving r_c, $\mathcal{T}$ checks that $[r_c] = t_c \otimes (\otimes_{i=1}^n z_i^{c_i})$, where $z_1, \ldots, z_n$ are the public keys corresponding to $x_1, \ldots, x_n$ respectively.

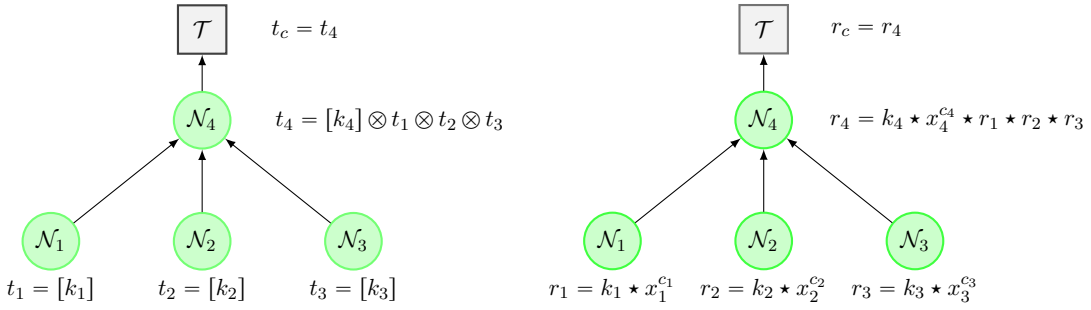


FIGURE 3.4: The proposed algorithm running on a network consisting of 4 nodes: computation of t_c (left) and of r_c (right).

Remark 3.6. The protocol we have just described may be interrupted at any step and such an action results in a failed authentication.

3.2.5.2 Security Analysis

Theorem 3.4. Let $\mathbb{H}$ be a commutative group and let $j \in [1, n]$. If an adversary corrupts $n' < n$ nodes and if values $\ell_j \in \mathbb{Z}$ and $u_j \in \mathbb{G}$ are known such that

- $\gcd(c''_j - c'_j, \ell_j) = 1$ for all $c'_j, c''_j \in \mathcal{C}$ with $c'_j \neq c''_j$,
- $[u_j] = z_j^{\ell_j}$,

then by running the protocol described in Section 3.2.5.1 for m rounds we obtain a proof of knowledge protocol if $1/|\mathcal{C}|^{(n-n')m}$ is negligible, and a zero-knowledge protocol if $|\mathcal{C}|^{(n-n')}$ is polynomially bounded.

Proof. If an adversary corrupts n' nodes, then n' secret keys x_i are known to him. Thus, the protocol is equivalent with a UGZK protocol with $n - n'$ secrets. Hence, using Theorem 3.2 we obtain our statement. □

3.2.5.3 Complexity Analysis

The number of operations necessary for authenticating the whole network depends on the topology. Precise complexity evaluations are given in Table 3.1. Note that each node performs in average only a few operations (a constant number).

Let d be the degree of the minimum spanning tree of the network. Then, only $O(d)$ messages are sent and, if we do not consider atypical cases, $d = O(\log n)$. Put differently, throughout the authentication process only a logarithmic number of messages is sent.

3.2.5.4 Variations

When implementing the distributed zero knowledge protocol several trade-offs are possible. Note that when doing so any combination of the trade-offs described below may be used.

Operation	Number of computations
$[\cdot]$	$(n+1)m$
Exponentiation	nm
$\otimes$	$\leqslant 2nm$
$\star$	$\leqslant 2nm$

TABLE 3.1: Complexity computations.

3.2.5.5 Hash based variant.

A distributed version of the UGZK protocol's hash variant (presented in Section 3.2.4) can be constructed. Using this "short commitment" version reduces somewhat the number of communicated bits, at the expense of a reduced security.

3.2.5.6 Short challenges variant.

In our protocol, the challenge c is sent throughout the network to all nodes. Assuming the use of an ideal hash function h, we may use shorter challenge without affecting security.

- A short c is sent to the nodes $\mathcal{N}_i$;
- Each $\mathcal{N}_i$ computes $c_i \leftarrow h(c\|i)$, and uses c_i as a challenge;
- The base station $\mathcal{T}$ computes c_i and uses it to check authentication.

3.2.5.7 Multiple-secret variant.

Each node $\mathcal{N}_i$ could use a set of secret values $\{x_{i,j}\}_{j\in[1,\ell]}$ instead of only one x_i. For the algorithm to be as efficient as possible the supplementary secrets can be expanded from a concealed seed. For clarity purposes we describe the multiple secret variant for a single node.

When receiving a challenge c_i, each node computes a response

$$r_i \leftarrow k_i \star \left(\star_{j=1}^{\ell} x_{i,j}^{c_{i,j}}\right).$$

This result be checked by the verifier by applying the next formula:

$$[r_i] = t_i \otimes \left(\otimes_{i=1}^{\ell} z_{i,j}^{c_{i,j}}\right).$$

In the case of multiple nodes, the modified protocol we obtain is a proof of knowledge if $1/|\mathcal{C}|^{(n-n')\ell m}$ is negligible and a zero-knowledge protocol if $|\mathcal{C}|^{(n-n')\ell}$ is polynomially bounded.

Practical aspects. Applying the multiple-secret variant, the trade-off between memory and communication can be adjusted, as the security level is ℓm (single-node compromission). Let μ be an integer. Therefore, if $\ell = \mu$ it suffices to authenticate once

to get the same security as $t = \mu$ authentications with $\ell = 1$[13]. It is obvious that such an approach significantly reduces bandwidth usage, a clearly desirable fact in the IoT context.

3.2.6 Future work

In order to take advantage of our main protocol's characteristics, an interesting research direction would be to apply it for obtaining generic versions of digital signature schemes [208, 209] and Section 3.3 and legally fair contract signing protocols [104] and Section 3.4. More generally, our proposal could be useful for future works on cryptographic protocol design. In the case of failed network authentication an open problem is to devise new batch verification algorithms or adapt the ones constructed for digital signatures [106, 107] for finding compromised nodes.

3.3 Signature Schemes

In 1986, Fiat and Shamir [108] described an important technique for deriving digital signatures from zero-knowledge protocols. At its core, the signer uses a hash function in order to create a virtual verifier. This technique was later used by Schnorr to transform his ZKP into a signature. The resulting signature was proven secure in ROM by Pointcheval and Stern [208, 209].

The UZK framework incorporates the Schnorr ZKP. Hence, it is natural to apply the Fiat-Shamir transform to UZK and thus extend Schnorr's signature. We will later use the resulting signature as the main building block for the contract signing protocol we propose in Section 3.3.2.

3.3.1 Preliminaries

Definition 3.13 (Signature Scheme)**.** A *Signature Scheme* consists of four PPT algorithms: *ParamGen*, *KeyGen*, *Sign* and *Verification*. The first one takes as input a security parameter and outputs the system's parameters. Using these parameters, the second algorithm generates the public key and the matching secret key. The secret key together with the *Sign* algorithm are used to generate a signature σ for a message m. Using only the public key, the last algorithm verifies if a signature σ for a message m is generated using the matching secret key.

[13] This corresponds to the protocol presented in Section 3.2.5.1.

Let us consider signature schemes which, on input a message m, produce triplets of the form $(\sigma_1, h(m\|\sigma_1), \sigma_2)$, independent of previous signatures. In these triplets we consider σ_2 as being dependent on m, σ_1 and $h(m\|\sigma_1)$. In some cases $h(m\|\sigma_1)$ is easily computable from the available data and, thus, can be omitted. For such signatures, the following security result can be proven [208, 209].

Lemma 3.1 (Forking Lemma)**.** Let $\mathcal{A}$ be a PPT algorithm, given only the public data as input. If $\mathcal{A}$ can find a valid signature[14] $(m, \sigma_1, h(m\|\sigma_1), \sigma_2)$ with non-negligible probability, then, also with non-negligible probability, a replay of $\mathcal{A}$ with a different hashing oracle h' outputs a second signature $(m, \sigma_1, h'(m\|\sigma_1), \sigma_2')$ such that $h(m\|\sigma_1) \neq h'(m\|\sigma_1)$.

Security Model. We further present the security model of [209] for signature schemes of type $(\sigma_1, h(m\|\sigma_1), \sigma_2)$.

Definition 3.14 (Signature Unforgeability - EF-CMA)**.** The notion of unforgeability for signatures is defined in terms of the following security game between the adversary $\mathcal{A}$ and a challenger:

KeyGen(λ): The challenger C generates the public key, sends it to adversary A and keeps the matching secret key to himself.

Query: Adversary A can perform any number of signature queries to the challenger.

Forgery: In this phase, the adversary outputs a tuple $(m, \sigma_1, h(m\|\sigma_1), \sigma_2)$.

$\mathcal{A}$ wins the game if $\mathsf{Verify}(m, \sigma_1, h(m\|\sigma_1), \sigma_2) = \mathsf{True}$ and $\mathcal{A}$ did not query the challenger on m. We say that a signature scheme is unforgeable when the success probability of $\mathcal{A}$ in this game is negligible.

Generalized ElGamal Signature. Originally described in [101], the ElGamal digital signature scheme can easily be generalized to any finite cyclic group $\mathbb{G}$ (see [180]). We shortly describe the algorithms of the generalized ElGamal signature scheme.

ParamGen(λ): Generate a large prime number q, such that $q \geqslant 2^\lambda$. Choose a cyclic group $\mathbb{G}$ of order q and let g be a generator of the group. Let $h : \mathbb{G} \to \mathbb{Z}_q$ be a hash function. Output the public parameters $pp = (q, g, \mathbb{G}, h)$.

KeyGen(pp): Choose $a \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $y \leftarrow g^a$. Output the public key $pk = y$. The secret key is $sk = a$.

[14] *i.e.* if the Verify algorithm outputs True for this signature

Sign(m, sk): To sign a message $m \in \mathbb{G}$, first generate a random number $k \xleftarrow{\$} \mathbb{Z}_q^*$. Then compute the values $r \leftarrow g^k$ and $s \leftarrow k^{-1}[h(m) - a \cdot h(r)] \bmod q$. Output the signature (r, s).

Verification(m, r, s, pk): To verify the signature (r, s) of message m, compute $v_1 \leftarrow y_V^{h(r)} \cdot r^s$ and $v_2 \leftarrow g^{h(m)}$. Output `true` if and only if $v_1 = v_2$. Else output `false`.

Schnorr Signature. In [219], Schnorr introduces a digital signature based on the discrete logarithm problem. Later on, the scheme was proven secure in the ROM by Stern and Pointcheval [208]. We further recall the Schnorr signature.

ParamGen(λ): Generate two large prime numbers p, q, such that $q \geqslant 2^\lambda$ and $q|p-1$. Select a cyclic group $\mathbb{G}$ of order p and let $g \in \mathbb{G}$ be an element of order q. Let $h : \{0,1\}^* \to \mathbb{Z}_q^*$ be a hash function. Output the public parameters $pp = (p, q, g, \mathbb{G}, h)$.

KeyGen(pp): Choose $x \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $y \leftarrow g^x$. Output the public key $pk = y$. The secret key is $sk = x$.

Sign(m, sk): To sign a message $m \in \{0,1\}^*$, first generate a random number $k \xleftarrow{\$} \mathbb{Z}_q^*$. Then compute the values $r \leftarrow g^k$, $e \leftarrow h(r \| m)$ and $s \leftarrow k - xe \bmod q$. Output the signature (e, s).

Verification(m, e, s, pk): To verify the signature (e, s) of message m, compute $r \leftarrow g^s y^e$ and $u \leftarrow h(r \| m)$. Output `true` if and only if $u = e$. Otherwise, output `false`.

3.3.2 A UZK Based Digital Signature Scheme

We describe our proposed UZK based digital signature scheme (further referred to as UDS) in Section 3.3.2.1. We further prove the security of our scheme in Section 3.3.2.2.

3.3.2.1 Description

By applying the Fiat-Shamir transform [108] to the UZK protocol in Figure 3.2 we obtain the following signature scheme.

ParamGen(λ): Select a group $\mathbb{G}$ and an homomorphism $[\cdot] : \mathbb{G} \to \mathbb{H}$. Also, let $h : \{0,1\}^* \to \mathcal{C}$ be a hash function, where $\mathcal{C} \subset \mathbb{N}$. Output the public parameters $pp = ([\cdot], \mathbb{G}, \mathbb{H}, h)$.

KeyGen(pp): Choose $x \xleftarrow{\$} \mathbb{G}$ and compute $y \leftarrow [x]$. Output the public key $pk = y$. The secret key is $sk = x$.

Sign(m, sk): To sign a message $m \in \{0,1\}^*$, first generate a random number $k \xleftarrow{\$} \mathbb{G}$. Then compute the values $r \leftarrow [k]$, $e \leftarrow h(r\|m)$ and $s \leftarrow k \star x^e$. Output the signature (r, s).

Verification(m, r, s, pk): To verify the signature (r, s) of message m, compute $u \leftarrow h(r\|m)$. Output `true` if and only if $[s] = r \otimes y^u$. Otherwise, output `false`.

3.3.2.2 Security Analysis

The proofs presented in [208, 209] do not cover the generic case. Thus, we adapt the initial results to the UDS case and provide the reader with the proof of Theorem 3.5.

Theorem 3.5. If an EF-CMA attack on the UDS has non-negligible probability of success in the ROM, then the homomorphism $[\cdot]$ can be inverted in polynomial time.

Proof. If an attacker $\mathcal{A}$ can forge a UDS, then we are able to construct a simulator $\mathcal{S}$ that interacts with $\mathcal{A}$ and forces it to produce a forgery. By using Lemma 3.1 we transform $\mathcal{A}$ into a homomorphism inverter (*i.e.* that computes an x' such that $y = [x']$). We further show how $\mathcal{S}$ can simulate the three phases necessary to mount the EF-CMA attack.

KeyGen Phase. In this phase $\mathcal{S}$ sets up the public key as $y = [x]$ and then activates $\mathcal{A}$ with input y.

Query Phase. $\mathcal{A}$ will start to present queries to the $\mathcal{S}$. Thus, $\mathcal{S}$ must respond to two types of queries: hash and signature queries. $\mathcal{S}$ will maintain a table T containing all the hash queries performed throughout the attack. At start $T \leftarrow \varnothing$. We further describe the simulations of the hash function in Algorithm 27 and the signature scheme in Algorithm 28.

Algorithm 27. Hashing oracle $\mathcal{O}_h$ simulation for h.

Input: A hashing query q_i from $\mathcal{A}$

if $\exists h_i, \{q_i, h_i\} \in T$ **then**
  $e \leftarrow h_i$
else
  $e \xleftarrow{\$} \mathcal{C}$
  Append $\{q_i, e\}$ to T
end
return e

Forgery Phase. After the query phase, $\mathcal{A}$ will eventually produce a forgery (r, s).

Algorithm 28. Signing oracle $\mathcal{O}_S$ simulation.

Input: A signature query m

$s \xleftarrow{\$} \mathbb{G}$
$e \xleftarrow{\$} \mathcal{C}$
$r \leftarrow [s] \otimes y^{-e}$
$u \leftarrow m \| r$
if $\exists e' \neq e, \{u, e'\} \in T$ **then**
  abort
else
  Append $\{u, e\}$ to T
end
return (r, s)

When simulating the signing oracle $\mathcal{O}_S$ there is a case when $\mathcal{S}$ aborts before completion: this happens when $m\|r$ has already been queried by $\mathcal{A}$. In this case, $\mathcal{S}$ can not reprogram $\mathcal{O}_h$, which is why it must abort. Since $\mathcal{A}$ does not know the random value r, the previously described event occurs with a negligible probability q_h/q, where q_h is the number of queries to $\mathcal{O}_h$.

Therefore, $\mathcal{A}$ is turned into a forger for the UDS with probability $(1-q_h/q)^{q_s} \geqslant 1-q_h q_s/q$, where q_s is the number of signing queries to $\mathcal{O}_S$. As $\mathcal{A}$ has a success probability ϵ_{succ}, the success probability of $\mathcal{A}$ in the simulated environment is $\epsilon_{sim} = (1 - q_h q_s/q)\epsilon_{succ}$.

Algorithm 29. Hashing oracle $\mathcal{O}'_h$ simulation for h.

Input: A hashing query q_i from $\mathcal{A}$, an index γ and a table T

if $i < \gamma$ **then**
  $e \leftarrow h_i$, where $(q_i, h_i) \in T$
  Append $\{q_i, e\}$ to $\tilde{T}$
else if $i = \gamma$ **then**
  $e \xleftarrow{\$} \mathcal{C} \backslash \{h_\gamma\}$, where $(q_\gamma, h_\gamma) \in T$
  Append $\{q_i, e\}$ to $\tilde{T}$
else
  if $\exists h_i, \{q_i, h_i\} \in \tilde{T}$ **then**
    $e \leftarrow h_i$
  else
    $e \xleftarrow{\$} \mathcal{C}$
    Append $\{q_i, e\}$ to $\tilde{T}$
  end
end
return e

Due to the ideal randomness of $\mathcal{O}_h$, $\mathcal{A}$ queries O_h on $m\|r$ with probability $1 - 1/c$, where $|\mathcal{C}| = c$. Hence, let γ be the position of $m\|r$ in T from $\mathcal{O}_h$. After $\mathcal{A}$ produces a forgery (r, s), $\mathcal{S}$ runs $\mathcal{A}$ with the same inputs and a different h oracle (Algorithm 29). As before,

$\mathcal{S}$ will maintain a table $\tilde{T}$ containing all the h queries performed throughout this phase of the attack. At start $\tilde{T} \leftarrow \varnothing$. Then, by Lemma 3.1, $\mathcal{A}$ will produce a different forgery (r, s'). Thus, we obtain $c = \mathcal{O}_h(m\|r) \neq \mathcal{O}'_h(m\|r) = c'$. Using the 2-extractable property of UZK, we obtain an x' such that $y = [x']$. □

3.4 Legally Fair Contract Signing Protocols

Various contract signing schemes which fall into three different design categories were proposed during the last decades: *gradual release* [122, 207, 111, 127], *optimistic* [29, 63, 181] and *concurrent* [71] or *legally fair* [104] models. A typical co-signing protocol involves two (mutually distrustful) signing partners, *Alice* and *Bob* wishing to compute a common function on their private inputs.

Compared to older paradigms like gradual release or optimistic models, concurrent signatures or legally fair protocols do not rely on trusted third parties and do not require too much interaction between co-signers. As such features seem much more attractive for users, we further consider legally fair co-signing protocols (rather than older solutions) in this section.

Inspired by Maurer's generic perspective, we considered of great interest extending the unification paradigm to contract signing protocols. Therefore, we construct our main idea considering the stringent issue of scheme compatibility which characterizes communication systems. Typical examples are the cases of certificates in a public key infrastructure and the general issue of upgrading the version of a system. Thus, working in a general framework may reduce implementation errors and save application development (and maintenance) time.

In this section we present a unified class of legally fair co-signing protocols without keystones and prove its security. To be more precise, we propose a class of UDS (see Section 3.3) based co-signing protocols that maintains the valuable properties[15] of the scheme presented in [104].

3.4.1 Preliminaries

In [104] the authors present a new contract signing paradigm that does not require keystones to achieve legal fairness. Their provably secure co-signature construction recalled in Figure 3.5 is based on Schnorr digital signatures [219].

[15] legal fairness without keystones, guaranteed output delivery

In Figure 3.5, $\mathcal{L}$ represents a local non-volatile memory used by Bob and $\mathcal{C} = [0, q-1]$. During the protocol, Alice makes use of a publicly known auxiliary signature scheme σ_{x_A} using her secret key x_A.

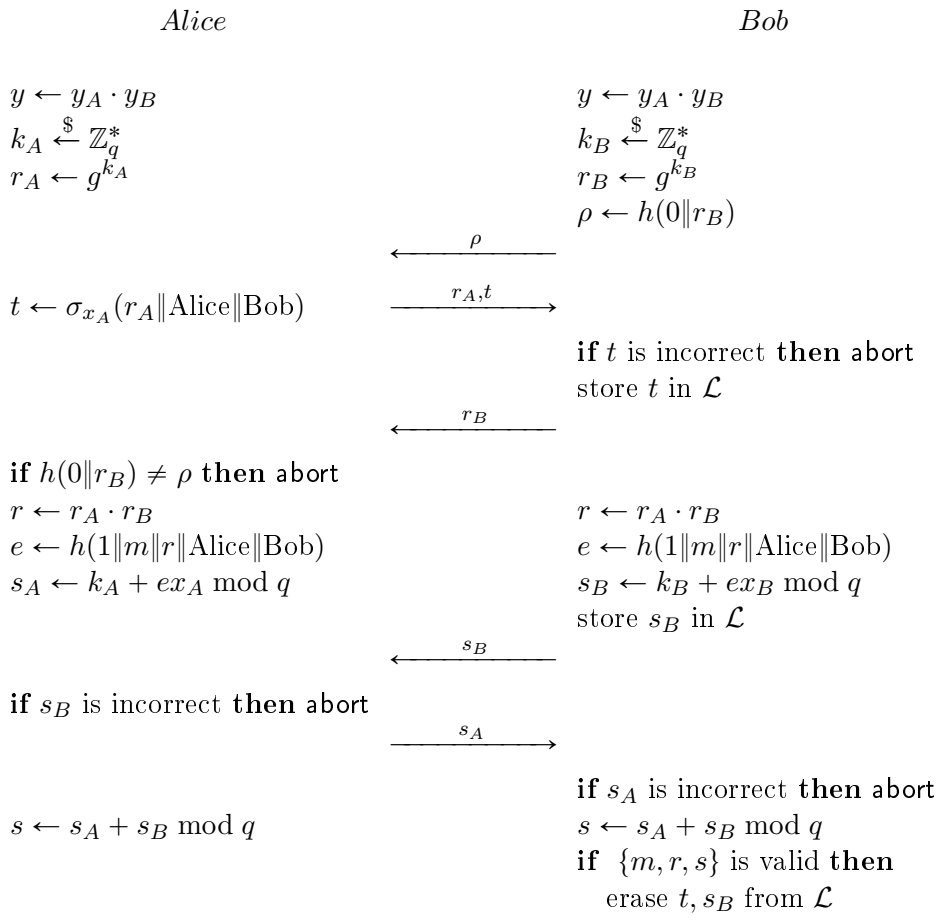


FIGURE 3.5: The legally fair signature (without keystones) of message m.

Security model. According to the analysis presented in [104], a legally fair signature scheme is secure when it achieves existential unforgeability against an active adversary $\mathcal{A}$ with access to an unlimited amount of conversations and valid co-signatures, *i.e.* $\mathcal{A}$ can perform the following queries:

- Hash queries: $\mathcal{A}$ can request the value of $h(x)$ for an x of his choosing.
- Sign queries: $\mathcal{A}$ can request a valid signature t for a message m and a public key y_C of his choosing.
- CoSign queries: $\mathcal{A}$ can request a valid co-signature (r, s) for a message m and a common public key $y_{C,D}$ of his choosing.

- Transcript queries: $\mathcal{A}$ can request a valid transcript $(m, \rho, r_C, t, r_D, s_C, s_D)$ of the co-signing protocol for a message m of his choosing, between users C and D of his choosing.
- SKExtract queries: $\mathcal{A}$ can request the private key corresponding to a public key.
- Directory queries: $\mathcal{A}$ can request the public key of any user.

The following definition captures the notion of unforgeability in the co-signing context:

Definition 3.15 (Co-Signature Unforgeability)**.** The notion of unforgeability for co-signatures is defined in terms of the following security game between the adversary $\mathcal{A}$ and a challenger:

KeyGen(λ): The challenger C generates all the public parameters and sends then to adversary A.

Query: Adversary $\mathcal{A}$ can perform any number of queries to the challenger, as described above.

Forgery: In this phase, the adversary outputs a tuple $(m, r, s, y_{C,D})$.

$\mathcal{A}$ wins the game if $\mathsf{Verify}(m, r, s) = \mathsf{True}$ and there exist public keys $y_C, y_D \in \mathcal{D}$ such that $y_{C,D} = y_C y_D$ and either of the following holds:

- $\mathcal{A}$ did not query SKExtract on y_C nor on y_D, and did not query CoSign on $(m, y_{C,D})$, and did not query Transcript on (m, y_C, y_D) nor (m, y_D, y_C).
- $\mathcal{A}$ did not query Transcript on (m, y_C, y_i) for any $y_i \neq y_C$ and did not query SKExtract on y_C, and did not query CoSign on (m, y_C, y_i) for any $y_i \neq y_C$.

We say that a co-signature scheme is unforgeable when the success probability of $\mathcal{A}$ in this game is negligible.

3.4.2 A Contract Signing Protocol

We describe our main result, a UZK class of legally fair contract signing protocols in Figure 3.6 and discuss its correctness. We further prove the security of our proposed idea in Section 3.4.2.2 based on the security of the UDS scheme we describe in Section 3.3.2.

Compared to the initial work on legally fair contract signing protocols without keystones [104], we give a more complete proof by taking into account the signature scheme σ too.

3.4.2.1 Description

To illustrate our unified paradigm, we now discuss a legally fair co-signing protocol built from the UDS (Figure 3.6), which produces signatures compatible with standard UDS. This contract signing protocol is provably secure in the ROM assuming the one-way property of $[\cdot]$.

We further consider a more restrictive set of initial conditions compared to [176], in the sense that we also assume that $\mathbb{G}$ is commutative[16].

Alice		*Bob*
$y \leftarrow y_A \otimes y_B$		$y \leftarrow y_A \otimes y_B$
$k_A \xleftarrow{\$} \mathbb{G}$		$k_B \xleftarrow{\$} \mathbb{G}$
$r_A \leftarrow [k_A]$		$r_B \leftarrow [k_B]$
		$\rho \leftarrow h(0 \Vert r_B)$
	$\xleftarrow{\rho}$	
$t \leftarrow \sigma_{x_A}(r_A \Vert \text{Alice} \Vert \text{Bob})$	$\xrightarrow{r_A, t}$	
		if t is incorrect **then** abort
		store t in $\mathcal{L}$
	$\xleftarrow{r_B}$	
if $h(0 \Vert r_B) \neq \rho$ **then** abort		
$r \leftarrow r_A \otimes r_B$		$r \leftarrow r_A \otimes r_B$
$e \leftarrow h(1 \Vert m \Vert r \Vert \text{Alice} \Vert \text{Bob})$		$e \leftarrow h(1 \Vert m \Vert r \Vert \text{Alice} \Vert \text{Bob})$
$s_A \leftarrow k_A \star x_A^e$		$s_B \leftarrow k_B \star x_B^e$
		store s_B in $\mathcal{L}$
	$\xleftarrow{s_B}$	
if s_B is incorrect **then** abort		
	$\xrightarrow{s_A}$	
		if s_A is incorrect **then** abort
$s \leftarrow s_A \star s_B \bmod q$		$s \leftarrow s_A \star s_B \bmod q$
		if $\{m, r, s\}$ is valid **then**
		erase t, s_B from $\mathcal{L}$

FIGURE 3.6: A class of legally fair co-signature schemes.

[16] The group $\mathbb{G}$ is considered as being generic in [176].

Correctness. To prove the correctness of the class of co-signing schemes described in Figure 3.6 we use the commutative property of $\mathbb{G}$ which is preserved by $f(x)$:

$$\begin{aligned}[s] &= [s_A \star s_B] \\ &= [s_A] \otimes [s_B] \\ &= [k_A] \otimes [x_A]^e \otimes [k_B] \otimes [x_B]^e \\ &= [k_A] \otimes [k_B] \otimes ([x_A] \otimes [x_B])^e \\ &= r \otimes y^e.\end{aligned}$$

3.4.2.2 Security Analysis

To prove that the unified co-signature protocol is secure in the ROM we use the following strategy: assuming $\mathcal{A}$ is an efficient forger for the co-signature scheme, we turn $\mathcal{A}$ into an efficient forger for UDS, then invoke Lemma 3.1 to prove the existence of an efficient inverter for the homomorphism $[\cdot]$. We further address two scenarios: when the attacker plays Alice's role, and when the attacker plays Bob's.

3.4.2.3 Adversary Attacks Bob

Theorem 3.6. If $\mathcal{A}_{\text{Alice}}$ plays the role of Alice and is able to forge a co-signature with non-negligible probability, then we can construct an EF-CMA attack on the UDS that has non-negligible probability of success.

Proof. The proof consists in constructing a simulator $\mathcal{S}_{\text{Bob}}$ that interacts with the adversary and forces it to actually produce a UDS forgery. Here is how this simulator behaves at each step of the protocol.

KeyGen Phase. $\mathcal{S}_{\text{Bob}}$ is given a target public key y. As a simulator, $\mathcal{S}_{\text{Bob}}$ emulates not only Bob, but also all oracles and the directory $\mathcal{D}$ (see Figure 3.7).

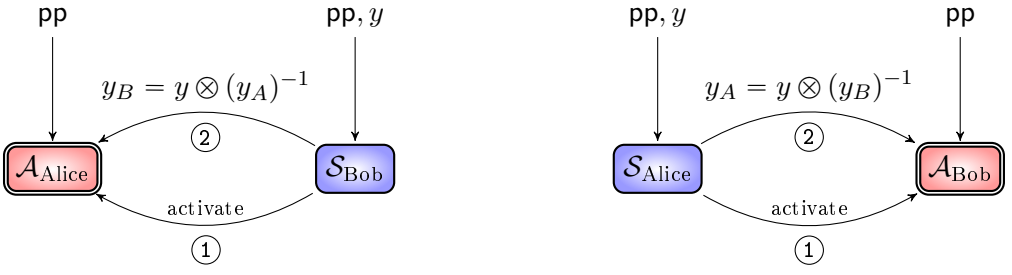


FIGURE 3.7: The simulator $\mathcal{S}_{\text{Bob}}$ (left) or $\mathcal{S}_{\text{Alice}}$ (right) answers the attacker's queries to the public directory $\mathcal{D}$.

To inject a target $y \leftarrow [x]$ into $\mathcal{A}$, the simulator $\mathcal{S}_{\text{Bob}}$ reads $y_{\mathcal{A}}$ from $\mathcal{D}$ and poses as an entity whose public-key is $y_{\mathcal{S}_{\text{Bob}}} \leftarrow y \otimes (y_{\mathcal{A}})^{-1}$. It follows that $y_{\mathcal{A},\mathcal{S}_{\text{Bob}}}$, the common public-key of $\mathcal{A}$ and $\mathcal{S}_{\text{Bob}}$ will be precisely $y_{\mathcal{A},\mathcal{S}_{\text{Bob}}} \leftarrow y_{\mathcal{S}_{\text{Bob}}} \otimes y_{\mathcal{A}}$ which, by construction, is exactly y.

Then $\mathcal{S}_{\text{Bob}}$ activates $\mathcal{A}_{\text{Alice}}$, who queries the directory and gets y_B. At this point in time, $\mathcal{A}_{\text{Alice}}$ is tricked into believing that she has successfully established a co-signature public-key y with the "co-signer" $\mathcal{S}_{\text{Bob}}$.

Query Phase. $\mathcal{A}_{\text{Alice}}$ will start to present queries to $\mathcal{S}$. Thus, $\mathcal{S}$ must respond to three types of queries: hash queries, co-signature queries and transcript queries. We consider oracle $\mathcal{O}_h$ as in Theorem 3.5. We further describe the simulation of the co-signature protocol in Algorithm 30. When $\mathcal{A}_{\text{Alice}}$ requests a conversation transcript, $\mathcal{S}_{\text{Bob}}$ replies by sending $(m, \rho, r_A, t, r_B, s_B, s_A)$ from a previously successful interaction.

Forgery Phase. After performing queries, $\mathcal{A}_{\text{Alice}}$ eventually outputs a co-signature (r, s) valid for $y_{\mathcal{A},\mathcal{S}_{\text{Bob}}}$ where $r = r_A \otimes r_B$ and $s = s_A \star s_B$. By design, these parameters are those of a UDS and therefore $\mathcal{A}_{\text{Alice}}$ has produced a UDS forgery.

Algorithm 30. Co-signing oracle simulation for $\mathcal{S}_{\text{Bob}}$.

Input: A co-signature query m from $\mathcal{A}_{\text{Alice}}$

$s_B \xleftarrow{\$} \mathbb{G}$
$e \xleftarrow{\$} \mathcal{C}$
$r_B \leftarrow [s_B] \otimes y^{-e}$
Send $h(0 \| r_B)$ to $\mathcal{A}_{\text{Alice}}$
Receive r_A, t from $\mathcal{A}_{\text{Alice}}$
Send r_B to $\mathcal{A}_{\text{Alice}}$
$r \leftarrow r_A \otimes r_B$
$u \leftarrow 1 \| m \| r \| \text{Alice} \| \text{Bob}$
if $\exists e' \neq e, \{u, e'\} \in T$ **then**
  abort
else
  Append $\{u, e\}$ to T
end
return s_B

To understand $\mathcal{S}_{\text{Bob}}$'s co-signature reply (Algorithm 30), assume that $\mathcal{A}_{\text{Alice}}$ is an honest Alice who plays by the protocol's rules. For such an Alice, (r, s) is a valid signature with respect to the co-signature public-key y.

There is a case in which $\mathcal{S}_{\text{Bob}}$ aborts the protocol before completion: this happens when it turns out that $1 \| m \| r \| \text{Alice} \| \text{Bob} \| t$ has been previously queried by $\mathcal{A}_{\text{Alice}}$. In that case, it is no longer possible for $\mathcal{S}_{\text{Bob}}$ to reprogram the oracle, which is why $\mathcal{S}_{\text{Bob}}$ must abort.

Since $\mathcal{A}_{\text{Alice}}$ does not know the random value r_B, such a bad event would only occur with a negligible probability exactly equal to q_h/q, where q_h is the number of queries to $\mathcal{O}_h$.

Therefore, $\mathcal{A}$ is turned into a forger for the SFS with probability $1 - q_h/q$. As $\mathcal{A}$ has a success probability ϵ_{succ}, the success probability of $\mathcal{A}$ in the simulated environment is $\epsilon_{sim} = (1 - q_h/q)\epsilon_{succ}$.

□

Corollary 3.1. If $\mathcal{A}_{\text{Alice}}$ plays the role of Alice and is able to forge a co-signature with non-negligible probability, then the homomorphism $[\cdot]$ can be inverted in polynomial time.

3.4.2.4 Adversary Attacks Alice

Theorem 3.7. If $\mathcal{A}_{\text{Bob}}$ plays the role of Bob and is able to forge a co-signature with non-negligible probability, then we can construct an EF-CMA attack on the UDS that has non-negligible probability of success if signature σ_{x_A} can be simulated without knowing the secret key x_A.

Proof. Here also the proof consists in constructing a simulator, $\mathcal{S}_{\text{Alice}}$, that interacts with the adversary and forces it to actually produce a UDS forgery. The simulator's behavior at different stages of the security game is as follows.

KeyGen Phase. $\mathcal{S}_{\text{Alice}}$ is given a target public key y. Again, $\mathcal{S}_{\text{Alice}}$ impersonates not only Alice, but also all the oracles and $\mathcal{D}$.

$\mathcal{S}_{\text{Alice}}$ injects the target y into the game as described in Theorem 3.6. Now $\mathcal{S}_{\text{Alice}}$ activates $\mathcal{A}_{\text{Bob}}$, who queries $\mathcal{D}$ (actually controlled by $\mathcal{S}_{\text{Alice}}$) to get y_A. $\mathcal{A}_{\text{Bob}}$ is thus tricked into believing that it has successfully established a co-signature public-key y with the "co-signer" $\mathcal{S}_{\text{Alice}}$.

Query Phase. $\mathcal{A}$ will start to present queries to $\mathcal{S}$. Thus, $\mathcal{S}$ must respond to four types of queries: hash queries, signature queries, co-signature queries and transcript queries. We consider oracles $\mathcal{O}_h$ as in Theorem 3.5. We denote by $\mathcal{O}_\sigma$ the simulation of σ_{x_A}. We further describe the simulation of the co-signature algorithm in Algorithm 31. When $\mathcal{A}_{\text{Alice}}$ requests a conversation transcript, $\mathcal{S}_{\text{Bob}}$ replies by sending $(m, \rho, r_A, t, r_B, s_B, s_A)$ from a previously successful interaction.

Forgery Phase. After performing queries, $\mathcal{A}_{\text{Bob}}$ eventually outputs a co-signature (r, s) valid for $y_{\mathcal{S}_{\text{Alice}}, \mathcal{A}_{\text{Bob}}}$ where $r = r_A \otimes r_B$ and $s = s_A \star s_B$. By design, these parameters are those of a UDS and therefore $\mathcal{A}_{\text{Bob}}$ has produced a UDS forgery.

Algorithm 31. Co-signing oracle simulation for $\mathcal{S}_{\text{Alice}}$.

Input: A co-signature query m from $\mathcal{A}_{\text{Bob}}$

Receive ρ from $\mathcal{A}_{\text{Bob}}$
Query T to retrieve r_B such that $h(0\|r_B) = \rho$
$s_A \xleftarrow{\$} \mathbb{G}$
$e \xleftarrow{\$} \mathcal{C}$
$r \leftarrow r_B \otimes [s_A] \otimes y^{-e}$
$u_1 \leftarrow 1\|m\|r$
if $\exists e' \neq e, \{u_1, e'\} \in T$ **then**
  abort
else
  Append $\{u_1, e\}$ to T
end
$r_A \leftarrow r \otimes r_B^{-1}$
$u_2 \leftarrow r_A\|\text{Alice}\|\text{Bob}$
$t \leftarrow \mathcal{O}_\sigma(u_2)$
Send r_A, t to $\mathcal{A}_{\text{Bob}}$
Receive r_B from $\mathcal{A}_{\text{Bob}}$
Receive s_B from $\mathcal{A}_{\text{Bob}}$
return s_A

As in Theorem 3.6, Algorithm 31 may fail with probability q_h/q. Thus, the success probability of $\mathcal{A}$ in the simulated environment is $\epsilon_{sim} = (1 - q_h/q)\epsilon_{succ}$.

□

Corollary 3.2. If $\mathcal{A}_{\text{Bob}}$ plays the role of Bob and is able to forge a co-signature with non-negligible probability, then the homomorphism $[\cdot]$ can be inverted in polynomial time if signature σ_{x_A} can be simulated without knowing the secret key x_A.

3.4.3 Future Work

A couple of interesting related studies could be the analysis of our co-signature protocols' resistance to SETUP (Secretly Embedded Trapdoor with Universal Protection) attacks and the proposal of suitable countermeasures.

3.5 Public Key Encryption

The scope of a public key encryption scheme is to provide confidentiality, while allowing users to distribute their public keys widely and openly. Therefore, only a user in

possession of the secret key can decrypt messages, while anyone in possession of the corresponding public key can encrypt data to send it to this one user. Usually, the design of PKEs is typically based on computationally intractable problems in number theory.

In this section we describe two PKE schemes, one based on discrete logarithms and one on factoring large numbers, that will be used in subsequent sections. Thus, the focus of this section will be on these two schemes and their security properties.

3.5.1 Preliminaries

Definition 3.16 (Public Key Encryption - PKE)**.** A *Public Key Encryption* scheme consists of three PPT algorithms: *KeyGen*, *Encrypt* and *Decrypt*. The first one takes as input a security parameter and outputs the system's parameters, the public key and the matching secret key. *Encrypt* takes as input the public key and a message and outputs the corresponding ciphertext. The *Decrypt* algorithm takes as input the secret key and a ciphertext and outputs either a valid message or an invalidity symbol (if the decryption failed).

Security Model. For PKE schemes several security notions have been proposed [49]. We further describe two security notions in the chosen plaintext scenario: one that protects messages and one that protect users' identity.

Definition 3.17 (Indistinguishability from Random Bits - IND$)**.** The security model of *indistinguishability from random bits* for a PKE scheme $\mathcal{AE}$ is captured in the following game:

KeyGen(λ): The challenger C generates the public key, sends it to adversary A and keeps the matching secret key to himself.

Query: Adversary A sends C a message m. The challenger encrypts m and obtains the ciphertext c_0. Let c_1 be a randomly chosen element from the same set as c_0. The challenger flips a coin $b \in \{0,1\}$ and returns c_b to the adversary.

Guess: In this phase, the adversary outputs a guess $b' \in \{0,1\}$. He wins the game, if $b' = b$.

The advantage of an adversary A attacking a PKE scheme is defined as

$$ADV^{\text{IND\$}}_{\mathcal{AE}}(A) = |2Pr[b = b'] - 1|,$$

where the probability is computed over the random bits used by C and A. A PKE scheme is IND$ secure, if for any PPT adversary A the advantage $ADV^{\text{IND\$}}_{\mathcal{AE}}(A)$ is negligible.

Definition 3.18 (Anonymity under Chosen Plaintext Attacks - ANO-CPA)**.** The security model against *anonymity under chosen plaintext attacks* for a PKE scheme $\mathcal{AE}$ is captured in the following game:

KeyGen(λ): The challenger C generates two public keys pk_0 and pk_1, sends them to adversary A and keeps the matching secret keys to himself.

Query: Adversary A sends C a message m. The challenger flips a coin $b \in \{0,1\}$ and encrypts m using pk_b. The resulting ciphertext c is sent to the adversary.

Guess: In this phase, the adversary outputs a guess $b' \in \{0,1\}$. He wins the game, if $b' = b$.

The advantage of an adversary A attacking a PKE scheme is defined as

$$ADV^{\text{ANO-CPA}}_{\mathcal{AE}}(A) = |2Pr[b = b'] - 1|,$$

where the probability is computed over the random bits used by C and A. A PKE scheme is ANO-CPA secure, if for any PPT adversary A the advantage $ADV^{\text{ANO-CPA}}_{\mathcal{AE}}(A)$ is negligible.

The ElGamal PKE scheme. The ElGamal encryption scheme was first described in [101] and later generalized in [180]. It can be proven that the generalized ElGamal encryption scheme is secure in the standard model under the DDH assumption [223]. We further describe the generalized version of the scheme and refer to it simply as the ElGamal encryption scheme (EG).

ParamGen(λ): Generate a large prime number q, such that $q \geqslant 2^{\lambda}$. Choose a cyclic group $\mathbb{G}$ of order q and let g be a generator of the group. Output the public parameters $pp = (q, g, \mathbb{G})$.

KeyGen(pp): Choose $x \xleftarrow{\$} \mathbb{Z}^*_q$ and compute $y \leftarrow g^x$. Output the public key $pk = y$. The secret key is $sk = x$.

Encryption(m, pk): To encrypt a message $m \in \mathbb{G}$, first generate a random number $k \xleftarrow{\$} \mathbb{Z}^*_q$. Then compute the values $c \leftarrow g^k$ and $d \leftarrow m \cdot y^k$. Output the pair (c, d).

Decryption(c, d, sk): To recover the original message compute $m \leftarrow d \cdot c^{-x}$.

The Joye-Libert PKE scheme. The Joye-Libert (JL) scheme was introduced in [149] and reconsidered in [51]. The scheme is proven secure in the standard model under the GR assumption. We shortly describe the algorithms of the Joye-Libert cryptosystem.

KeyGen(λ): Set an integer $k \geqslant 1$. Randomly generate two distinct large prime numbers p, q such that $p, q \geqslant 2^{\lambda}$ and $p, q \equiv 1 \bmod 2^k$. Output the public key $pk = (n, y, k)$, where $n = pq$ and $y \in J_n \backslash QR_n$. The corresponding secret key is $sk = (p, q)$.

Encrypt(pk, m): To encrypt a message $m \in [0, 2^k)$, we choose $x \xleftarrow{\$} \mathbb{Z}_n^*$ and compute $c \equiv y^m x^{2^k} \bmod n$. Output the ciphertext c.

Decrypt(sk, c): Compute $z \equiv \left(\frac{c}{p}\right)_{2^k}$ and find m such that the relation $\left[\left(\frac{y}{p}\right)_{2^k}\right]^m \equiv z \bmod p$ holds. Efficient methods to recover m can be found in [150].

3.5.2 Multiplicative ElGamal Encryption

The ElGamal encryption scheme was first described in [101]. The underlying group of the scheme is $\mathbb{Z}_p$, where p is a prime number. The scheme can easily be generalized to any finite cyclic group $\mathbb{G}$. The description of the generalized ElGamal can be found in [180]. Based on this description, we propose a new version of the ElGamal encryption scheme, which will later be used to deploy our SETUP mechanisms. We prove that the scheme is secure and that it preserves anonymity.

3.5.2.1 Description

KeyGen(λ): Generate a large prime number q, such that $q \geqslant 2^{\lambda}$. Choose a cyclic group $\mathbb{G}$ of order q and let g be a generator of the group. Let $H : \mathbb{G} \to \mathbb{Z}_q^*$ be a hash function. Choose $x \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $y \leftarrow g^x$. Output the system parameters $pp = (q, g, \mathbb{G}, H)$ and the public key $pk = y$. The secret key is $sk = x$.

Encryption(m, pk): To encrypt a message $m \in \mathbb{Z}_q^*$, first generate a random number $k \xleftarrow{\$} \mathbb{Z}_q^*$. Then compute the values $\alpha \leftarrow g^k$, $\beta \leftarrow y^k$, $\gamma \leftarrow H(\beta)$ and $\delta \leftarrow m \cdot \gamma$. Output the pair (α, δ).

Decryption(α, δ, sk): To decrypt ciphertext (α, δ), compute $\epsilon \leftarrow \alpha^x$, $\zeta \leftarrow H(\epsilon)$. Recover the original message by computing $m \leftarrow \delta \cdot \zeta^{-1}$.

We need to prove that the scheme is sound. If the pair (α, δ) is generated according to the scheme, it is easy to see that $\delta \cdot \zeta^{-1} \equiv m \cdot H(y^k) \cdot [H(\alpha^x)]^{-1} \equiv m \cdot H((g^x)^k) \cdot [H((g^k)^x)]^{-1} \equiv m$.

3.5.2.2 Security Analysis

In this section we prove that the Multiplicative ElGamal is a secure encryption scheme and it preserves anonimity. We denote by MEG the Multiplicative ElGamal scheme.

Theorem 3.8. MEG is IND$ secure in the standard model if and only if HDH is hard in $\mathbb{G}$. Formally, let A be an efficient PPT IND$ adversary. There exists an efficient algorithm B such that

$$ADV^{\text{IND\$}}_{MEG}(A) \leqslant 2ADV^{\text{HDH}}_{\mathbb{G},g,H}(B).$$

Proof. Let A be an IND$ adversary for MEG with access to "random coins" sampled uniformly from a set R. We construct an adversary B for the HDH assumption and then we provide an upper bound for the advantage of A.

Algorithm 32. The IND$ game.

1 Set the keys $x \xleftarrow{\$} \mathbb{Z}_q^*$ and $y \leftarrow g^x$,
2 Choose $\rho \xleftarrow{\$} R$ and initialize $m \leftarrow A(\rho, y)$
3 Select $b \xleftarrow{\$} \{0,1\}$ and run the encryption algorithm $k \xleftarrow{\$} \mathbb{Z}_q^*$, $\alpha_0 \leftarrow g^k$, $\beta \leftarrow y^k$, $\gamma \leftarrow H(\beta)$, $\delta_0 \leftarrow m \cdot \gamma$
4 Choose $\alpha_1 \xleftarrow{\$} \mathbb{G}$, $\delta_1 \xleftarrow{\$} \mathbb{Z}_q^*$ and $b \xleftarrow{\$} \{0,1\}$
5 **return** $A(\rho, y, \alpha_b, \delta_b)$

Algorithm 33. Algorithm B.

Input: $U \leftarrow g^u$ and $V \leftarrow g^v$, for random u, v, and W is either $H(g^{uv})$ or random
1 Set $y \leftarrow U$
2 Choose $\rho \xleftarrow{\$} R$ and initialize $m \leftarrow A(\rho, y)$
3 Set $\alpha_0 \leftarrow V$ and $\delta_0 \leftarrow m \cdot W$
4 Select $\alpha_1 \xleftarrow{\$} \mathbb{G}$, $\delta_1 \xleftarrow{\$} \mathbb{Z}_q^*$ and $b \xleftarrow{\$} \{0,1\}$
5 Initialize $b' \leftarrow A(\rho, y, \alpha_b, \delta_b)$
6 **if** $b = b$ **then**
7 | **return** 1
8 **else**
9 | **return** 0
10 **end**

Algorithm 32 describes the IND$ game. The first row sets up the public key. In the second row, A chooses the message m it wants to be challenged on. The challenger then picks a random k and computes the encryption (α_0, δ_0) of m. It also picks random choices (α_1, δ_1) from the same sampling sets, flips a bit b and reveals (α_b, δ_b). A then computes its guess b' for b. A wins if $b = b'$. Formally, the probability of A winning the IND$ game

is

$$|2Pr[b' = b] - 1| = ADV^{\text{IND\$}}_{MEG}(A). \tag{3.1}$$

Algorithm 33 depicts the behavior of an adversary B who runs the IND\$ distinguisher A as a subroutine. B is given as input U, V, W, where $U \leftarrow g^u$ and $V \leftarrow g^v$, for random u, v, and W is either $H(g^{uv})$ or random. Algorithm B outputs a bit indicating its guess for which of these cases occurs, where 1 means B guesses $W = H(g^{uv})$. Formally, the HDH advantage of B is

$$\begin{aligned} ADV^{\text{HDH}}_{\mathbb{G},g,H}(B) = |&Pr[B(U,V,W) = 1 | W = H(g^{uv})] \\ &- Pr[B(U,V,W) = 1 | W \xleftarrow{\$} \mathbb{Z}_q^*]|. \end{aligned} \tag{3.2}$$

Lets us consider the case $W = H(g^{uv})$ and compute the probability of B outputting 1. We note that B is running A as the latter would run an attack on the IND\$ security of MEG. Thus, we have

$$\begin{aligned} Pr[B(U,V,W) = 1 | W = H(g^{uv})] &= Pr[b = b' | W = H(g^{uv})] \\ &= \frac{1}{2}(2Pr[b = b' | W = H(g^{uv})] - 1 + 1) \\ &= \frac{1}{2} ADV^{\text{IND\$}}_{MEG}(A) + \frac{1}{2}. \end{aligned} \tag{3.3}$$

We will now compute the probability of B outputting 1 when W random. Then if we multiply an element m from $\mathbb{Z}_q^*$ with an uniformly random element W of the same set, we obtain an uniformly random element. Raising g to a random value v, yields a random element of $\mathbb{G}$ because g generates $\mathbb{G}$. Thus, $\alpha_0, \delta_0, \alpha_1, \delta_1$ are random. Since A has to choose between random elements, we have that

$$Pr[B(U,V,W) = 1 | W \xleftarrow{\$} \mathbb{Z}_q^*] = Pr[b = b' | W \xleftarrow{\$} \mathbb{Z}_q^*] = \frac{1}{2}. \tag{3.4}$$

Finally, the statement is proven by combining the equalities (3.1) – (3.4).

□

Theorem 3.9. *MEG* is ANO-CPA secure in the standard model if and only if HDH is hard in $\mathbb{G}$. Formally, let A be an efficient PPT ANO-CPA adversary. There exists an efficient algorithm B such that

$$ADV^{\text{ANO-CPA}}_{MEG}(A) \leqslant 4 ADV^{\text{HDH}}_{\mathbb{G},g,H}(B).$$

Proof. Let A be an ANO-CPA adversary for MEG with access to "random coins" sampled uniformly from a set R. We construct two adversaries B_1, B_2 for the HDH assumption and then we provide an upper bound for the advantage of A.

Algorithm 34. The ANO-CPA game.

Set the keys $x_0 \xleftarrow{\$} \mathbb{Z}_q^*$, $y_0 \leftarrow g^{x_0}$, $x_1 \xleftarrow{\$} \mathbb{Z}_q^*$ and $y_1 \leftarrow g^{x_1}$
Choose $\rho \xleftarrow{\$} R$ and initialize $m \leftarrow A(\rho, y_0, y_1)$
Select $b \xleftarrow{\$} \{0,1\}$ and run the encryption algorithm $k \xleftarrow{\$} \mathbb{Z}_q^*$, $\alpha \leftarrow g^k$, $\beta \leftarrow y_b^k$, $\gamma \leftarrow H(\beta)$, $\delta \leftarrow m \cdot \gamma$
return $A(\rho, y_0, y_1, \alpha, \delta)$

Algorithm 35. Algorithm B.

Input: $U \leftarrow g^u$ and $V \leftarrow g^v$, for random u, v, and W is either $H(g^{uv})$ or random
Set $y_0 \leftarrow U$, $z \xleftarrow{\$} \mathbb{Z}_q^*$, $y_1 \leftarrow g^z$, $\mu_0 \leftarrow H(V^z)$ and $\mu_1 \xleftarrow{\$} \mathbb{Z}_q^*$
Choose $\rho \xleftarrow{\$} R$ and initialize $m \leftarrow A(\rho, y_0, y_1)$
Select $b \xleftarrow{\$} \{0,1\}$ and set $\alpha \leftarrow V$, $\omega_0 \leftarrow W$, $\omega_1 \leftarrow \mu_b$
Select $b' \xleftarrow{\$} \{0,1\}$ and compute $\delta \leftarrow m \cdot \omega_{b'}$
Initialize $b'' \leftarrow A(\rho, y_0, y_1, \alpha, \delta)$
if $b' = b''$ **then**
  return 1
else
  return 0
end

Algorithm 34 describes the ANO-IND game. The first row sets up the public keys y_0 and y_1. In the second row, the adversary selects the message m it wants to be challenged on. The challenger then flips a bit b, chooses a random k and it reveals the encryption of m under y_b. A then computes its guess b' for b. A wins if $b = b'$. Formally, the probability of A winning the ANO-IND game is

$$|2Pr[b' = b] - 1| = ADV_{MEG}^{\text{ANO-CPA}}(A). \tag{3.5}$$

Algorithm 35 depict the behavior of algorithm B who runs the ANO-IND distinguisher A as a subroutine. B is given as input U, V, W, where $U \leftarrow g^u$ and $V \leftarrow g^v$, for random u, v, and W is either $H(g^{uv})$ or random. B outputs a bit indicating its guess for which of these cases occurs, where 1 means B guesses $W = H(g^{uv})$. Formally, the HDH advantage of B is

$$\begin{aligned} ADV_{\mathbb{G},g,H}^{\text{HDH}}(B) = |&Pr[B(U,V,W) = 1 | W = H(g^{uv})] \\ &- Pr[B(U,V,W) = 1 | W \xleftarrow{\$} \mathbb{Z}_q^*]|. \end{aligned} \tag{3.6}$$

Let us consider the case $W = H(g^{uv})$ and compute the probability of B outputting 1. There are two sub-cases when $b = 0$ and when $b = 1$. In the former sub-case, we note that B is running A as the latter would run an attack on the ANO-IND security of MEG. Thus, we have

$$\begin{aligned} Pr[B(U,V,W) = 1 | W = H(g^{uv})] &= Pr[b' = b'' | W = H(g^{uv})] \\ &= Pr[b' = b'' | W = H(g^{uv}), b = 0] Pr[b = 0] \\ &+ Pr[b' = b'' | W = H(g^{uv}), b = 1] Pr[b = 1] \\ &= \frac{1}{4}(2Pr[b = b' | W = H(g^{uv}), b = 0] - 1 + 1) \\ &+ \frac{1}{2} Pr[b' = b'' | W = H(g^{uv}), b = 1] \\ &= \frac{1}{4} ADV^{\text{ANO-CPA}}_{MEG}(A) + \frac{1}{4} \\ &+ \frac{1}{2} Pr[b' = b'' | W = H(g^{uv}), b = 1]. \end{aligned} \tag{3.7}$$

The probability of B outputting 1 when W is random is

$$\begin{aligned} Pr[B(U,V,W) = 1 | W \xleftarrow{\$} \mathbb{Z}_q^*] &= Pr[b' = b'' | W = H(g^{uv})] \\ &= Pr[b' = b'' | W \xleftarrow{\$} \mathbb{Z}_q^*, b = 0] Pr[b = 0] \\ &+ Pr[b' = b'' | W \xleftarrow{\$} \mathbb{Z}_q^*, b = 1] Pr[b = 1]. \end{aligned} \tag{3.8}$$

Lets consider the sub-case $b = 1$. If we multiply an element m from $\mathbb{Z}_q^*$ with an uniformly random element ω_0 or ω_1 of the same set, we obtain an uniformly random element. Then A has two decide between two pairs that have the same distribution. Thus, we have

$$Pr[b' = b'' | W \xleftarrow{\$} \mathbb{Z}_q^*, b = 1] = \frac{1}{2}. \tag{3.9}$$

In the sub-case $b = 0$, we have that

$$Pr[b' = b'' | W \xleftarrow{\$} \mathbb{Z}_q^*, b = 0] = Pr[b' = b'' | W = H(g^{uv}), b = 1], \tag{3.10}$$

since in both case A receives one random element and one of the form $H(V^e)$, where e is random. Thus, equality (3.8) becomes

$$Pr[B(U,V,W) = 1 | W \xleftarrow{\$} \mathbb{Z}_q^*] = \frac{1}{2} Pr[b' = b'' | W = H(g^{uv}), b = 1] + \frac{1}{4} \tag{3.11}$$

Finally, the statement is proven by combining the equalities (3.5) – (3.11).

□

3.5.3 A Generalisation of the Goldwasser-Micali Cryptosystem

The authors of [149] introduced a PKE scheme[17] representing a rather natural extension of the Goldwasser-Micali (GM) [123, 124] cryptosystem, the first probabilistic encryption scheme. The Goldwasser-Micali cryptosystem achieves ciphertext indistinguishability under the *Quadratic Residuosity* (QR) assumption. Despite being simple and stylish, this scheme is quite uneconomical in terms of bandwidth[18]. Various attempts of generalizing the Goldwasser-Micali scheme were proposed in the literature in order to address the previously mentioned issue. The Joye-Libert scheme can be considered a follow-up of the cryptosystems proposed in [190] and [79] and efficiently supports the encryption of larger messages.

Inspired by the Joye-Libert scheme, we propose a new public key cryptosystem, analyze its security and provide the reader with an implementation and performance discussion. We construct our scheme based on 2^k-th power residue symbols. Our generalization of the Joye-Libert cryptosystem makes use of two important parameters when it comes to the encryption and decryption functions: the number of bits of a message and the number of distinct primes of a public modulus n. Thus, our proposal not only supports the encryption of larger messages (as in the Joye-Libert variant), but also operates on *a variable number of large primes* (instead of two in the Joye-Libert case). Both these parameters can be chosen depending on the desired security application.

Our scheme can be viewed as a flexible solution characterized by the ability of making adequate trade-offs between encryption speed and ciphertext expansion in a given context.

3.5.3.1 Description

KeyGen(λ): Set an integer $k \geqslant 1$. Randomly generate $\gamma+1$ distict large prime numbers $p_i, i \in [0, \gamma+1)$ such that $p_i \geqslant 2^\lambda$ and $p_i \equiv 1 \bmod 2^k$. Let $n = p_0 \cdot \ldots \cdot p_\gamma$. Select $y_i \xleftarrow{\$} \mathbb{Z}_n^*, i \in [0, \gamma)$, such that the following conditions hold

1. $\left(\frac{y_i}{p_i}\right) = -1$;
2. $\left(\frac{y_i}{p_\gamma}\right) = -1$;
3. $\left(\frac{y_i}{p_j}\right)_{2^k} = 1$, where $j \neq i$.

[17] reconsidered in [51]

[18] $k \cdot \log_2 n$ bits are needed to encrypt a k-bit message, where n is an RSA modulus as described in [123, 124]

We denote by $y = \{y_i\}_{i\in[0,\gamma)}$ and $p = \{p_i\}_{i\in[0,\gamma)}$. Output the public key $pk = (n, y, k)$. The secret key is $sk = p$.

Encrypt(pk, m): To encrypt message $m \in [0, 2^k\gamma)$, first we divide it into γ blocks $m = m_0\| \ldots \|m_{\gamma-1}$. Then, we choose $x \xleftarrow{\$} \mathbb{Z}_n^*$ and compute $c \equiv x^{2^k} \cdot \prod_{i=0}^{\gamma-1} y_i^{m_i} \bmod n$. The output is ciphertext c.

Decrypt(sk, c): For each $i \in [0, \gamma)$, compute $m_i = Dec_{p_i}(p_i, y_i, c)$.

Algorithm 36. $Dec_{p_i}(p_i, y_i, c)$.

Input: The secret prime p_i, the value y_i and the ciphertext c
Output: The message block m_i

$m_i \leftarrow 0$, $B \leftarrow 1$
foreach $s \in [1, k+1)$ **do**
  $z \leftarrow \left(\frac{c}{p_i}\right)_{2^s}$
  $t \leftarrow \left(\frac{y_i}{p_i}\right)_{2^s}$
  $t \leftarrow t^{m_i} \bmod p_i$
  if $t \neq z$ **then**
    $m_i \leftarrow m_i + B$
  end
  $B \leftarrow 2B$
end
return m_i

Correctness. Let $m_i = \sum_{w=0}^{k-1} b_w 2^w$ be the binary expansion of block m_i. Note that

$$\left(\frac{c}{p_i}\right)_{2^s} = \left(\frac{x^{2^k} \cdot \prod_{v=0}^{\gamma-1} y_v^{m_v}}{p_i}\right)_{2^s} = \left(\frac{y_i^{m_i}}{p_i}\right)_{2^s} = \left(\frac{y_i}{p_i}\right)_{2^s}^{\sum_{w=0}^{s-1} b_w 2^w}$$

since

1. $\left(\frac{x^{2^k}}{p_i}\right)_{2^s} = 1$, where $1 \leqslant s \leqslant k$;
2. $\left(\frac{y_j}{p_i}\right)_{2^k} = 1$, where $j \neq i$;
3. $\sum_{w=0}^{k-1} b_w 2^w = \left(\sum_{w=0}^{s-1} b_w 2^w\right) + 2^s \cdot \left(\sum_{w=s}^{k-1} b_w 2^{w-s}\right)$.

As a result, the message block m_i can be recovered bit by bit using p_i.

Remark 3.7. The case $\gamma = k = 1$ corresponds to the Goldwasser-Micali cryptosystem [123] and the case $\gamma = 1$ corresponds to the Joye-Libert PKE scheme [149].

Remark 3.8. In the *KeyGen* phase, we have to compute a special type of y_i. An efficient way to perform this step is to randomly select $y_{i,i} \xleftarrow{\$} \mathbb{Z}^*_{p_i}$, $y_{i,\gamma} \xleftarrow{\$} \mathbb{Z}^*_{p_\gamma}$ and $w_j \xleftarrow{\$} \mathbb{Z}_{p_j^*}$, compute $y_j \leftarrow w_j^{2^k} \bmod p_j$ and finally use the Chinese remainder theorem to compute an element $y_i \in \mathbb{Z}^*_n$ such that $y_i \equiv y_{i,\ell} \bmod p_\ell$.

3.5.3.2 Security Analysis

Theorem 3.10. Assume that the QR and SJS assumptions hold. Then, the proposed scheme is IND$ secure in the standard model. Formally, let A be an efficient PPT adversary, then there exist two efficient PPT algorithms B_1 and B_2 such that

$$ADV_A^{\text{IND\$}}(\lambda) \leqslant \frac{3}{2}\gamma\left((k-\frac{1}{3})\cdot ADV_{B_1}^{\text{QR}}(\lambda) + (k-1)\cdot ADV_{B_2}^{\text{SJS}}(\lambda)\right).$$

Proof. To prove the statement, we simply replace the distribution of the public key y for the encryption query. Let $n_i = p_i p_\gamma$, $i \in [0,\gamma)$. Instead of choosing $y_i \in J_{n_i} \backslash QR_{n_i}$ we choose y_i from the multiplicative subgroup of 2^k residues modulo n_i. Under the GR assumption, the adversary does not detect the difference between the original scheme and the one with the modified y_is. In this case, the value c is not carrying any information about the message. Thus, the IND-CPA security of our proposed cryptosystem follows. □

Remark 3.9. Note that in Theorem 3.10 is sufficient to consider the GR assumption modulo n_i instead of modulo n. To prove this, lets consider an efficient PPT distinguisher B for the GR assumption modulo n. Then we construct an efficient distinguisher C for the GR assumption modulo n_i.

Thus, on input (y_i, k, n_i), C first randomly selects $\gamma - 1$ primes $\{p_j\}_{j\in[0,\gamma)\backslash\{i\}}$ such that $p_j \equiv 1 \bmod 2^k$ and computes $n = n_i \cdot \prod_{j\in[0,\gamma)\backslash\{i\}} p_j$. Then, using the Chinese theorem, C computes a value $\bar{y_i}$ such that $\bar{y_i} \equiv y_i \bmod n_i$ and $\bar{y_i} \equiv 1 \bmod n/n_i$. Finally, C sends $(\bar{y_i}, k, n)$ to B and he outputs B answer. It is easy to see that C and B have the same success probability.

3.5.3.3 Complexity Analysis

In our performance analysis we use the complexities of the mathematical operations listed in Table 3.2. These complexities are in accordance with the algorithms presented in [82]. We do not use the explicit complexity of multiplication, but instead we refer to it as $M(\cdot)$ for clarity.

Operation	**Complexity**
Multiplication	$M(\mu) = \mathcal{O}(\mu \log(\mu) \log(\log(\mu)))$
Exponentiation	$\mathcal{O}(kM(\mu))$
Jacobi symbol	$\mathcal{O}(\log(\mu)M(\mu))$

TABLE 3.2: Computational complexity for μ-bit numbers and k-bit exponents.

For simplicity, when computing the ciphertext expansion, the encryption and the decryption complexities, we consider the length of the prime numbers as being λ. Based on the complexities presented in Table 3.2, we obtain the results listed in Table 3.3.

Scheme	**Ciphertext size**	**Encryption Complexity**
GM [123]	$2\lambda \cdot \eta$	$\mathcal{O}(2M(2\lambda)\eta)$
JL [149]	$2\lambda \cdot \lceil \frac{\eta}{k} \rceil$	$\mathcal{O}(2(k+1)M(2\lambda)\lceil \frac{\eta}{k} \rceil)$
This work	$(\gamma+1) \cdot \lambda \cdot \lceil \frac{\eta}{\gamma k} \rceil$	$\mathcal{O}((\gamma+1)(k+1)M((\gamma+1)\lambda)\lceil \frac{\eta}{\gamma k} \rceil)$

Scheme	**Decryption Complexity**
GM [123]	$\mathcal{O}(\log(\lambda)M(\lambda)\eta)$
JL [149]	$\mathcal{O}((2k\lambda + \frac{k^2}{2})M(\lambda)\lceil \frac{\eta}{k} \rceil)$
This work	$\mathcal{O}(\gamma(2k\lambda + \frac{k^2}{2})M(\lambda)\lceil \frac{\eta}{\gamma k} \rceil)$

TABLE 3.3: Performance analysis for an η-bit message.

3.5.3.4 Implementation Details

We further provide the reader with benchmarks for our proposed PKE scheme.

We ran each of the three sub-algorithms on a CPU Intel i7-4790 4.00 GHz and used GCC to compile it (with the O3 flag activated for optimization). Note that for all computations we used the GMP library [19]. To calculate the running times we used the *omp_get_wtime()* function [15]. To obtain the average running time we chose to encrypt 100 128-bit messages.

For generating the primes needed in the *KeyGen* phase we used the naive implementation[19]. A more efficient method of generating primes is presented in [149, 51].

[19] *i.e.* we randomly generated $r \xleftarrow{\$} [2^{\lambda-k}, 2^{\lambda-k+1})$ until the $2^k r + 1$ was prime.

We further list our results in Table 3.4 (running times in seconds). When analyzing Table 3.4, note that in the case $\gamma = 1$ we obtain the Goldwasser-Micali scheme ($k = 1$) and the Joye-Libert scheme ($k = 2, 4, 8$). We stress that we considered $\lambda = 1536$[20].

For completeness, in Table 3.5 we also present the ciphertext size (in kilobytes $= 10^3$ bytes) for the previously mentioned parameters.

TABLE 3.4: Average running times for a 128-bit message

Algorithm	$\gamma = 1$				
	$k = 1$	$k = 2$	$k = 4$	$k = 8$	$k = 16$
KeyGen	0.441997	0.475780	0.461101	0.440292	0.424711
Encrypt	0.006875	0.004460	0.003152	0.002389	0.001868
Decrypt	0.670574	0.669387	0.672676	0.669685	0.665928

Algorithm	$\gamma = 2$			
	$k = 1$	$k = 2$	$k = 4$	$k = 8$
KeyGen	0.670943	0.684832	0.688289	0.719769
Encrypt	0.006601	0.005058	0.003982	0.003295
Decrypt	0.666815	0.665174	0.664918	0.664416

Algorithm	$\gamma = 4$			$\gamma = 8$	
	$k = 1$	$k = 2$	$k = 4$	$k = 1$	$k = 2$
KeyGen	1.020130	1.122080	1.119700	1.958500	1.925660
Encrypt	0.008205	0.008011	0.006905	0.012401	0.015711
Decrypt	0.666383	0.666766	0.660244	0.660967	0.659406

TABLE 3.5: Ciphertext size for a 128-bit message

	$k = 1$	$k = 2$	$k = 4$	$k = 8$	$k = 16$
$\gamma = 1$	49.152	24.576	12.288	6.1440	3.0720
$\gamma = 2$	36.864	18.432	9.2160	4.6080	–
$\gamma = 4$	30.720	15.360	7.6800	–	–
$\gamma = 8$	27.648	13.824	–	–	–

3.5.3.5 Future Work

An attractive research direction for the future is the construction of *lossy trapdoor functions* (based on the inherited homomorphic properties of our proposed cryptosystem). Another appealing future work idea is to propose a threshold variant of our scheme and to discuss security and efficiency matters.

[20] According to NIST this choice of λ offers a security strength of 128 bits.

3.6 Biometric Authentication

In biometric authentication protocols, when a user identifies himself using his biometric characteristics (captured by a sensor), the collected data will vary. Thus, traditional cryptographic approaches (such as storing a hash value) are not suitable in this case, since they are not error tolerant. As a result, biometric-based protocols must be constructed in a special way and, moreover, the system must protect the sensitivity and privacy of a user's biometric characteristics. Such a protocol is proposed in [61]. Its core is the Goldwasser-Micali encryption scheme. Thus, a natural extension of the protocol in [61] can be obtained using our generalization of the Joye-Libert scheme. Thus, we describe such a biometric authentication protocol and discuss its security.

3.6.1 Preliminaries

We further consider the security model for biometric authentication described in [40] in accordance with the terminology established in [61]. We stress that the authors of [61] preferred a rather informal way of presenting their security model while the approach of [40] is formal.

Participants and Roles. The data flow between the different roles assumed in the authentication protocol of [40] is depicted in Figure 3.8.

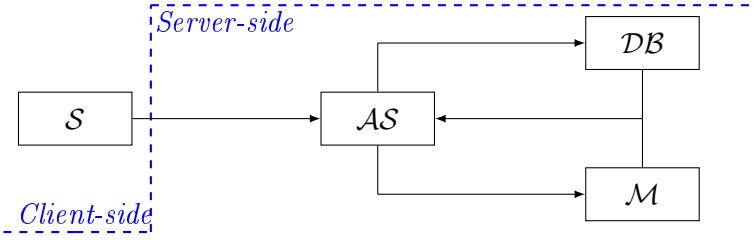


FIGURE 3.8: Data flow and roles.

The *server-side* functionality consists of three components to ensure that no single entity can associate a user's identity with the biometric data being collected during authentication. The roles assumed in the authentication protocol are:

- The *Sensor* ($\mathcal{S}$) represents the *client-side* component. As in [61], we assume that the sensor is capable of capturing the user's biometric data, extracting it into a binary string[21], and performing cryptographic operations such as PKE. We also assume a *liveness link* between the sensor and the server-side components, to provide

[21] We further consider the binary string as a vector of fixed length blocks.

confidence that the biometric data received on the *server-side* is from a present living person.

- The *Authentication Server* ($\mathcal{AS}$) is responsible for communicating with the user who wants to authenticate and organizing the entire *server-side* procedure. In a successful authentication the AS obviously learns the user's identity, meaning that it should learn nothing about the biometric data being submitted.

- The *Database* ($\mathcal{DB}$) securely stores the users' profile and its job is to execute the pre-decision part of classification. Since the $\mathcal{DB}$ is aware of privileged biometric data, it should learn nothing about the user's identity, or even be able to correlate or trace authentication runs from a given (unknown) user.

- The *Matcher* ($\mathcal{M}$) completes the authentication process by taking the output produced by the $\mathcal{DB}$ server and computing the final decision step. This implies that the $\mathcal{M}$ possesses privileged information that allows it to make a final decision, and again that it should not be able to learn anything about the user's real identity, or even be able to correlate or trace authentication runs from a given (unknown) user.

Definition 3.19. Let $v = \{v_i\}_{i\in[0,s)}$ and $w = \{w_i\}_{i\in[0,s)}$ be two s-dimensional vectors. Then the taxicab distance is defined as $\mathcal{T}(v,w) = \sum_{i=0}^{s-1} |v_i - w_i|$. The taxicab norm is defined as $\mathcal{T}(v,0)$.

The first step in having a useful authentication protocol is for it to be sound. This requirement is formalized in Requirement 1. Requirements 2. and 3. are concerned with the sensitive[22] relation between a user's identity and its biometric characteristics. We want to guarantee that the only entity in the infrastructure that knows information about this relation is the sensor.

Requirement 1. The matcher $\mathcal{M}$ can compute the taxicab distance $\mathcal{T}(b_i, b'_i)$, where b_i is the reference biometric template and b'_i is the fresh biometric template sent in the authentication request. Therefore, $\mathcal{M}$ can compare the distance to a given threshold value d and the server $\mathcal{AS}$ can make the right decision.

Requirement 2. For any identity ID_{i_0}, two biometric templates b'_{i_0}, b'_{i_1}, where $i_0, i_1 \geqslant 1$ and b'_{i_0} is the biometric template related to ID_{i_0}, it is infeasible for any of $\mathcal{M}$, $\mathcal{DB}$ and $\mathcal{AS}$ to distinguish between (ID_{i_0}, b'_{i_0}) and (ID_{i_0}, b'_{i_1}).

[22] in terms of the system's security

Requirement 3. For any two users U_{i_0} and U_{i_1}, where $i_0, i_1 \geqslant 1$, if U_{i_β}, where $\beta \xleftarrow{\$} \{0,1\}$ makes an authentication attempt, then the database $\mathcal{DB}$ can only guess β with a negligible advantage. Suppose the database $\mathcal{DB}$ makes a guess β', the advantage is $|Pr[b = b'] - 1/2|$.

3.6.2 A Biometric Authentication Protocol

In [61], the authors propose a biometric authentication protocol based on the Goldwasser-Micali scheme. A security flaw[23] of the protocol was indicated and fixed in [40]. A natural extension of Bringer *et al.*'s protocol can be obtained using the scheme proposed in Section 3.5.3.1. Thus, we describe our protocol in Section 3.6.2.1 and analyze its security in Section 3.6.2.4. A performance analysis is provided in Section 3.6.2.5.

3.6.2.1 Description

3.6.2.2 Enrollment Phase

In the protocol we consider U_i's biometric template b_i as being a γM-dimensional vector $b_i = \{b_{i,j}\}_{j\in[0,M)}$, where $b_{i,j} = \{b_{i,j,\ell}\}_{\ell\in[0,\gamma)}$ and $b_{i,j,\ell} \in [0, 2^k)$.

In the enrollment phase, U_i registers (b_i, i) at the database $\mathcal{DB}$ and (ID_i, i) at the authentication server $\mathcal{AS}$, where ID_i is U_i's pseudonym and i is the index of record b_i in $\mathcal{DB}$. Let N denote the number of records in $\mathcal{DB}$. Note that the matcher $\mathcal{M}$ possesses a key pair (sk, pk) for the scheme presented in Section 3.5.3.1.

We further denote by $\mathcal{E}(pk, \cdot)$ and $\mathcal{E}_{JL}(pk, y_\ell, \cdot)$ the encryption algorithms for the scheme presented in Section 3.5.3.1 with $pk = (n, y, k)$ and the Joye-Libert scheme[24] with $pk = (n, y_\ell, k)$, where $\ell \in [0, \gamma)$.

3.6.2.3 Verification Phase

If a user U_i wishes to authenticate himself to $\mathcal{AS}$, the next procedure is followed:

1. $\mathcal{S}$ captures the user's biometric data b'_i and sends to $\mathcal{AS}$ the user's identity ID_i together with $\mathcal{E}(pk, b'_i) = \{\mathcal{E}(pk, b'_{i,j})\}_{j\in[0,M)}$. Note that a *liveness link* is available between $\mathcal{S}$ and $\mathcal{AS}$ to ensure that data is coming from the sensor are indeed fresh and not artificial.

[23] The running time is exponential in the number of users

[24] Note that in this case we consider n to be a product of $\gamma + 1$ primes.

2. $\mathcal{AS}$ retrieves the index i using ID_i and then sends $\mathcal{E}_{JL}(pk, y_\ell, t_j)$ to the database, for $\ell \in [0, \gamma)$ and $j \in [0, N)$, where $t_j = 1$ if $j = i$, $t_j = 0$ otherwise.

3. For every $s \in [0, M)$, $\mathcal{DB}$ computes

$$\mathcal{E}(pk, b_{i,s}) = \prod_{j=0}^{N-1} \prod_{\ell=0}^{\gamma-1} \mathcal{E}_{JL}(pk, y_\ell, t_j)^{b_{j,s,\ell}} \bmod n.$$

   To prevent $\mathcal{AS}$ from performing an exhaustive search of the profile space, $\mathcal{DB}$ re-randomizes the encryptions by calculating $\mathcal{E}(pk, b_{i,s}) = x_s^{2^k} \mathcal{E}(pk, b_{i,s})$, where $x_s \xleftarrow{\$} \mathbb{Z}_n^*$. Then, $\mathcal{DB}$ sends $\mathcal{E}(pk, b_{i,s})$, for $s \in [0, M)$ to the authentication server.

4. $\mathcal{AS}$ computes v_s, $s \in [0, M)$, where

$$v_s = \mathcal{E}(pk, b'_{i,s}) / \mathcal{E}(pk, b_{i,s}) \bmod n = \mathcal{E}(pk, b'_{i,s} - b_{i,s}), \tag{3.12}$$

   and $b'_{i,s} - b_{i,s} = \{b'_{i,s,\ell} - b_{i,s,\ell}\}_{\ell \in [0,\gamma)}$. Then, $\mathcal{AS}$ makes a random permutation among v_s, for $s \in [0, M)$, and sends the permuted vector w_s, for $s \in [0, M)$, to $\mathcal{M}$. Note that Item 4 will return a valid result with high probability, thus we do not explicitly require $\mathcal{E}(pk, b_{i,s})$ to be invertible.

5. $\mathcal{M}$ decrypts w_s to check that the taxicab norm of the corresponding plaintext vector

$$\sum_{s=0}^{M-1} \sum_{\ell=0}^{\gamma-1} |w_{s,\ell}|$$

   is equal to or less than d and sends the result $\mathcal{AS}$.

6. $\mathcal{AS}$ accepts or rejects the authentication request accordingly.

Correctness (Requirement 1). We need to show that $v_s = \mathcal{E}(pk, b'_{i,s} - b_{i,s})$, for $s \in [0, M)$. First observe that

$$\begin{aligned}
\mathcal{E}(pk, b_{i,s}) &= \prod_{j=0}^{N-1} \prod_{\ell=0}^{\gamma-1} \mathcal{E}_{JL}(pk, y_\ell, t_j)^{b_{j,s,\ell}} \\
&\equiv \prod_{j=0}^{N-1} \prod_{\ell=0}^{\gamma-1} (r_{j,\gamma}^{2^k} y_\ell^{t_j})^{b_{j,s,\ell}} \\
&\equiv r_i^{2^k} \prod_{\ell=0}^{\gamma-1} y_\ell^{b_{i,s,\ell}} \bmod n.
\end{aligned}$$

Thus,

$$\mathcal{E}(pk, b'_{i,s})/\mathcal{E}(pk, b_{i,s}) \equiv \mathcal{E}(pk, b'_{i,s} - b_{i,s}) \bmod n.$$

It is obvious that the taxicab distance between b_i and b'_i

$$\sum_{s=0}^{M-1} \sum_{\ell=0}^{\gamma-1} |b'_{i,s,\ell} - b_{i,s,\ell}|$$

is equal to the taxicab norm of the plaintext vector corresponding to $\{v_s\}_{s\in[0,M)}$ and $\{w_s\}_{s\in[0,M)}$.

3.6.2.4 Security Analysis

The proofs of Theorems 3.11 and 3.12 are similar to the security proofs from [61] and, thus, are omitted. The only changes we have to make in the proofs of Theorems 3.11 and 3.12 is replacing Goldwasser-Micali with our scheme and, respectively, the Joye-Libert scheme.

Theorem 3.11 (Requirement 2)**.** For any identity ID_{i_0} and two biometric templates b'_{i_0}, b'_{i_1}, where $i_0, i_1 \geqslant 1$ and b'_{i_0} is the biometric template related to ID_{i_0}, any $\mathcal{M}$, $\mathcal{DB}$ and $\mathcal{AS}$ can distinguish between (ID_{i_0}, b'_{i_0}) and (ID_{i_0}, b'_{i_1}) with negligible advantage.

Theorem 3.12 (Requirement 3)**.** For any two users U_{i_0} and U_{i_1}, where $i_0, i_1 \geqslant 1$, if U_{i_β}, where $\beta \xleftarrow{\$} \{0,1\}$ makes an authentication attempt, then the database $\mathcal{DB}$ can only guess β with a negligible advantage.

3.6.2.5 Performance Analysis

It is easy to see that the sensor $\mathcal{S}$ and the matcher $\mathcal{M}$ perform only M encryptions and, respectively, decryptions. Comparing our proposed protocol's complexity with Bringer *et al.*'s, reduces to comparing the scheme from Section 3.5.3.1 with the Goldwasser-Micali cryptosystem.[25] On the authentication server's side, we perform γN Joye-Libert encryptions (which can be precomputed) and M divisions. Bringer *et al.*'s protocol, performs step 2 using the Goldwasser-Micali scheme and, thus, in step 4 they can use multiplications instead of divisions [26]. Since we took into consideration the fix from [40] when proposing our protocol, we have to perform M extra multiplications compared

[25] See Sections 3.5.3.3 and 3.5.3.4

[26] In $\mathbb{Z}_2$ addition and subtraction are equivalent.

to the scheme in [61]. Since we have to assemble our scheme's ciphertexts from Joye-Libert's ciphertexts we have a blowout of γ multiplications on the database's side. Thus, we perform $\gamma MN/2$ multiplications on average .

Chapter 4

Identity Based Cryptography

Identity-based cryptography was proposed in 1984 by Adi Shamir [222] who formulated its basic principles and provided an identity-based signature scheme. In 2000, Sakai, Ohgishi and Kasahara [216] have proposed an identity-based key agreement scheme, and one year later, Cocks [77] and Boneh and Franklin [59] have proposed the first identity-based encryption schemes. Cocks' scheme is based on quadratic residues, while Boneh and Franklin's scheme is based on bilinear maps. Since then, some other IBE schemes based on quadratic residues have been proposed [60, 147, 31, 76, 99, 100, 148], although some of them are not secure (see [246] for details).

Cocks'scheme encrypts messages bit by bit and each encrypted bit is a pair of two integers. The decryption consists of computing the Jacobi symbol of one of the two integers in each pair. Although Cocks' IBE scheme is efficient only for small messages, it is very elegant and *per se* revolutionary. The scheme attracted the interest of many researchers [60, 31, 76, 148]. A careful analysis of [77, 60, 31, 76, 148] shows that integers of the form $a+r$, where a is an integer and r is a quadratic residue (modulo a given integer n), play an important role in these papers. Particularly, it turns out to be important to know the distribution of quadratic residues among all integers of the form $a + r$. A study in this direction was initiated by Perron [206] for the case of a prime modulus p. However, most applications of quadratic residues to cryptography require the use of a composite modulus $n = pq$. We are thus faced with the need to extend Perron's results to composite moduli. The same was advocated in [31] (see Section 2.3 in [31]). Here, the authors avoided the extension of Perron's results to composite moduli with the price of weaker indistinguishability results (this will be fully discussed in Section 4.3.1).

The contributions presented in this chapter are structured into two parts. The first part (Section 4.2) considers sets of the form $a + X = \{(a + x) \bmod n \mid x \in X\}$, where n is a prime or the product of two primes $n = pq$, and X is a subset of $\mathbb{Z}_n^*$ whose elements

have some given Jacobi symbols modulo prime factors of n. For instance, X may be the set of all integers in $\mathbb{Z}_n^*$ whose Jacobi symbol modulo p is 1 and Jacobi symbol modulo q is -1 (assuming $n = pq$); we say that the *Jacobi pattern* of the integers in X, in this case, is "$+-$". Then, given a set $a + X$, we look for the distribution of the quadratic residues, quadratic non-residues, etc., in $a + X$. We develop complete results for all the Jacobi patterns of length one, $+$ and - (this corresponds to quadratic residues and non-residues modulo a prime) and Jacobi patterns of length two, $++$, $--$, $+-$, and $-+$ (this corresponds to moduli that are product of two distinct primes).

The results presented in Section 4.2 are a major extension of Perron's findings [206], where only the distribution of quadratic residues in the set $a + QR_p$, where p is a prime, has been considered. Related studies to the one conducted in Section 4.3 were performed in [86, 87, 204, 151], where the problem is to calculate the probability that

$$J_p(a)J_p(a+1)\cdots J_p(a+\ell-1)$$

meets some Jacobi residuosity modulo p, a priori given, for the ℓ elements, when a is chosen uniformly at random from $a \in \mathbb{Z}_p^*$ (p is a prime). Thus, in [204] it was shown that the number of integers a with the property above is in between $p/2^\ell - \epsilon$ and $p/2^\ell + \epsilon$, where $\epsilon = \ell(3 + \sqrt{p})$. Dividing these two bounds by p we obtain the probability that an integer a induces a given Jacobi residuosity for the ℓ consecutive elements. A direct extension of this result to the case of RSA moduli may lead to "much larger bound" than ϵ. In [151], an extension to RSA moduli has been proposed by generalizing [87]. Thus, it was shown that the number of integers a with the property above is $n/2^\ell + \mathcal{O}(\sqrt{n}\cdot\log^2 n)$, where n is an RSA modulus and $1 \leqslant \ell \leqslant (1/2 - \delta)\log_2 n$, for some $0 < \delta < 1/2$.

The results developed in this chapter are different than those mentioned above for at least two main reasons. First of all, we have developed exact and not approximate formulas for the number of integers with a given Jacobi pattern in sets $a + X$. Secondly, the increment factor is arbitrary in all our studies, while it is one in all the results mentioned above.

The second part of the chapter's contribution (Section 4.3) points out some applications of the results developed in the first part (Section 4.2). There are two main applications discussed here. The first one relates to Galbraith's test for Cocks' IBE scheme. This test was briefly described in several papers such us [58, 31, 148], except that some claims were not rigorously formulated and/or proved. Based on the results developed in Section 4.2, we were able to make a deep analysis of some distributions related to Cocks' IBE scheme and Galbraith's test, providing thus rigorous proofs for Galbraith's test.

The second application discussed in Section 4.3 relates to the computational indistinguishability of some distributions in [31, 76, 148], under the quadratic residuosity assumption. Based on the results developed in Section 4.2, we were able to prove statistical indistinguishability of those distributions (without any assumption).

In addition to the applications already mentioned in Section 4.3, we believe that our study in Section 4.2 is important also because it contributes to a better understanding of the structure of $\mathbb{Z}_n^*$ with respect to Jacobi patterns of length at most two, which are frequently employed in cryptography.

4.1 Preliminaries

Conventions. Positive integers $n = pq$ that are product of two distinct primes p and q will be usually called RSA integers or RSA moduli. As a convention, we assume $p < q$ for all RSA moduli $n = pq$. For simplicity, the Jacoby symbol of an integer a modulo n is denoted $J_n(a)$.

Definition 4.1 (Identity Based Encryption - IBE)**.** A *Identity Based Encryption* scheme consists of four PPT algorithms: *ParamGen*, *KeyGen*, *Encrypt* and *Decrypt*. The first one takes as input a security parameter and outputs the system's public parameters together with a master key. The *KeyGen* algorithm takes as input an identity ID together with the public parameters and the master key and outputs a private key associated to ID. The *Encrypt* algorithm, starting with a message m, an identity ID, and the public parameters, encrypts m into some ciphertext c (the encryption key is ID or some binary string derived from ID). The last algorithm decrypts c into m by using the private key associated to ID.

Remark 4.1. A standard scenario on using IBE is as follows. Whenever Alice wants to send a message m to Bob, she encrypts m by using Bob's identity $ID(B)$. In order to decrypt the message received from Alice, Bob asks the key generator *KeyGen* to deliver him the private key associated to $ID(B)$ (if he does not already have it).

Security Model. Compared to PKE adversaries, an IBE attacker can corrupt several users and obtain their private keys before attacking a certain user. Hence, an IBE model has to take this into account. Such a model was introduced by [59].

Definition 4.2 (Anonymity and Indistinguishability under Selective Identity and Chosen Plaintext Attacks - ANON-IND-ID-CPA)**.** The ANON-IND-ID-CPA security of an IBE scheme $\mathcal{S}$ is formulated by means of the following game between a challenger C and a PPT adversary A:

KeyGen(λ): The challenger C generates the public parameters and sends them to adversary A, while keeping the master key to himself.

Queries: The adversary issues a finite number of adaptive queries. A query can be one of the following types:

- Private key query. When A requests a query for an identity, the challenger runs the *Extract* algorithm and returns the resulting private key to A.
- Encryption query. Adversary A can issue only one query of this type. He sends C two pairs (ID_0, m_0) and (ID_1, m_1) consisting of two equal length plaintexts m_0 and m_1 and two identities ID_0 and ID_1. The challenger flips a coin $b \in \{0, 1\}$ and encrypts m_b using ID_b. The resulting ciphertext c is sent to the adversary. The following restrictions are in place: private key queries for ID_0 and ID_1 must never be issued.

Guess: In this phase, the adversary outputs a guess $b' \in \{0, 1\}$. He wins the game, if $b' = b$.

The advantage of an adversary A attacking an IBE scheme is defined as

$$\text{IBEAdv}_{\mathcal{S}}(A) = |Pr[b = b'] - 1/2|$$

where the probability is computed over the random bits used by C and A. An IBE scheme is ANON-IND-ID-CPA secure, if for any PPT adversary A the advantage $\text{IBEAdv}_{\mathcal{S}}(A)$ is negligible. If we consider $ID_0 = ID_1$ in the above game, we obtain the concept of IND-ID-CPA security.

The Cocks IBE scheme. The Cocks encryption scheme [77] was the first IBE scheme based on quadratic residues. It can be proven that Cocks' scheme is secure in ROM under the QR assumption [77, 121, 148]. We further describe the generalized version of the scheme [148].

ParamGen(λ): Randomly generate two large prime numbers p, q and compute $n = pq$. Generate uniformly at random $e \in J_n^+ \backslash QR_n$ and output the public parameters $pp = (n, e, h)$, where h is a cryptographic hash function that maps identities into J_n^+. The master key is the factorization (p, q) of n.

KeyGen(p, q, ID): Let $a = h(ID)$. Set $a' = a$, if $a \in QR_n$, and $a' = ea$, otherwise. Uniformly at random choose a square root r of a' and output it as the private key.

Encryption(m, ID): Let $a = h(ID)$. To encrypt a bit $m \in \{-1, 1\}$, choose uniformly at random $t_1, t_2 \in \mathbb{Z}_n^*$ such that $J_n(t_1) = J_n(t_2) = m$. Compute $c_1 = t_1 + at_1^{-1} \bmod n$ and $c_2 = t_2 + eat_2^{-1} \bmod n$ and output the ciphertext (c_1, c_2).

Decryption(c_1, c_2, r): Set $c = c_1$ if $r^2 \equiv a \bmod n$, or $c = c_2$, otherwise. Then, output $m = J_n(c + 2r)$.

Remark 4.2. For a given $m \in \{-1, 1\}$, the generation of an integer $t \in \mathbb{Z}_n^*$ with $J_n(t) = m$ can be done by repetition because the probability of success for a random choice of t is $1/2$. This is due to the fact that $|J_n^+| = |J_n^-|$ [224].

4.2 The set $a + \mathbb{Z}_n^*$

In [206], Perron studied the set $\{(a + r)_p \mid r \in QR_p\}$, where $p > 2$ is a prime and $a \in \mathbb{Z}_p^*$, in order to establish how many of its elements are still quadratic residues modulo p. In this section we extend Perron's study to sets

$$a + X = \{(a + x)_n \mid x \in X\}$$

where n is a prime or an RSA modulus, $a \in \mathbb{Z}_n^*$, and $X \subseteq \mathbb{Z}_n^*$. When n is a prime, X will be $\mathbb{Z}_n^*$, QR_n, and QNR_n; when n is an RSA modulus, X will be $\mathbb{Z}_n^*$, J_n^+, J_n^-, QR_n, QNR_n, $J_n^+ \backslash QR_n$, $J_n^{\pm}$, and $J_n^{\mp}$.

Given a set $A = a + X$ as above, we will partition $A^* = A \cap \mathbb{Z}_n^*$ into two subsets

- $QR_n(A) = A^* \cap QR_n$ and
- $QNR_n(A) = A^* \cap QNR_n$,

if n is a prime, and into four subsets

- $QR_n(A) = A^* \cap QR_n$,
- $(J_n^+ \backslash QR_n)(A) = A^* \cap (J_n^+ \backslash QR_n)$,
- $J_n^{\pm}(A) = A^* \cap J_n^{\pm}$, and
- $J_n^{\mp}(A) = A^* \cap J_n^{\mp}$,

if n is an RSA modulus. Moreover, in this last case, we will also consider $J_n^+(A) = A^* \cap J_n^+$ and $J_n^-(A) = A^* \cap J_n^-$. The diagram in Figure 4.1 provides a pictorial view of this case.

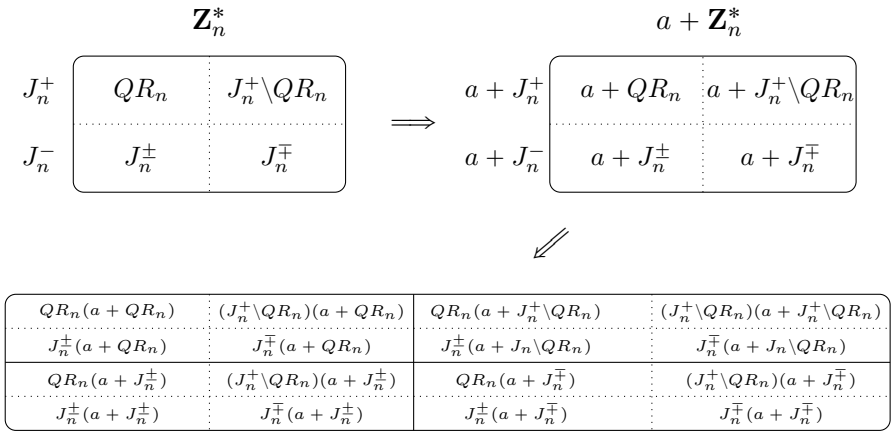


FIGURE 4.1: Partitioning the set $a + \mathbb{Z}_n^*$ when n is an RSA modulus.

Now, our goal is to estimate the cardinalities of these subsets of $\mathbb{Z}_n^*$ and then to compute probability distributions on them, such as $P(x \in QR_n : x \leftarrow a + J_n^\mp)$ (this is the probability that x is a quadratic residue when it is uniformly at random sampled from $a + J_n^\mp$).

Perron's study developed in [206] corresponds, although not exactly in the form we use in our paper, to the case of the set $QR_n(a + QR_n)$ with a prime n.

4.2.1 Prime moduli

We will focus in this sub-section on the calculation of the cardinalities of the sets defined above, when $n > 2$ is a prime. Recall that, in this case, $QR_n = J_n^+$, $QNR_n = J_n^-$, and $\mathbb{Z}_n^*$ is the disjoint union of the sets QR_n and QNR_n.

Proposition 4.1. Let $p > 2$ be a prime and $a \in \mathbb{Z}_p^*$. Then,

1. $a + \mathbb{Z}_p = \mathbb{Z}_p$ and $|(a + \mathbb{Z}_p)^*| = |\mathbb{Z}_p^*| = p - 1$;
2. $a + \mathbb{Z}_p^* = \mathbb{Z}_p\backslash\{a\}$ and $|(a + \mathbb{Z}_p^*)^*| = |\mathbb{Z}_p^*\backslash\{a\}| = p - 2$.

Proof. Both (1) and (2) are straightforward from definitions. However, we will provide some details for the first part of (2).

Given $x \in \mathbb{Z}_p^*$, the integer $(a+x)_p$ is different from a. Therefore, it is in $\mathbb{Z}_p\backslash\{a\}$. Moreover, for any $y \in \mathbb{Z}_p\backslash\{a\}$ there exists $x \in \mathbb{Z}_p^*$ such that $(a + x)_p = y$. □

Proposition 4.2. Let $p > 2$ be a prime and $a \in \mathbb{Z}_p^*$. Then,

$$|(a + QR_p)^*| = \frac{p - 2 - J_p(-a)}{2}$$

and

$$|(a + QNR_p)^*| = \frac{p - 2 + J_p(-a)}{2}$$

Proof. Let $\alpha \in QR_p$. Clearly, $(a + \alpha)_p$ is co-prime to p iff $\alpha \neq (-a)_p$. Therefore, if $(-a)_p \in QNR_p$ then $\alpha \neq (-a)_p$ because $\alpha \in QR_p$ and, as a conclusion, all integers in $a + QR_p$ are co-prime to p.

If $(-a)_p \in QR_p$, exactly one integer in $a+QR_p$, namely $(a+(-a)_p)_p = 0$, is not co-prime to p.

If we add to these remarks the fact that $|QR_p| = (p-1)/2$, we obtain the first part of the proposition.

The second part of this proposition follows a similar proof line to its first part. Alternatively, it can be obtained from the set partitioning

$$(a + \mathbb{Z}_p^*)^* = (a + QR_p)^* \cup (a + QNR_p)^*,$$

Proposition 4.1, and the formula for $|(a + QR_p)^*|$. □

Corollary 4.1. Let $p > 2$ be a prime and $a, b \in \mathbb{Z}_p^*$.

1. If a and b are of the same quadratic residuosity, then
   (a) $|(a + QR_p)^*| = |(b + QR_p)^*|$ and
   (b) $|(a + QNR_p)^*| = |(b + QNR_p)^*|$;
2. If a and b are of opposite quadratic residuosities, then

$$|(a + QR_p)^*| = |(b + QNR_p)^*|.$$

Proof. All the equalities simply follow from Proposition 4.2 and from the fact that $(-a)_p$ and $(-b)_p$ are of the same quadratic residuosity in the first case, and are of opposite quadratic residuosities in the second case. □

We go further to estimate $|QR_p(A)|$ and $|QNR_p(A)|$ for the aforementioned values of A. We begin with the case of $A = a + \mathbb{Z}_p^*$, which simply follows from Proposition 4.1 and from the fact that $|QR_p| = |QNR_p| = (p-1)/2$ [194, 224].

Corollary 4.2. Let $p > 2$ be a prime and $a \in \mathbb{Z}_p^*$. Then,

$$|QR_p(a + \mathbb{Z}_p^*)| = \frac{p - 2 - J_p(a)}{2}$$

and

$$|QNR_p(a + \mathbb{Z}_p^*)| = \frac{p - 2 + J_p(a)}{2}$$

Proof. By Proposition 4.1, Item 2, $QR_p(a + \mathbb{Z}_p^*) = QR_p(\mathbb{Z}_p^* \backslash \{a\})$. If $a \in QR_p$, then $|QR_p(\mathbb{Z}_p^* \backslash \{a\})| = |QR_p(\mathbb{Z}_p^*)| - 1$; otherwise, $|QR_p(\mathbb{Z}_p^* \backslash \{a\})| = |QR_p(\mathbb{Z}_p^*)|$. By taking into account that $|QR_p| = (p-1)/2$, we obtain the first part of the proposition.

The second part of the proposition follows a similar proof line to its first part. Alternatively, one may partition $(a + \mathbb{Z}_p^*)^*$ into $QR_p(a + \mathbb{Z}_p^*)$ and $QNR_p(a + \mathbb{Z}_p^*)$, and then use Proposition 4.1 and the first part of this corollary. □

In [206], Perron proposed a very useful characterization of the quadratic residues in the set $a + QR_p$. However, he considered the integer 0 as a quadratic residue, which is not the case in our paper. For this reason and for the sake of uniformity and completeness of the paper we recall and adapt Perron's results to fit our case.

Lemma 4.1. Let $p > 2$ be a prime, $a \in \mathbb{Z}_p^*$, and $r \in QR_p$. Then, $(a + r)_p \in QR_p$ if and only if there exists $u \in \mathbb{Z}_p^* \backslash SQRT_p(a, -a)$ such that $r \equiv_p \frac{1}{4}\left(u - \frac{a}{u}\right)^2$.

Proof. Let $p > 2$ be a prime, $a \in \mathbb{Z}_p^*$, and $r \in QR_p$.

Assume first that $r \equiv_p \frac{1}{4}\left(u - \frac{a}{u}\right)^2$ for some $u \in \mathbb{Z}_p^* \backslash SQRT_p(a, -a)$. We remark that $r \not\equiv_p 0$ since u is not a square root of a. Then, the following congruences hold:

$$\begin{aligned} a + r &\equiv_p a + \tfrac{1}{4}\left(u - \tfrac{a}{u}\right)^2 \\ &\equiv_p a + \tfrac{1}{4}\left(u^2 - 2a + \tfrac{a^2}{u^2}\right) \\ &\equiv_p \tfrac{1}{4}\left(u^2 + 2a + \tfrac{a^2}{u^2}\right) \\ &\equiv_p \tfrac{1}{4}\left(u + \tfrac{a}{u}\right)^2. \end{aligned}$$

As u is not a square root of $-a$ modulo p, we deduce that $a + r \not\equiv_p 0$ and, therefore, $(a + r)_p \in QR_p$.

Conversely, assume that $(a+r)_p \in QR_p$. Therefore, there exists $t \in \mathbb{Z}_p^*$ such that $a + r \equiv_p t^2$. As $r \in QR_p$, there exists $s \in \mathbb{Z}_p^*$ such that $r \equiv_p s^2$. Combining the two congruences we obtain

$$(s-t)(s+t) \equiv_p -a.$$

As $a \in \mathbb{Z}_p^*$ it follows that $(-a)_p \in \mathbb{Z}_p^*$ and, therefore, $(s+t)$ cannot be divisible by p. So, we may write

$$s - t \equiv_p -\frac{a}{s+t}$$

which leads to

$$s \equiv_p \frac{1}{2}\left((s+t) - \frac{a}{s+t}\right).$$

Now, we take $u = (s+t)_p$. It follows that $u \in \mathbb{Z}_p^*$ and

$$r \equiv_p s^2 \equiv_p \frac{1}{4}\left(u - \frac{a}{u}\right)^2.$$

It remains to prove that $u \notin SQRT_p(a, -a)$.

Because $r \in QR_p$, it follows that $r \not\equiv_p 0$, which leads to the fact that u cannot be a square root of a modulo p. Similarly, because $(a+r)_p \in QR_p$ it follows that $a + r \not\equiv_p 0$. As

$$a + r \equiv_p \frac{1}{4}\left(u + \frac{a}{u}\right)^2,$$

we deduce that u cannot be a square root of $-a$ modulo p. □

Remark 4.3. One may reformulate Lemma 4.1 as follows:

> Let $p > 2$ be a prime, $a \in \mathbb{Z}_p^*$, and $r \in QR_p$. Then, $(a+r)_p \in QR_p$ if and only if $r \equiv_p \frac{(s-a)^2}{4s}$, for some $s \in QR_p \backslash \{a, -a\}$.

This reformulation shows that $(a+r)_p$ is a quadratic residue modulo p if and only if the quadratic residue r can be written as an expression that depends of another quadratic residue modulo p.

Lemma 4.2. Let $p > 2$ be a prime and $a \in \mathbb{Z}_p^*$. Then, the function $\psi_a : \mathbb{Z}_p^* \backslash SQRT_p(a, -a) \rightarrow QR_p$ given by

$$\psi_a(u) = \frac{1}{4}\left(u - \frac{a}{u}\right)^2 \bmod p,$$

for all $u \in \mathbb{Z}_p^* \backslash SQRT_p(a, -a)$, is a four-to-one map. Moreover, $(a + \psi_a(u))_p \in QR_p(a + QR_p)$, for all $u \in \mathbb{Z}_p^* \backslash SQRT_p(a, -a)$.

Proof. Let $a \in \mathbb{Z}_p^*$ and $u \in \mathbb{Z}_p^* \backslash SQRT_p(a, -a)$. The form of $\psi_a(u)$ together with the fact that u is not a square root of a modulo p show that $\psi_a(u) \in QR_p$. Therefore, the function ψ_a is well-defined.

The congruence

$$\left(x - \frac{a}{x}\right)^2 \equiv_p \left(u - \frac{a}{u}\right)^2$$

has four solutions in $\mathbb{Z}_p^*$ in the nondeterminate x, namely u, $(-u)_p$, $(a/u)_p$, and $(-a/u)_p$. If we prove that the four solutions above are pairwise incongruent modulo p, then ψ_a is a four-to-one map.

Due to the fact that $p > 2$ and $(u, p) = 1$, we obtain $u \not\equiv_p -u$. In a similar way and taking into consideration that $(a, p) = 1$, we obtain $a/u \not\equiv_p -a/u$. Finally, the hypothesis $u \notin SQRT_p(a, -a)$ leads to the fact that neither u nor $-u$ can be congruent to a/u or $-a/u$ modulo p. Therefore, the four integers u, $(-u)_p$, $(a/u)_p$, and $(-a/u)_p$ are pairwise incongruent modulo p.

A simple computation (see also the proof of Lemma 4.1) shows that

$$a + \psi_a(u) \equiv_p \frac{1}{4}\left(u + \frac{a}{u}\right)^2.$$

Combining this with the fact that $u \notin SQRT_p(-a)$, we obtain $(a + \psi_a(u))_p \in QR_p(a + QR_p)$, for all $u \in \mathbb{Z}_p^* \backslash SQRT_p(a, -a)$. □

The two lemmata proved above lead directly to the following very important result.

Theorem 4.1. Let $p > 2$ be a prime and $a \in \mathbb{Z}_p^*$. Then,

$$|QR_p(a + QR_p)| = \frac{|\mathbb{Z}_p^* \backslash SQRT_p(a, -a)|}{4}.$$

We have now all the necessary elements to calculate the cardinals of the sets $Y(a + X)$, with $X, Y \in \{QR_p, QNR_p\}$. For the sake of simplicity we introduce the following notation.

Notation 4.1. Let $p > 2$ be a prime and $a \in \mathbb{Z}$ such that p does not divide a. We denote by $\tau_{p,a}^1$, $\bar{\tau}_{p,a}^1$, $\tau_{p,a}^3$, and $\bar{\tau}_{p,a}^3$ the following symbols:

$$\tau_{p,a}^i = \begin{cases} 1, & \text{if } (p)_4 = i \text{ and } (a)_p \in QR_p \\ 0, & \text{otherwise} \end{cases}$$

and

$$\bar{\tau}^i_{p,a} = \begin{cases} 1, & \text{if } (p)_4 = i \text{ and } (a)_p \in QNR_p \\ 0, & \text{otherwise,} \end{cases}$$

where $i = 1, 3$.

These symbols have useful properties such as:

1. $\tau^1_{p,a} = \tau^1_{p,-a}$, $\bar{\tau}^1_{p,a} = \bar{\tau}^1_{p,-a}$, and $\tau^3_{p,a} = \bar{\tau}^3_{p,-a}$ (when $(p)_4 = 1$, $(a)_p$ is a quadratic residue modulo p if and only if $(-a)_p$ is a quadratic residue modulo p);
2. Exactly one of these symbols is one, the others being zero (there are exactly two possibilities for $(p)_4$ and exactly two possibilities for $(a)_p$ with respect to its quadratic residuosity; therefore, there are exactly four combinations and exactly one of them holds for a given p and a);
3. The product of two or more symbols, all of them for the same integers p and a, is zero (this follows immediately from the second property).

Corollary 4.3. Let $p > 2$ be a prime, $k = p \text{ div } 4$, and $a \in \mathbb{Z}^*_p$. Then,

$$|QR_p(a + QR_p)| = k - \tau^1_{p,a}.$$

Proof. According to Theorem 4.1, everything comes down to the computation of $|\mathbb{Z}^*_p \backslash SQRT_p(a, -a)|$. Four cases are in order.

Case 1: $p = 4k + 1$ for some integer k, and $a \in QR_p$. Then $|SQRT_p(a, -a)| = 4$ because $(-a)_p \in QR_p$. As a result, $|\mathbb{Z}^*_p \backslash SQRT_p(a, -a)| = 4k - 4$.

Case 2: $p = 4k + 1$ for some integer k, and $a \in QNR_p$. Then $|SQRT_p(a, -a)| = 0$ because $(-a)_p \in QNR_p$. As a result, $|\mathbb{Z}^*_p \backslash SQRT_p(a, -a)| = 4k$.

Case 3: $p = 4k + 3$ for some integer k, and $a \in QR_p$. Then $|SQRT_p(a, -a)| = 2$ because $(-a)_p \in QNR_p$. As a result, $|\mathbb{Z}^*_p \backslash SQRT_p(a, -a)| = 4k$.

Case 4: $p = 4k + 3$ for some integer k, and $a \in QNR_p$. Then $|SQRT_p(a, -a)| = 2$ because $(-a)_p \in QR_p$. As a result, $|\mathbb{Z}^*_p \backslash SQRT_p(a, -a)| = 4k$.

All these cases lead to the statement in the corollary. □

Corollary 4.4. Let $p > 2$ be a prime, $k = p \text{ div } 4$, and $a \in \mathbb{Z}^*_p$. Then,

$$|QNR_p(a + QR_p)| = k + \tau^3_{p,a}.$$

Proof. The set $(a + QR_p)^*$ is partitioned into quadratic residues and quadratic non-residues modulo p. Therefore,

$$|QNR_p(a + QR_p)| = |(a + QR_p)^*| - |QR_p(a + QR_p)|.$$

We will accomplish this computation on cases.

Case 1: $p = 4k+1$ *for some integer* k*, and* $a \in QR_p$. Then, $(-a)_p \in QR_p$ and, therefore, $|(a+QR_p)^*| = 2k-1$ (by Proposition 4.2) and $|QR_p(a+QR_p)| = k-1$ (by Corollary 4.3). As a result, $|QNR_p(a + QR_p)| = k$.

Case 2: $p = 4k+1$ *for some integer* k*, and* $a \in QNR_p$. Then, $(-a)_p \in QNR_p$ and, therefore, $|(a+QR_p)^*| = 2k$ (by Proposition 4.2) and $|QR_p(a+QR_p)| = k$ (by Corollary 4.3). As a result, $|QNR_p(a + QR_p)| = k$.

Case 3: $p = 4k + 3$ *for some integer* k*, and* $a \in QR_p$. Then, $(-a)_p \in QNR_p$ and, therefore, $|(a + QR_p)^*| = 2k + 1$ (by Proposition 4.2) and $|QR_p(a + QR_p)| = k$ (by Corollary 4.3). As a result, $|QNR_p(a + QR_p)| = k + 1$.

Case 4: $p = 4k+3$ *for some integer* k*, and* $a \in QNR_p$. Then, $(-a)_p \in QR_p$ and, therefore, $|(a + QR_p)^*| = 2k$ (by Proposition 4.2) and $|QR_p(a + QR_p)| = k$ (by Corollary 4.3). As a result, $|QNR_p(a + QR_p)| = k$. □

Corollary 4.5. Let $p > 2$ be a prime, $k = p \text{ div } 4$, and $a \in \mathbb{Z}_p^*$. Then,

$$|QR_p(a + QNR_p)| = k + \bar{\tau}^3_{p,a}$$

and

$$|QNR_p(a + QNR_p)| = k - \bar{\tau}^1_{p,a}.$$

Proof. The set $\mathbb{Z}_p^*$ is partitioned into QR_p and QNR_p. Therefore, $a + \mathbb{Z}_p^*$ is a disjoint set union

$$a + \mathbb{Z}_p^* = (a + QR_p) \cup (a + QNR_p)$$

This leads to the set partitions

$$QR_p(a + \mathbb{Z}_p^*) = QR_p(a + QR_p) \cup QR_p(a + QNR_p)$$

and

$$QNR_p(a + \mathbb{Z}_p^*) = QNR_p(a + QR_p) \cup QNR_p(a + QNR_p).$$

As a conclusion,

$$|QR_p(a + QNR_p)| = |QR_p(a + \mathbb{Z}_p^*)| - |QR_p(a + QR_p)|$$

and

$$|QNR_p(a + QNR_p)| = |QNR_p(a + \mathbb{Z}_p^*)| - |QNR_p(a + QR_p)|$$

Now, the corollary follows from Corollaries 4.2 to 4.4. □

Similar to Corollary 4.1 one can prove the following result.

Corollary 4.6. Let $p > 2$ be a prime and $a, b \in \mathbb{Z}_p^*$.

1. If a and b are of the same quadratic residuosity, then
   (a) $|QR_p(a + QR_p)| = |QR_p(b + QR_p)|$,
   (b) $|QNR_p(a + QR_p)| = |QNR_p(b + QR_p)|$,
   (c) $|QR_p(a + QNR_p)| = |QR_p(b + QNR_p)|$, and
   (d) $|QNR_p(a + QNR_p)| = |QNR_p(b + QNR_p)|$;
2. If a and b are of opposite quadratic residuosities, then
   (a) $|QR_p(a + QNR_p)| = |QNR_p(b + QR_p)|$ and
   (b) $|QNR_p(a + QNR_p)| = |QR_p(b + QR_p)|$.

We close this sub-section with a result which establishes an interesting bijection between $(a + QR_p)^*$ and $(b + QNR_p)^*$. This bijection can be used to obtain alternative proofs for some of the results already obtained in this sub-section.

Lemma 4.3. Let $p > 2$ be a prime and $a, b \in \mathbb{Z}_p^*$ of opposite quadratic residuosities. Then, there exists a bijective map

$$f : (a + QR_p)^* \to (b + QNR_p)^*$$

such that f maps $QR_p(a + QR_p)$ onto $QNR_p(b + QNR_p)$ and $QNR_p(a + QR_p)$ onto $QR_p(b+QNR_p)$. Moreover, such a bijection can be found in $\mathcal{O}((\log p)^2)$ time complexity.

Proof. Corollary 4.1 shows that $|(a + QR_p)^*| = |(b + QNR_p)^*|$ and, therefore, there exists a bijection from $(a + QR_p)^*$ to $(b + QNR_p)^*$. Such a bijection can be easily

found if we compute the unique solution $e \in \mathbb{Z}_p^*$ to the linear congruence $ax \equiv_p b$ in the non-determinate x. Once we have found the solution e, we consider the function

$$f : (a + QR_p)^* \to (b + QNR_p)^*$$

given by $f(x) = (e \cdot x)_p$, for any $x \in (a + QR_p)^*$.

By taking into account that e must be a quadratic non-residue (a and b are of opposite quadratic residuosities) and $QNR_p = \{(e \cdot \alpha)_p \mid \alpha \in QR_p\}$, we easily obtain that f is well-defined and bijective.

Let us show now that f maps $QR_p(a + QR_p)$ onto $QNR_p(b + QNR_p)$. Indeed,

- Given $x = (a+\alpha)_p \in QR_p(a+QR_p)$, where $\alpha \in QR_p$, we have that $(e \cdot x)_p \in QNR_p$ and
$$(e \cdot x)_p = ((e \cdot a)_p + (e \cdot \alpha)_p)_p = (b + (e \cdot \alpha)_p)_p \in b + QNR_p.$$
This shows that $f(x)$ is a quadratic non-residue in $b + QNR_p$;
- For any quadratic non-residue $y = (b + \beta)_p \in b + QNR_p$, the integer $x = (a + \alpha)_p$, where $e \cdot \alpha = \beta$, satisfies $f(x) = y$; moreover, $y = (e \cdot x)_p$ which shows that x is a quadratic residue modulo p (because both y and e are quadratic non-residues modulo p).

As a conclusion, f maps $QR_p(a + QR_p)$ onto $QNR_p(b + QNR_p)$.

In a similar way it is shown that f maps $QNR_p(a + QR_p)$ onto $QR_p(b + QNR_p)$. Moreover, to obtain f we only need to compute e, and this can be done in $\mathcal{O}((\log p)^2)$ time complexity [224]. □

4.2.2 Composite moduli

We are now extending the results in the previous sub-section to the case of RSA moduli $n = pq$, where p and q are distinct odd primes[1]. First of all, we recall a well-known result, tailored for RSA moduli, that can be found in almost any standard book on number theory, such as [194].

Theorem 4.2 ([194]). Let $n = pq$ be an RSA modulus. Then, the function $f : \mathbb{Z}_n \to \mathbb{Z}_p \times \mathbb{Z}_q$ given by

$$f(x) = ((x)_p, (x)_q),$$

[1] One may notice that the results developed in this section can easily be extended to moduli that are product of more than two distinct odd primes.

for any $x \in Z_n$, is bijective and maps $\mathbb{Z}_n^*$ onto $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$.

One of the main applications of the bijection f in Theorem 4.2 is to show that Euler's totient function ϕ is multiplicative. The bijection f has other applications as well, and some of them will be discussed by us below.

Theorem 4.3. Let $n = pq$ be an RSA modulus and $a \in \mathbb{Z}_n^*$. Then, the bijection f in Theorem 4.2 has the following properties:

1. f maps $(a + \mathbb{Z}_n^*)^*$ onto $((a)_p + \mathbb{Z}_p^*)^* \times ((a)_q + \mathbb{Z}_q^*)^*$;
2. f maps $(a + QR_n)^*$ onto $((a)_p + QR_p)^* \times ((a)_q + QR_q)^*$;
3. f maps $(a + J_n^+ \backslash QR_n)^*$ onto $((a)_p + QNR_p)^* \times ((a)_q + QNR_q)^*$;
4. f maps $(a + J_n^{\pm})^*$ onto $((a)_p + QR_p)^* \times ((a)_q + QNR_q)^*$;
5. f maps $(a + J_n^{\mp})^*$ onto $((a)_p + QNR_p)^* \times ((a)_q + QR_q)^*$.

Proof. We will only prove (3) (the other properties follow a similar proof line).

Let $x \in (a + J_n^+ \backslash QR_n)^*$. Then, $x \in \mathbb{Z}_n^*$ and $x = (a + \alpha)_n$, for some $\alpha \in J_n^+ \backslash QR_n$. Therefore, $(x)_p \in \mathbb{Z}_p^*$, $(x)_q \in \mathbb{Z}_q^*$, $(x)_p = ((a)_p + (\alpha)_p)_p$, and $(x)_q = ((a)_q + (\alpha)_q)_q$. As

$$1 = J_n(\alpha) = J_p((\alpha)_p) \cdot J_q((\alpha)_q)$$

and $\alpha \notin QR_n$, it follows that

$$((\alpha)_p, (\alpha)_q) \in QNR_p \times QNR_q.$$

Therefore,

$$f(x) \in ((a)_p + QNR_p)^* \times ((a)_q + QNR_q)^*.$$

To show that f is onto, we consider $(y, z) \in ((a)_p + QNR_p)^* \times ((a)_q + QNR_q)^*$. Therefore, $y \in \mathbb{Z}_p^*$, $z \in \mathbb{Z}_q^*$, $y = ((a)_p + \beta)_p$, and $z = ((a)_q + \gamma)_q$, for some $\beta \in QNR_p$ and $\gamma \in QNR_q$. Starting with β and γ, CRT gives rise to a unique $\alpha \in \mathbb{Z}_n^*$ such that $(\alpha)_p = \beta$ and $(\alpha)_q = \gamma$. Then,

$$J_n(\alpha) = J_p(\alpha) \cdot J_q(\alpha) = J_p(\beta) \cdot J_q(\gamma) = (-1)(-1) = 1,$$

which shows that $\alpha \in J_n^+$. Moreover, $\alpha \notin QR_n$ because $(\alpha)_p \notin QR_p$ and $(\alpha)_q \notin QR_q$.

Consider now $x = (a + \alpha)_n$. Clearly, $x \in a + J_n^+ \backslash QR_n$. Moreover, $x \in (a + J_n^+ \backslash QR_n)^*$ because $(x)_p = y \in \mathbb{Z}_p^*$ and $(x)_q = z \in \mathbb{Z}_q^*$. Therefore, f is onto. □

Several consequences of Theorem 4.3 are in order.

Corollary 4.7. Let $n = pq$ be an RSA modulus and $a \in \mathbb{Z}_n^*$. Then,

1. $|(a + \mathbb{Z}_n^*)^*| = (p-2)(q-2)$.
2. $|(a + QR_n)^*| = \frac{(p-2-J_p(-a))(q-2-J_q(-a))}{4}$.
3. $|(a + J_n^+ \backslash QR_n)^*| = \frac{(p-2+J_p(-a))(q-2+J_q(-a))}{4}$.
4. $|(a + J_n^\pm)^*| = \frac{(p-2-J_p(-a))(q-2+J_q(-a))}{4}$.
5. $|(a + J_n^\mp)^*| = \frac{(p-2+J_p(-a))(q-2-J_q(-a))}{4}$.
6. $|(a + J_n^+)^*| = \frac{(p-2)(q-2)+J_p(-a)J_q(-a)}{2}$.
7. $|(a + J_n^-)^*| = \frac{(p-2)(q-2)-J_p(-a)J_q(-a)}{2}$.
8. $|(a + QNR_n)^*| = \frac{3(p-2)(q-2)+J_p(-a)(q-2)+J_q(-a)(p-2)-J_p(-a)J_q(-a)}{4}$.

Proof. (1) follows from Theorem 4.3(1) and Proposition 4.1, (2) from Theorem 4.3(2) and Proposition 4.2, (3) from Theorem 4.3(3) and Proposition 4.2, (4) from Theorem 4.3(4) and Proposition 4.2, and (5) from Theorem 4.3(5) and Proposition 4.2.

(6) is based on the disjoint set union

$$(a + J_n^+)^* = (a + QR_n)^* \cup (a + J_n^+ \backslash QR_n)^*$$

together with (2) and (3), while (7) is based on

$$(a + J_n^-)^* = (a + J_n^\pm)^* \cup (a + J_n^\mp)^*,$$

(4), and (5). The last property follows, for instance, from (7) and (3). □

We present now other properties of the function f in Theorem 4.2, necessary to partition the set $(a + \mathbb{Z}_n^*)^*$ in the same way $\mathbb{Z}_n^*$ is partitioned by Jacobi symbols.

Theorem 4.4. Let $n = pq$ be an RSA modulus and $a \in \mathbb{Z}_n^*$. Then, the bijection f in Theorem 4.2 has the following properties:

1. f maps $QR_n(a + \mathbb{Z}_n^*)$ onto $QR_p((a)_p + \mathbb{Z}_p^*) \times QR_q((a)_q + \mathbb{Z}_q^*)$;
2. f maps $(J_n^+ \backslash QR_n)(a + \mathbb{Z}_n^*)$ onto $QNR_p((a)_p + \mathbb{Z}_p^*) \times QNR_q((a)_q + \mathbb{Z}_q^*)$;
3. f maps $J_n^\pm(a + \mathbb{Z}_n^*)$ onto $QR_p((a)_p + \mathbb{Z}_p^*) \times QNR_q((a)_q + \mathbb{Z}_q^*)$;

4. f maps $J_n^{\mp}(a+\mathbb{Z}_n^*)$ onto $QNR_p((a)_p+\mathbb{Z}_p^*)\times QR_q((a)_q+\mathbb{Z}_q^*)$.

Proof. It is similar to the proof of Theorem 4.3. □

Corollary 4.8. Let $n=pq$ be an RSA modulus and $a\in\mathbb{Z}_n^*$. Then,

1. $|QR_n(a+\mathbb{Z}_n^*)|=\frac{(p-2-J_p(a))(q-2-J_q(a))}{4}$.
2. $|(J_n^+\backslash QR_n)(a+\mathbb{Z}_n^*)|=\frac{(p-2+J_p(a))(q-2+J_q(a))}{4}$.
3. $|J_n^{\pm}(a+\mathbb{Z}_n^*)|=\frac{(p-2-J_p(a))(q-2+J_q(a))}{4}$.
4. $|J_n^{\mp}(a+\mathbb{Z}_n^*)|=\frac{(p-2+J_p(a))(q-2-J_q(a))}{4}$.
5. $|J_n^{+}(a+\mathbb{Z}_n^*)|=\frac{(p-2)(q-2)+J_p(a)J_q(a)}{2}$.
6. $|J_n^{-}(a+\mathbb{Z}_n^*)|=\frac{(p-2)(q-2)-J_p(a)J_q(a)}{2}$.
7. $|QNR_n(a+\mathbb{Z}_n^*)|=\frac{3(p-2)(q-2)+J_p(a)(q-2)+J_q(a)(p-2)-J_p(a)J_q(a)}{4}$.

Proof. For (1)-(4) we use Theorem 4.4(1)-(4), respectively, and Corollary 4.2. The properties (5)-(7) are immediate consequences of (1)-(4). □

We use now the function f in Theorem 4.2 to partition the set $a+QR_n$.

Theorem 4.5. Let $n=pq$ be an RSA modulus and $a\in\mathbb{Z}_n^*$. Then, the bijection f in Theorem 4.2 has the following properties:

1. f maps $QR_n(a+QR_n)$ onto $QR_p((a)_p+QR_p)\times QR_q((a)_q+QR_q)$;
2. f maps $(J_n^+\backslash QR_n)(a+QR_n)$ onto $QNR_p((a)_p+QR_p)\times QNR_q((a)_q+QR_q)$;
3. f maps $J_n^{\pm}(a+QR_n)$ onto $QR_p((a)_p+QR_p)\times QNR_q((a)_q+QR_q)$;
4. f maps $J_n^{\mp}(a+QR_n)$ onto $QNR_p((a)_p+QR_p)\times QR_q((a)_q+QR_q)$.

Proof. It is similar to the proof of Theorem 4.3. □

Corollary 4.9. Let $n=pq$ be an RSA modulus, $k_1=p \text{ div } 4$, $k_2=q \text{ div } 4$, and $a\in\mathbb{Z}_n^*$. Then,

1. $|QR_n(a+QR_n)|=(k_1-\tau_{p,a}^1)(k_2-\tau_{q,a}^1)$.

2. $|(J_n^+\backslash QR_n)(a+QR_n)| = (k_1+\tau_{p,a}^3)(k_2+\tau_{q,a}^3)$.
3. $|J_n^{\pm}(a+QR_n)| = (k_1-\tau_{p,a}^1)(k_2+\tau_{q,a}^3)$.
4. $|J_n^{\mp}(a+QR_n)| = (k_1+\tau_{p,a}^3)(k_2-\tau_{q,a}^1)$.
5. $|J_n^+(a+QR_n)| = 2k_1k_2 + k_1(\tau_{q,a}^3-\tau_{q,a}^1) + k_2(\tau_{p,a}^3-\tau_{p,a}^1) + \tau_{p,a}^1\tau_{q,a}^1 + \tau_{p,a}^3\tau_{q,a}^3$.
6. $|J_n^-(a+QR_n)| = 2k_1k_2 + k_1(\tau_{q,a}^3-\tau_{q,a}^1) + k_2(\tau_{p,a}^3-\tau_{p,a}^1) - \tau_{p,a}^1\tau_{q,a}^3 - \tau_{p,a}^3\tau_{q,a}^1$.
7. $|QNR_n(a+QR_n)| = 3k_1k_2 + k_1(2\tau_{q,a}^3-\tau_{q,a}^1) + k_2(2\tau_{p,a}^3-\tau_{p,a}^1) - \tau_{p,a}^1\tau_{q,a}^3 - \tau_{p,a}^3\tau_{q,a}^1 + \tau_{p,a}^3\tau_{q,a}^3$.

Proof. For (1)-(4) we use Theorem 4.5(1)-(4), respectively, and Corollaries 4.3 and 4.4. The properties (5)-(7) are immediate consequences of (1)-(4). □

The following theorem shows how to partition $a+J_n^+\backslash QR_n$.

Theorem 4.6. Let $n=pq$ be an RSA modulus and $a\in\mathbb{Z}_n^*$. Then, the bijection f in Theorem 4.2 has the following properties:

1. f maps $QR_n(a+J_n^+\backslash QR_n)$ onto $QR_p((a)_p+QNR_p)\times QR_q((a)_q+QNR_q)$;
2. f maps $(J_n^+\backslash QR_n)(a+J_n^+\backslash QR_n)$ onto $QNR_p((a)_p+QNR_p)\times QNR_q((a)_q+QNR_q)$;
3. f maps $J_n^{\pm}(a+J_n^+\backslash QR_n)$ onto $QR_p((a)_p+QNR_p)\times QNR_q((a)_q+QNR_q)$;
4. f maps $J_n^{\mp}(a+J_n\backslash QR_n)$ onto $QNR_p((a)_p+QNR_p)\times QR_q((a)_q+QNR_q)$.

Proof. It is similar to the proof of Theorem 4.3. □

Corollary 4.10. Let $n=pq$ be an RSA modulus, $k_1=p \text{ div } 4$, $k_2=q \text{ div } 4$, and $a\in\mathbb{Z}_n^*$. Then,

1. $|QR_n(a+J_n^+\backslash QR_n)| = (k_1+\bar{\tau}_{p,a}^3)(k_2+\bar{\tau}_{q,a}^3)$.
2. $|(J_n^+\backslash QR_n)(a+J_n^+\backslash QR_n)| = (k_1-\bar{\tau}_{p,a}^1)(k_2-\bar{\tau}_{q,a}^1)$.
3. $|J_n^{\pm}(a+J_n^+\backslash QR_n)| = (k_1+\bar{\tau}_{p,a}^3)(k_2-\bar{\tau}_{q,a}^1)$.
4. $|J_n^{\mp}(a+J_n^+\backslash QR_n)| = (k_1-\bar{\tau}_{p,a}^1)(k_2+\bar{\tau}_{q,a}^3)$.
5. $|J_n^+(a+J_n^+\backslash QR_n)| = 2k_1k_2 + k_1(\bar{\tau}_{q,a}^3-\bar{\tau}_{q,a}^1) + k_2(\bar{\tau}_{p,a}^3-\bar{\tau}_{p,a}^1) + \bar{\tau}_{p,a}^3\bar{\tau}_{q,a}^3 + \bar{\tau}_{p,a}^1\bar{\tau}_{q,a}^1$.
6. $|J_n^-(a+J_n^+\backslash QR_n)| = 2k_1k_2 + k_1(\bar{\tau}_{q,a}^3-\bar{\tau}_{q,a}^1) + k_2(\bar{\tau}_{p,a}^3-\bar{\tau}_{p,a}^1) - \bar{\tau}_{p,a}^3\bar{\tau}_{q,a}^1 - \bar{\tau}_{p,a}^1\bar{\tau}_{q,a}^3$.

7. $|QNR_n(a + J_n^+ \backslash QR_n)| = 3k_1k_2 + k_1(\bar{\tau}_{q,a}^3 - 2\bar{\tau}_{q,a}^1) + k_2(\bar{\tau}_{p,a}^3 - 2\bar{\tau}_{p,a}^1) - \bar{\tau}_{p,a}^3\bar{\tau}_{q,a}^1 - \bar{\tau}_{p,a}^1\bar{\tau}_{q,a}^3 + \bar{\tau}_{p,a}^1\bar{\tau}_{q,a}^1$.

Proof. For (1)-(4) we use Theorem 4.6(1)-(4), respectively, and Corollary 4.5. The properties (5)-(7) are immediate consequences of (1)-(4). □

For the partitioning of the set $a + J_n^{\pm}$ we have the following result.

Theorem 4.7. Let $n = pq$ be an RSA modulus and $a \in \mathbb{Z}_n^*$. Then, the bijection f in Theorem 4.2 has the following properties:

1. f maps $QR_n(a + J_n^{\pm})$ onto $QR_p((a)_p + QR_p) \times QR_q((a)_q + QNR_q)$;
2. f maps $(J_n^+ \backslash QR_n)(a + J_n^{\pm})$ onto $QNR_p((a)_p + QR_p) \times QNR_q((a)_q + QNR_q)$;
3. f maps $J_n^{\pm}(a + J_n^{\pm})$ onto $QR_p((a)_p + QR_p) \times QNR_q((a)_q + QNR_q)$;
4. f maps $J_n^{\mp}(a + J_n^{\pm})$ onto $QNR_p((a)_p + QR_p) \times QR_q((a)_q + QNR_q)$.

Proof. It is similar to the proof of Theorem 4.3. □

Corollary 4.11. Let $n = pq$ be an RSA modulus, $k_1 = p \text{ div } 4$, $k_2 = q \text{ div } 4$, and $a \in \mathbb{Z}_n^*$. Then,

1. $|QR_n(a + J_n^{\pm})| = (k_1 - \tau_{p,a}^1)(k_2 + \bar{\tau}_{q,a}^3)$.
2. $|(J_n^+ \backslash QR_n)(a + J_n^{\pm})| = (k_1 + \tau_{p,a}^3)(k_2 - \bar{\tau}_{q,a}^1)$.
3. $|J_n^{\pm}(a + J_n^{\pm})| = (k_1 - \tau_{p,a}^1)(k_2 - \bar{\tau}_{q,a}^1)$.
4. $|J_n^{\mp}(a + J_n^{\pm})| = (k_1 + \tau_{p,a}^3)(k_2 + \bar{\tau}_{q,a}^3)$.
5. $|J_n^+(a + J_n^{\pm})| = 2k_1k_2 + k_1(\bar{\tau}_{q,a}^3 - \bar{\tau}_{q,a}^1) + k_2(\tau_{p,a}^3 - \tau_{p,a}^1) - \tau_{p,a}^1\bar{\tau}_{q,a}^3 - \tau_{p,a}^3\bar{\tau}_{q,a}^1$.
6. $|J_n^-(a + J_n^{\pm})| = 2k_1k_2 + k_1(\bar{\tau}_{q,a}^3 - \bar{\tau}_{q,a}^1) + k_2(\tau_{p,a}^3 - \tau_{p,a}^1) + \tau_{p,a}^1\bar{\tau}_{q,a}^1 + \tau_{p,a}^3\bar{\tau}_{q,a}^3$.
7. $|QNR_n(a + J_n^{\pm})| = 3k_1k_2 + k_1(\bar{\tau}_{q,a}^3 - 2\bar{\tau}_{q,a}^1) + k_2(2\tau_{p,a}^3 - \tau_{p,a}^1) + \tau_{p,a}^1\bar{\tau}_{q,a}^1 + \tau_{p,a}^3\bar{\tau}_{q,a}^3 - \tau_{p,a}^3\bar{\tau}_{q,a}^1$.

Proof. For (1)-(4) we use Theorem 4.7(1)-(4), respectively, and Corollaries 4.3 to 4.5. The properties (5)-(7) are immediate consequences of (1)-(4). □

The last application of Theorem 4.1 we discuss in this subsection is with respect to the set $a + J_n^{\mp}$.

Theorem 4.8. Let $n = pq$ be an RSA modulus and $a \in \mathbb{Z}_n^*$. Then, the bijection f in Theorem 4.2 has the following properties:

1. f maps $QR_n(a + J_n^{\mp})$ onto $QR_p((a)_p + QNR_p) \times QR_q((a)_q + QR_q)$;
2. f maps $(J_n^{+} \backslash QR_n)(a + J_n^{\mp})$ onto $QNR_p((a)_p + QNR_p) \times QNR_q((a)_q + QR_q)$;
3. f maps $J_n^{\pm}(a + J_n^{\mp})$ onto $QR_p((a)_p + QNR_p) \times QNR_q((a)_q + QR_q)$;
4. f maps $J_n^{\mp}(a + J_n^{\mp})$ onto $QNR_p((a)_p + QNR_p) \times QR_q((a)_q + QR_q)$.

Proof. It is similar to the proof of Theorem 4.3. □

Corollary 4.12. Let $n = pq$ be an RSA modulus, $k_1 = p \text{ div } 4$, $k_2 = q \text{ div } 4$, and $a \in \mathbb{Z}_n^*$. Then,

1. $|QR_n(a + J_n^{\mp})| = (k_1 + \bar{\tau}_{p,a}^3)(k_2 - \tau_{q,a}^1)$.
2. $|(J_n^{+} \backslash QR_n)(a + J_n^{\mp})| = (k_1 - \bar{\tau}_{p,a}^1)(k_2 + \tau_{q,a}^3)$.
3. $|J_n^{\pm}(a + J_n^{\mp})| = (k_1 + \bar{\tau}_{p,a}^3)(k_2 + \tau_{q,a}^3)$.
4. $|J_n^{\mp}(a + J_n^{\mp})| = (k_1 - \bar{\tau}_{p,a}^1)(k_2 - \tau_{q,a}^1)$.
5. $|J_n^{+}(a + J_n^{\mp})| = 2k_1k_2 + k_1(\tau_{q,a}^3 - \tau_{q,a}^1) + k_2(\bar{\tau}_{p,a}^3 - \bar{\tau}_{p,a}^1) - \bar{\tau}_{p,a}^3\tau_{q,a}^1 - \bar{\tau}_{p,a}^1\tau_{q,a}^3$.
6. $|J_n^{-}(a + J_n^{\mp})| = 2k_1k_2 + k_1(\tau_{q,a}^3 - \tau_{q,a}^1) + k_2(\bar{\tau}_{p,a}^3 - \bar{\tau}_{p,a}^1) + \bar{\tau}_{p,a}^3\tau_{q,a}^3 + \bar{\tau}_{p,a}^1\tau_{q,a}^1$.
7. $|QNR_n(a + J_n^{\mp})| = 3k_1k_2 + k_1(2\tau_{q,a}^3 - \tau_{q,a}^1) + k_2(\bar{\tau}_{p,a}^3 - 2\bar{\tau}_{p,a}^1) + \bar{\tau}_{p,a}^3\tau_{q,a}^3 + \bar{\tau}_{p,a}^1\tau_{q,a}^1 - \bar{\tau}_{p,a}^1\tau_{q,a}^3$.

Proof. For (1)-(4) we use Theorem 4.8(1)-(4), respectively, and Corollaries 4.3 to 4.5. The properties (5)-(7) are immediate consequences of (1)-(4). □

We have thus provided formulas for all cardinalities of the sets in Figure 4.1.

4.2.3 Probability distributions on $a + \mathbb{Z}_n^*$

The results developed in the previous sub-sections allow us to calculate various probability distributions on subsets $(a + X)^*$, where X is $\mathbb{Z}_n^*$, QR_n, $J_n^+ \backslash QR_n$, $J_n^{\pm}$, $J_n^{\mp}$, J_n^+, J_n^-, or QNR_n. We will give below a few examples. The notation

$$P(x \in Y \,:\, x \leftarrow (a + X)^*)$$

stands for the probability that x is in Y when it is uniformly at random sampled from $(a + X)^*$, where Y is a subset of $(a + X)^*$ as those in the previous sub-sections. Using our notation, this probability can be calculated by

$$P(x \in Y \,:\, x \leftarrow (a + X)^*) = \frac{|Y(a + X)|}{|(a + X)^*|} = \frac{|(a + X)^* \cap Y|}{|(a + X)^*|}.$$

We will provide below just a few examples of calculating such probabilities.

Corollary 4.13. Let $n = pq$ be an RSA modulus and $a \in \mathbb{Z}_n^*$. Then, the following hold:

1. $P\left(x \in QR_n \,:\, x \leftarrow (a + QR_n)^*\right) = \begin{cases} \frac{1}{4}, & \text{if } a \in J_n^+ \backslash QR_n, \\ \frac{1}{4} - \mathcal{O}\left(\frac{1}{\sqrt{n}}\right), & \text{otherwise.} \end{cases}$

2. $P\left(x \in J_n^+ \backslash QR_n \,:\, x \leftarrow (a + QR_n)^*\right) = \begin{cases} \frac{1}{4}, & \text{if } a \in J_n^+ \backslash QR_n, \\ \frac{1}{4} + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right), & \text{otherwise.} \end{cases}$

3. $P\left(x \in J_n^{\pm} \,:\, x \leftarrow (a + QR_n)^*\right) = \begin{cases} \frac{1}{4}, & \text{if } a \in J_n^+ \backslash QR_n, \\ \frac{1}{4} - \mathcal{O}\left(\frac{1}{\sqrt{n}}\right), & \text{otherwise.} \end{cases}$

4. $P\left(x \in J_n^{\mp} \,:\, x \leftarrow (a + QR_n)^*\right) = \begin{cases} \frac{1}{4}, & \text{if } a \in J_n^+ \backslash QR_n, \\ \frac{1}{4} + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right), & \text{otherwise.} \end{cases}$

5. $P\left(x \in J_n^+ \,:\, x \leftarrow (a + QR_n)^*\right) = \begin{cases} \frac{1}{2}, & \text{if } a \in QNR_n, \\ \frac{1}{2} + \mathcal{O}\left(\frac{1}{n}\right), & \text{otherwise.} \end{cases}$

6. $P\left(x \in J_n^- \,:\, x \leftarrow (a + QR_n)^*\right) = \begin{cases} \frac{1}{2}, & \text{if } a \in QNR_n, \\ \frac{1}{2} - \mathcal{O}\left(\frac{1}{n}\right), & \text{otherwise.} \end{cases}$

7. $P\left(x \in QNR_n \,:\, x \leftarrow (a + QR_n)^*\right) = \begin{cases} \frac{3}{4}, & \text{if } a \in J_n^+ \backslash QR_n, \\ \frac{3}{4} + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right), & \text{otherwise.} \end{cases}$

Proof. We will only prove (1) as an example; the other properties follow a similar proof line.

First, recall that by Corollaries 4.7 and 4.9 we have

$$\begin{aligned} P\left(x \in QR_n \,:\, x \leftarrow (a+QR_n)^*\right) &= \frac{|QR_n(a+QR_n)|}{|(a+QR_n)^*|} \\ &= \frac{4(k_1-\tau^1_{p,a})(k_2-\tau^1_{q,a})}{(p-2-J_p(-a))(q-2-J_q(-a))}, \end{aligned}$$

where $k_1 = p \operatorname{div} 4$ and $k_2 = q \operatorname{div} 4$. What we have now to do is to consider several cases with respect to p, q, and a.

Case 1: $a \in J_n^+ \backslash QR_n$. Then, regardless of $(p)_4$ and $(q)_4$ we have $p-2-J_p(-a) = 4k_1$, $q-2-J_q(-a) = 4k_2$, and $\tau^1_{p,a} = 0 = \tau^1_{q,a}$. Therefore,

$$P\left(x \in QR_n \,:\, x \leftarrow (a+QR_n)^*\right) = \frac{1}{4}.$$

Case 2: $a \in J_n^{\mp}$. Then, regardless of $(p)_4$ we have $p-2-J_p(-a) = 4k_1$ and $\tau^1_{p,a} = 0$. A simple computation on the two possible values of $(q)_4$ leads to

$$P\left(x \in QR_n \,:\, x \leftarrow (a+QR_n)^*\right) = \begin{cases} \frac{1}{4}\left(1-\frac{1}{2k_2-1}\right), & \text{if } (q)_4 = 1 \\ \frac{1}{4}\left(1-\frac{1}{2k_2+1}\right), & \text{if } (q)_4 = 3 \end{cases}$$

Case 3: $a \in J_n^{\pm}$. This case is similar to the previous one (simply switch k_2 with k_1 and q with p).

Case 4: $a \in QR_n$. Then, we obtain

$$P\left(x \in QR_n \,:\, x \leftarrow (a+QR_n)^*\right) = \begin{cases} \frac{1}{4}\left(1-\frac{2k_1+2k_2-3}{(2k_1-1)(2k_2-1)}\right), & \text{if } (p)_4 = 1 = (q)_4 \\ \frac{1}{4}\left(1-\frac{2k_1+2k_2-1}{(2k_1-1)(2k_2+1)}\right), & \text{if } (p)_4 = 1 \text{ and } (q)_4 = 3 \\ \frac{1}{4}\left(1-\frac{2k_1+2k_2-1}{(2k_1+1)(2k_2-1)}\right), & \text{if } (p)_4 = 3 \text{ and } (q)_4 = 1 \\ \frac{1}{4}\left(1-\frac{2k_1+2k_2+1}{(2k_1+1)(2k_2+1)}\right), & \text{if } (p)_4 = 3 = (q)_4 \end{cases}$$

From these cases one can easily infer the result in (1). □

Remark 4.4. The probability in Corollary 4.13(1) is paired with that in Corollary 4.13(2). The pairing means that when

$$P\left(x \in QR_n \,:\, x \leftarrow (a + QR_n)^*\right) = \frac{1}{4} - \frac{c}{\sqrt{n}}$$

then

$$P\left(x \in J_n \backslash QR_n \,:\, x \leftarrow (a + QR_n)^*\right) = \frac{1}{4} + \frac{c}{\sqrt{n}},$$

for some constant $c > 0$ (to see it, one has to do a similar calculation for Corollary 4.13, Item 2 as we did for Corollary 4.13Item 1). The same happens with the probabilities in Corollary 4.13, Items 3 and 4, Corollary 4.13,Items 5 and 6, and Corollary 4.13, Items 1 and 7.

4.3 Applications

We believe that the results developed in the previous section have important applications to quadratic residuosity-based cryptography. We will illustrate some of these applications in the next subsections.

4.3.1 Cocks' IBE Scheme and Galbraith's Test

According to [58], Galbraith developed a test to show that Cocks' IBE scheme is not anonymous in the following sense. Given two random public keys (identities) $a, b \in J_n^+$, one may distinguish with overwhelming probability whether a ciphertext c is encrypted under the public key a or under the public key b. Galbraith's test (abbreviated GT), was briefly described in [58, 31], but some claims were not rigorously proved. Using the results developed in the previous sections, we can complete GT description in [58, 31] by rigorous arguments. First of all, let us introduce the following sets of integers:

$$\begin{aligned} C_n(a) &= \{(t + at^{-1})_n \mid t \in \mathbb{Z}_n^*\} \\ C_n^*(a) &= C_n(a) \cap \mathbb{Z}_n^* \\ G_n(a) &= \{c \in \mathbb{Z}_n^* \mid J_n(c^2 - 4a) = 1\} \end{aligned}$$

where $n > 2$ and $a \in \mathbb{Z}_n^*$.

When n is an RSA modulus and $a \in J_n^+$, the set $C_n(a)$ corresponds to the set of encryptions in Cocks' IBE scheme, and $G_n(a)$ corresponds to the set of all integers in $\mathbb{Z}_n^*$ that "pass" GT [58, 31].

We develop now some counting results, similar to the ones in Section 2, for $C_n(a)$, $C_n^*(a)$, and $G_n(a)$. We begin with the case of prime moduli.

Theorem 4.9. Let $p > 2$ be a prime and $a, c \in \mathbb{Z}_p^*$. Then,

1. $0 \in C_p(a)$ if and only if $-a \in QR_p$;
2. $c \in C_p(a)$ if and only if $c^2 - 4a \equiv_p 0$ or $(c^2 - 4a)_p \in QR_p$.

Proof. For (1) we have:

$$\begin{aligned} 0 \in C_p(a) &\Leftrightarrow t + at^{-1} \equiv_p 0, \text{ for some } t \in \mathbb{Z}_p^* \\ &\Leftrightarrow a + t^2 \equiv_p 0, \text{ for some } t \in \mathbb{Z}_p^* \\ &\Leftrightarrow -a \equiv_p t^2, \text{ for some } t \in \mathbb{Z}_p^* \\ &\Leftrightarrow -a \in QR_p. \end{aligned}$$

In order to prove (2) remark that $c \in C_p(a)$ if and only if the quadratic congruence

$$t^2 - ct + a \equiv_p 0 \tag{4.2}$$

has solutions in $\mathbb{Z}_p^*$. Moreover, Section 4.3.1 has integer solutions if and only if its discriminant $\Delta = (c^2 - 4a)_p$ is 0 or a quadratic residue modulo p. It remains to prove that, in any of these two cases, Section 4.3.1 has solutions in $\mathbb{Z}_p^*$.

If $\Delta = 0$, then $a \equiv_p (c/2)^2$. The Section 4.3.1 has exactly one solution in $\mathbb{Z}_p$, namely $t = (c/2)_p$. Moreover, $t \in \mathbb{Z}_p^*$ because $c \in \mathbb{Z}_p^*$.

If $\Delta \in QR_p$, then the Section 4.3.1 has two solutions in $\mathbb{Z}_p$, namely $((c + r)/2)_p$ and $((c - r)/2)_p$, where r and $(-r)_p$ are the two square roots of Δ in $\mathbb{Z}_p$. If we assume now that $((c + r)/2) \equiv_p 0$ or $((c - r)/2) \equiv_p 0$, then $(c + r)(c - r) \equiv_p 0$ and, therefore, $c^2 - r^2 \equiv_p 0$. This leads to a contradiction because $c^2 - r^2 \equiv_p 4a$ and $a \in \mathbb{Z}_p^*$. Therefore, $((c + r)/2)_p$ and $((c - r)/2)_p$ are in $\mathbb{Z}_p^*$. □

Corollary 4.14. Let $p > 2$ be a prime and $a \in \mathbb{Z}_p^*$. Then, $C_p^*(a)$ can be written as a disjoint set union $C_p^*(a) = C_p^0(a) \cup C_p^1(a)$, where

- $C_p^0(a) = \{c \in \mathbb{Z}_p^* \mid J_p(c^2 - 4a) = 0\}$ and
- $C_p^1(a) = \{c \in \mathbb{Z}_p^* \mid J_p(c^2 - 4a) = 1\}$.

Proof. This is in fact a new way to express the statement in Theorem 4.9(2). □

Corollary 4.15. Let $p > 2$ be a prime, $k = p \text{ div } 4$, and $a \in \mathbb{Z}_p^*$. Then,

1. $|C_p^0(a)| = 2(\tau_{p,a}^1 + \tau_{p,a}^3)$;
2. $|C_p^1(a)| = 2|QR_p(a + QRp)| = 2(k - \tau_{p,a}^1)$;
3. $|C_p^*(a)| = 2(k + \tau_{p,a}^3)$;
4. $|C_p(a)| = 2(k + \tau_{p,a}^3) + \tau_{p,a}^1 + \bar{\tau}_{p,a}^3$;

Proof. We count the integers in $C_p(a)$ with the help of Theorem 4.9 as follows:

(a) If $a \in QR_p$ and r and $(-r)_p$ are the square roots modulo p of a, then $(2r)_p$ and $(-2r)_p$ are the only (incongruent modulo p) integers in $C_p^0(a)$;

(b) Each quadratic residue $u = (-4a+c^2)_p \in (-4a+QR_p)$ gives rise to two (incongruent modulo p) integers in $C_p^1(a)$, namely c and $(-c)_p$;

(c) The integers obtained as above are pairwise incongruent modulo p;

(d) $0 \in C_p(a)$ if and only if $-a \in QR_p$.

Therefore, the item (a) gives rise to $|C_p^0(a)| = 2(\tau_{p,a}^1 + \tau_{p,a}^3)$. The items (b) and (c) lead to

$$|C_p^1(a)| = 2|QR_p(-4a + QR_p)| = 2(k - \tau_{p,-a}^1) = 2(k - \tau_{p,a}^1) = 2|QR_p(a + QR_p)|$$

(we have used Corollary 4.3, the fact that a and $4a$ have the same quadratic residuosity, and $\tau_{p,a}^1 = \tau_{p,-a}^1$). From these, (3) follows immediately. To obtain (4) we count the integer 0 as well, by means of (d). □

Theorem 4.10. Let $n = pq$ be an RSA modulus and $a \in \mathbb{Z}_n^*$. Then, the bijection f in Theorem 4.2 maps the set $C_n(a)$ onto $C_p((a)_p) \times C_q((a)_q)$ and the set $C_n^*(a)$ onto $C_p^*((a)_p) \times C_q^*((a)_q)$.

Proof. We will only prove the theorem for the case of the set $C_n^*(a)$ (the other case is similar to this). Given $c = (t + at^{-1})_n \in C_n^*(a)$, $f(c) = ((c)_p, (c)_q)$. We may write $(c)_p = ((t)_p + (a)_p(t)_p^{-1})_p$ and $(c)_q = ((t)_q + (a)_q(t)_q^{-1})_q$. Then, clearly, $((c)_p, (c)_q) \in C_p^*((a)_p) \times C_q^*((a)_q)$.

Conversely, given $c_1 = (t_1 + (a)_p t_1^{-1})_p \in C_p^*((a)_p)$ and $c_2 = (t_2 + (a)_q t_2^{-1})_q \in C_q^*((a)_q)$, one may compute by means of CRT an unique $t \in \mathbb{Z}_n^*$ such that $t \equiv_p t_1$ and $t \equiv_q t_2$. Then, it is straightforward to check that $c = (t + at^{-1})_n \in C_n^*(a)$ and $f(c) = (c_1, c_2)$. □

Given an RSA modulus $n = pq$, $a \in \mathbb{Z}_n^*$, and $e_1, e_2 \in \{-1, 0, 1\}$, define

$$C_n^{e_1,e_2} = \{c \in \mathbb{Z}_n^* \mid J_p(c^2 - 4a) = e_1,\ J_q(c^2 - 4a) = e_2\}.$$

Now, we are ready to prove the following results regarding $|C_n(a)|$, $|C_n^*(a)|$, and $|G_n(a)|$.

Corollary 4.16. Let $n = pq$ be an RSA modulus and $a \in \mathbb{Z}_n^*$. Then, $C_n^*(a)$ can be written as a disjoint set union $C_n^*(a) = C_n^{0,0}(a) \cup C_n^{0,1}(a) \cup C_n^{1,0}(a) \cup C_n^{1,1}(a)$.

Proof. It follows directly from Theorem 4.10 and Corollary 4.14. □

Corollary 4.17. Let $n = pq$ be an RSA modulus, $k_1 = p \text{ div } 4$, $k_2 = q \text{ div } 4$, and $a \in \mathbb{Z}_n^*$. Then,

1. $|C_n^{0,0}(a)| = 4(\tau_{p,a}^1 + \tau_{p,a}^3)(\tau_{q,a}^1 + \tau_{q,a}^3)$;
2. $|C_n^{0,1}(a)| = 4(\tau_{p,a}^1 + \tau_{p,a}^3)(k_2 - \tau_{q,a}^1)$;
3. $|C_n^{1,0}(a)| = 4(\tau_{q,a}^1 + \tau_{q,a}^3)(k_1 - \tau_{p,a}^1)$;
4. $|C_n^{1,1}(a)| = 4|QR_n(a + QR_n)| = 4(k_1 - \tau_{p,a}^1)(k_2 - \tau_{q,a}^1)$;
5. $|C_n^*(a)| = 4(k_1 + \tau_{p,a}^3)(k_2 + \tau_{q,a}^3)$;
6. $|C_n(a)| = (2(k_1 + \tau_{p,a}^3) + \tau_{p,a}^1 + \bar{\tau}_{p,a}^3)(2(k_2 + \tau_{q,a}^3) + \tau_{q,a}^1 + \bar{\tau}_{q,a}^3)$.

Proof. It follows directly from Theorem 4.10 and Corollary 4.15. □

Theorem 4.11. Let $n = pq$ be an RSA modulus and $a \in \mathbb{Z}_n^*$. Then,

1. $G_n(a) = C_n^{1,1}(a) \cup C_n^{-1,-1}(a)$.
2. $|G_n(a)| = 4|QR_n(a + J_n^+)|$.

Proof. (1) follows from the definitions of $G_n(a)$ and $C_n^{1,1}(a)$. For (2) we remark first that

$$G_n(a) = \{c \in \mathbb{Z}_n^* \mid (c^2)_n \in 4a + J_n^+\}.$$

Then, observe that each $u \in QR_n$ has exactly four square roots in $\mathbb{Z}_n^*$. Moreover, distinct quadratic residues modulo n have distinct square roots in $\mathbb{Z}_n^*$. Then, (2) follows from $|QR_n(4a + J_n^+)| = |QR_n(a + J_n^+)|$. □

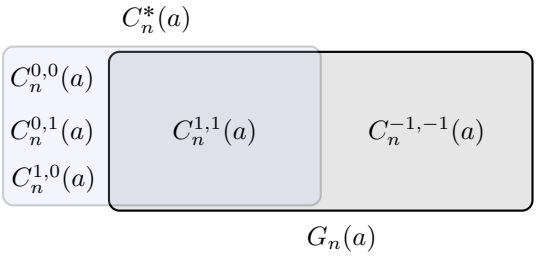


FIGURE 4.2: The sets C_n^* and $G_n(a)$.

Theorem 4.11 provides a very good image on the relationship between $C_n^*(a)$ and $G_n(a)$; this is pictorially represented in Figure 4.2. There is one more remark we would like to make. Given $c \in \mathbb{Z}_n^*$, there exists $a \in J_n^+$ such that $c \in C_n^*(a)$. Indeed, such an a can be obtained as described in Algorithm 37.

Algorithm 37. Computing a.

Input: RSA modulus $n = pq$ and $c \in \mathbb{Z}_n^*$.
Output: An integer $a \in J_n^+$ such that $c \in C_n^*(a)$.

1 **while** $c^2 - s_1 \notin QR_p$ **do**
2 $\quad s_1 \xleftarrow{\$} QR_p$
3 **end**
4 $a_1 \leftarrow ((c^2 - s_1)/4)_p$
5 **while** $c^2 - s_2 \notin QR_q$ **do**
6 $\quad s_2 \xleftarrow{\$} QR_q$
7 **end**
8 $a_2 \leftarrow ((c^2 - s_2)/4)_q$
9 Use CRT to compute a such that $a \equiv_p a_1$ and $a \equiv_q a_2$ **return** a

The probability of generating s_1 as in the first step of Algorithm 37 is negligible close to 1/2 because

$$c^2 - QR_p = \begin{cases} c^2 + QR_p, & \text{if } (p)_4 = 1 \\ c^2 + QNR_p, & \text{if } (p)_4 = 3 \end{cases}$$

and thus almost half of the integers in $c^2 - QR_p$ are quadratic residues modulo p (by Corollaries 4.3 and 4.5). Similarly, the probability of generating s_2 as in the third step of Algorithm 37 is negligible close to 1/2. The integer a computed in the fifth step of Algorithm 37 is a quadratic residue modulo n because a_1 and a_2 are quadratic residues modulo p and q, respectively. ‘ To show that $c \in C_n^*(a)$ it is sufficient to remark that $c^2 - 4a$ is a quadratic residue modulo n because, according to our construction, $c^2 - 4a \equiv_p s_1$, $c^2 - 4a \equiv_q s_2$, and both s_1 and s_2 are quadratic residues modulo p and q, respectively.

We are now in a position to present Galbraith's test. Assume that an identity $a \in J_n^+$ is given and we would like to decide whether an integer $c \in \mathbb{Z}_n^*$ was encrypted under a.

Directly from Corollary 4.17 it follows that $c \notin C_n(a)$ if $J_n(c^2 - 4a) = -1$. On the other side, if $J_n(c^2 - 4a) = 1$, then the probability that $c \in C_n^*(a)$ is

$$P(c \in C_n^*(a) \ : \ c \leftarrow G_n(a)) = \frac{|C_n^{1,1}(a)|}{|G_n(a)|} = \frac{4|QR_n(a + QR_n)|}{4|QR_n(a + J_n^+)|} = \frac{1}{2} - \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$$

(the property $|QR_n(a + J_n^+)| = |QR_n(a + QR_n)| + |QR_n(a + J_n^+ \backslash QR_n)|$ has been used, together with Corollaries 4.9 and 4.10).

Algorithm 38 presents Galbraith's test. As we have already discussed, the probability that Algorithm 38 outputs 1 is negligible close to 1/2. One may also remark that it outputs 0 even for $c \in C_n^{0,0} \cup C_n^{0,1} \cup C_n^{1,0} \subseteq C_n^*(a)$. However, the probability that Cocks' IBE scheme outputs such ciphertexts is $\mathcal{O}(1/\sqrt{n})$ (according to Corollary 4.17), which is negligible.

Algorithm 38. Galbraith's Test.

Input: RSA modulus n, $a \in J_n^+$, and $c \in \mathbb{Z}_n^*$.
Output: 1, if $c \in C_n^*(a)$ with probability negligible close to 1/2, and 0, otherwise.

```
if J_n(c^2 − 4a) = 1 then
    return 1
end
else
    return 0
end
```

When using Cocks' IBE scheme, the ciphertext consists of a sequence of encrypted bits under the same identity. Therefore, Galbraith's test applied to each encrypted bit in the sequence determines whether the ciphertext is encrypted under a given identity or not with overwhelming probability.

4.3.2 Statistical indistinguishability

We will illustrate in this subsection the utility of the results developed in our paper to prove statistical indistinguishability.

As argued in the previous subsection, Cocks' IBE scheme is not anonymous. In [31], several results have been developed in order to obtain an anonymous variant of Cocks' IBE scheme. In order to prove security of their schemes, the authors of [31] have first established a series of computational indistinguishability results, denoted Lemma 2.1, Lemma 2.2, and Lemma 2.3 (these results are also used in [76, 148]). The first indistinguishability result in [31] (Lemma 2.1) states that, given n an RSA modulus and $a \in J_n^+$,

the distribution

$$X_n = \{J_n(x) \mid x \leftarrow (a + QR_n)^*\}$$

is computationally indistinguishable from the uniform distribution U on $\{-1, 1\}$, under the QR assumption for $RSAgen$. The third result in [31] (Lemma 2.3) states that, given n an RSA modulus and $a \in J_n^+$, the distribution

$$Y_n = \{J_n(x) \mid x \leftarrow (-4a + QR_n)^*\}$$

is computationally indistinguishable from the uniform distribution U on $\{-1, 1\}$, under the QR assumption for $RSAgen$. Both proofs of Lemma 2.1 and 2.3 in [31] are directly based on the ANON-IND-ID-CPA security of Cocks' IBE scheme.

Using the results developed in Section 2 we can prove stronger results for the two distributions above.

Theorem 4.12. Let n be an RSA modulus and $a \in J_n^+$. Then, the distributions

$$X_n = \{J_n(x) \mid x \leftarrow (a + QR_n)^*\}$$

and

$$Y_n = \{J_n(x) \mid x \leftarrow (-a + QR_n)^*\}$$

are each of them statistically indistinguishable from the uniform distribution U on $\{-1, 1\}$.

Proof. We will prove the theorem only for the case of X_n (the other case follows a similar proof line). Therefore, we show that the statistical distance $\Delta(X_n, U)$ between X_n and U is negligible, where

$$\Delta(X_n, U) = \frac{1}{2}\left(\sum_{b \in \{-1,1\}} \mid P(X_n = b) - P(U = b) \mid\right).$$

In order to compute $P(X_n = b)$ we make use of Corollary 4.13. Thus, taking into account that $P(a \in QR_n) = P(a \in J_n^+ \backslash QR_n) = 1/2$ because $a \in J_n^+$, we obtain

$$\begin{aligned} P(X_n = 1) =& P(x \in J_n^+ \ : \ x \leftarrow (a + QR_n)^*) \\ =& P(x \in J_n^+ \ : \ x \leftarrow (a + QR_n)^* \mid a \in QR_n) \cdot P(a \in QR_n) + \\ & P(x \in J_n^+ \ : \ x \leftarrow (a + QR_n)^* \mid a \in J_n^+ \backslash QR_n) \cdot P(a \in J_n^+ \backslash QR_n) \\ =& \left(\frac{1}{2} + \mathcal{O}\left(\frac{1}{n}\right)\right) \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2} + \mathcal{O}\left(\frac{1}{n}\right). \end{aligned}$$

In a similar way one can obtain

$$P(X_n = -1) = \frac{1}{2} - \mathcal{O}\left(\frac{1}{n}\right).$$

Now, the statistical distance $\Delta(X_n, U)$ becomes

$$\Delta(X_n, U) = \frac{1}{2}\left(\left|\frac{1}{2} + \mathcal{O}\left(\frac{1}{n}\right) - \frac{1}{2}\right| + \left|\frac{1}{2} - \mathcal{O}\left(\frac{1}{n}\right) - \frac{1}{2}\right|\right) = \mathcal{O}\left(\frac{1}{n}\right).$$

Since n is exponentially large in the security parameter λ, the statistical distance is negligible. □

It is well-known that the statistical indistinguishability implies the computational indistinguishability [120]. Therefore, the results mentioned above, namely Lemma 2.1 and Lemma 2.3 in [31], simply follow from Theorem 4.12. Moreover, our result does not make use of the QR assumption for $RSAgen$, nor of the security of Cocks' IBE scheme.

Lemma 2.2 in [31] states that the distributions

$$D_0(\lambda) = \{(a, c, n) \mid n \leftarrow RSAgen(\lambda),\ a \leftarrow J_n^+,\ c \leftarrow C_n^*(a)\}$$

and

$$D_1(\lambda) = \{(a, c, n) \mid n \leftarrow RSAgen(\lambda),\ a \leftarrow J_n^+,\ c \leftarrow G_n(a) \backslash C_n^*(a)\}$$

are computationally indistinguishable under the QR assumption for $RSAgen$. Moreover, the proof of this result in [31] uses the ANON-IND-ID-CPA security of Cocks' IBE scheme (because it uses Lemma 2.1). However, this is not necessary if one uses Theorem 4.12 instead of Lemma 2.1 (see [31] for the proof of Lemma 2.2).

We would like to emphasize that a and n are "variable" in the distributions $D_0(\lambda)$ and $D_1(\lambda)$, while they are fixed in the distributions in Theorem 4.12. The variability of a is very important to prove that $D_0(\lambda)$ and $D_1(\lambda)$ are computationally indistinguishable

under the QR assumption. This property allows, given $r \in J_n^+$, to find c and a such that $(c^2 - 4a)_n = r$. If a is fixed, finding c with the above property would have required the extraction of a square root of $(4a + r)_n$ modulo n without knowing the factorization of n (for details, the reader is referred to [31]).

Lemma 2.2 in [31] also implies that the distributions

$$D_{0,n,a} = \{c \mid c \leftarrow C_n^*(a)\}$$

and

$$D_{1,n,a} = \{c \mid c \leftarrow G_n(a) \backslash C_n^*(a)\},$$

where n is an RSA modulus and $a \in J_n^+$, are computationally indistinguishable under the QR assumption.

4.4 Future work

The results developed in Section 4.2 refer only to sequences of length two. A natural question is whether they can be extended to sequences of length three or more, such as

$$QR_n(a_2 + QNR_n(a_1 + J_n^{\pm}))$$

or

$$J_n^{\pm}(a_3 + QR_n(a_2 + QNR_n(a_1 + J_n^{\pm}))).$$

This question does not have a straightforward answer because we are looking for exact formulas and the increment for our sets is arbitrary (a_1, a_2, a_3, etc.).

Chapter 5

Kleptographic Attacks

As more and more countries require individuals and providers to hand over passwords and decryption keys [22], we might observe an increase in the usage of *subliminal channels*. Subliminal channels are secondary channels of communication hidden inside a potentially compromised communication channel. The concept was introduced by Simmons [226, 227, 228] as a solution to the *prisoners' problem*. In the prisoners' problem *Alice* and *Bob* are incarcerated and wish to communicate confidentially and undetected by their guard *Walter* who imposes to read all their communication. Note that *Alice* and *Bob* can exchange a secret key before being incarcerated.

Classical security models assume that the cryptographic algorithms found in a device are correctly implemented and according to technical specifications. Unfortunately, in the real world, users have little control over the design criteria or the implementation of a security module. When using a hardware device, for example a smartcard, the user implicitly assumes an honest manufacturer that builds devices according to the provided specifications. The idea of a malicious manufacturer that tampers with the device or embeds a backdoor in an implementation was first suggested by Young and Yung [261, 262]. As proof of concept, they developed secretly embedded trapdoor with universal protection (SETUP) attacks. These attacks combine subliminal channels and public key cryptography to leak a user's private key or a message. Young and Yung assumed a black-box environment[1], while mentioning the existence of other scenarios. The input and output distributions of a device with SETUP should not be distinguishable from the regular distribution. However, if the device is reverse engineered, the deployed mechanism may be detectable.

[1] A black-box is a device, process or system, whose inputs and outputs are known, but its internal structure or working is not known or accessible to the user (*e.g.* tamper proof devices).

Although SETUP attacks were considered far-fetched by some cryptographers, recent events [36, 205] suggest otherwise. As a consequence, this research area seems to have been revived [32, 45, 94, 214]. In [47], SETUP attacks implemented in symmetric encryption schemes are referred to as *algorithmic substitution attacks* (ASA). The authors of [47] point out that the sheer complexity of open-source software (*e.g.* OpenSSL) and the small number of experts who review them make ASAs plausible not only in the black-box model. ASAs in the symmetric setting are further studied in [45, 88] and, in the case of hash functions, in [28]. A link between *secret-key steganography* and ASAs can be found in [53].

A practical example of leaking user keys is the Dual-EC generator, a cryptographically secure pseudorandom number generator standardized by NIST. Internal NSA documents leaked by Edward Snowden [36, 205] indicated a backdoor embedded into the Dual-EC generator. As pointed out in [54], using the Dual-EC generator facilitates a third party to recover a user's private key. Such an attack is a natural application of Young and Yung's work. Some real world SETUP attack examples may be found in [70, 69]. Building on the earlier work of [250] and influenced by the Dual-EC incident, [94, 89] provide the readers with a formal treatment of backdoored pseudorandom generators (PRNG).

A more general model entitled *subversion attacks* is considered in [32]. This model includes SETUP attacks and ASAs, but generic malware and virus attacks are also included. The authors provide subversion resilient signature schemes in the proposed model. Their work is further extended in [214, 215], where subversion resistant solutions for one-way functions, signature schemes and PRNGs are provided. In [214], the authors point out that the model from [32] assumes the system parameters are honestly generated (but this is not always the case). In the discrete logarithm case, examples of algorithms for generating trapdoored prime numbers may be found in [126, 110].

A different method for protecting users from subversion attacks are *cryptographic reverse firewalls* (RF). RFs represent external trusted devices that sanitize the outputs of infected machines. The concept was introduced in [184, 96]. A reverse firewall for signature schemes is provided in [32].

5.1 Preliminaries

Covert channels [166] have the capability of transporting information through system parameters apparently not intended for information transfer. Subliminal channels and SETUP attacks are special cases of covert channels and achieve information transfer by

modifying the original specifications of cryptographic primitives[2]. We further restrict covert channels to two sub-cases: subliminal channels and SETUP attacks.

Definition 5.1 (Subliminal channel)**.** A *Subliminal channel* is an algorithm that can be inserted in a system such that it allows the system's owner to communicate[3] with a recipient without their communication being detected by a third party[4]. It is assumed that the prisoners' communication is encrypted using a secret/public key encryption scheme and the decryption function is accessible to the recipient.

Definition 5.2 (Secretly Embedded Trapdoor with Universal Protection - SETUP)**.** A *Secretly Embedded Trapdoor with Universal Protection* (SETUP) is an algorithm that can be inserted in a system such that it leaks encrypted private key information to an attacker through the system's outputs. Encryption of the private key is performed using an asymmetric encryption scheme. It is assumed that corresponding the decryption function is accessible only to the attacker.

Remark 5.1. Note that SETUP mechanisms are special cases of subliminal channels. In the SETUP case, the sender is the system, the recipient is the attacker, while the third party is the owner of the system.

Definition 5.3 (Covert channel indistinguishability - IND-COVERT)**.** Let C_0 be a black-box system that uses a secret key sk. Let $\mathcal{E}$ be the encryption scheme used by a covert channel as defined above, in Definitions 5.1 and 5.2. We consider C_1 an altered version of C_0 that contains a covert channel based on $\mathcal{E}$. Let A be a PPT algorithm which returns 1 if it detects that C_0 is altered. We define the advantage

$$ADV^{\text{IND-COVERT}}_{\mathcal{E},C_0,C_1}(A) = |Pr[A^{C_1(sk,\cdot)}(\lambda) = 1] - Pr[A^{C_0(sk,\cdot)}(\lambda) = 1]|.$$

If $ADV^{\text{IND-COVERT}}_{\mathcal{E},C_0,C_1}(A)$ is negligible for any PPT algorithm A, we say that C_0 and C_1 are *polynomially indistinguishable*.

Remark 5.2. In the case of SETUP attacks we refer to covert channel indistinguishability as SETUP indistinguishability - IND-SETUP and we identify $ADV^{\text{IND-SETUP}}_{C_0,C_1}(A) = ADV^{\text{IND-COVERT}}_{\mathcal{E},C_0,C_1}(A)$.

Remark 5.3. Remark 5.2 is a formalization of the indistinguishability property for a regular SETUP mechanism described in [262]. The authors of [32] propose a more general concept (*public undetectability*) that allows *Mallory* to tailor his attacks depending on each of his victim's public key. The two formalizations, SETUP indistinguishability and public undetectability, assume that the public parameters $(g, \mathbb{G}, \mathbb{H})$ and the secret/public

[2] for example, by modifying the way random numbers are generated
[3] through the system's outputs
[4] The sender and receiver will further be called prisoners and the third party warden.

key pair (x, z) are honestly generated. In some cases, *Mallory* can also maliciously generate these. This scenario is captured in [214] (*cliptographic game*). A consequence of the three formalizations is that C_0 and C_1 have the same security.

Remark 5.4. In some cases, if sk is known, the covert channel can be detected by using its description and parameters. Thus, depending on the context we will specify if A has access to sk or not. If $\mathcal{E}$ is a public key encryption scheme we always assume that A has access to the public key[5].

Throughout the paper, when presenting covert channels, we make use of the following additional algorithms:

- *Subliminal/Malicious ParamGen* – used by the prisoners/attacker(s) to generate their (his) parameters;
- *Subliminal/Malicious KeyGen* – used by the prisoners/attacker to generate their (his) keys;
- *Extract* – used by the recipient to extract the secret message;
- *Recovering* – used by the attacker to recover *Charlie*'s secret key.

The algorithms above are not implemented in D. For simplicity, covert parameters will further be implicit when describing an algorithm.

Trivial Subliminal Channel. The Schnorr signature supports a subliminal channel based on rejection sampling. We further describe the trivial subliminal channel.

Sign(m, sk): Choose $k \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $r \leftarrow g^k$, until $\omega \equiv r \bmod 2$. To sign a message $m \in \{0,1\}^*$ compute the values $e \leftarrow h(r\|m)$ and $s \leftarrow k - xe \bmod q$. Output the signature (e, s).

Extract(e, s) : To extract the embedded message ω compute $\omega \leftarrow g^s y^e \bmod 2$.

Young-Yung SETUP Attack on the Generalized ElGamal Signature. In [261, 262, 263, 264], the authors propose a kleptographic version of ElGamal signatures and prove it secure in the standard model under the HDH assumption. The Young-Yung SETUP mechanism can be easily adapted to the generalized ElGamal signature, while maintaining its security. The algorithms of the generalized version are shortly described below. We assume that user V is the victim of a malicious user M. After D signs at least two messages, M can recover V's secret key and thus impersonate V.

[5] found by means of reverse engineering the system, for example

Malicious ParamGen(pp): Let $H : \mathbb{G} \to \mathbb{Z}_q^*$ be a hash function. Output the public parameter $sp_M = H$. Note that H will be stored in D's volatile memory.

Malicious KeyGen(pp): Choose $x_M \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $y_M \leftarrow g^{x_M}$. Output the public key $pk_M = y_M$. The public key pk_M and H will be stored in D's volatile memory. The secret key is $sk_M = x_M$; it will only be known by *Mallory* and will not be stored in the black-box.

Signing Sessions: The possible signing sessions performed by D are described below. Let $i \geqslant 1$.

Session$_0(m_0, sk)$: To sign message $m_0 \in \mathbb{G}$, D does the following

$$k_0 \xleftarrow{\$} \mathbb{Z}_q^*, r_0 \leftarrow g^{k_0}, s_0 \leftarrow k_0^{-1}[h(m_0) - a \cdot h(r_0)] \bmod q.$$

The value k_0 is stored in D's volatile memory until the end of *Session*$_1$. Output the signature (r_0, s_0).

Session$_i(m_i, sk, pk_M)$: To sign message $m_i \in \mathbb{G}$, D does the following

$$z_i \leftarrow y_M^{k_{i-1}}, k_i \leftarrow H(z_i), r_i \leftarrow g^{k_i}, s_i \leftarrow k_i^{-1}[h(m_i) - a \cdot h(r_i)].$$

The value k_i is stored in D's volatile memory until the end of *Session*$_{i+1}$. Output the signature (r_i, s_i).

Recovering$(m_i, r_{i-1}, r_i, s_i, sk_M)$: Compute $\alpha \leftarrow r_{i-1}^{x_M}$ and $k_i \leftarrow H(\alpha)$. Recover a by computing

$$a \leftarrow h(r_i)^{-1}[h(m_i) - k_i \cdot s_i)].$$

Remark 5.5. Let S be an honest generator for the values r used by the Generalized ElGamal signature scheme and let σ_i denote the i-th internal state and $\rho_i = g^{\sigma_i}$ the i-th output of S. The mechanism described above can be seen as a malicious PRNG $\tilde{S}$ based on the honest PRNG S. We define the internal states and outputs of $\tilde{S}$ by

- $\tilde{\sigma}_0 = \sigma_0$, $\tilde{\rho}_0 = \rho_0$;
- $\tilde{\sigma}_i = H(y_M^{\tilde{\sigma}_{i-1}})$, $\tilde{\rho}_i = g^{\tilde{\sigma}_i}$, where $i \geqslant 1$.

In [94], the authors state that the Dual-EC generator does not output bits that are provably indistinguishable from random bits. To improve Dual-EC, they introduce $\tilde{S}$ and prove it secure under the HDH assumption.

Young-Yung SETUP Attack on the Schnorr Signature. In [261, 262, 263, 264], the authors propose a kleptographic version of Schnorr signatures and prove it IND-SETUP secure in the standard model under the HDH assumption. The algorithms of the SETUP attack are shortly described below. Note that after D signs at least two messages, *Mallory* can recover *Charlie*'s secret key and, thus, impersonate *Charlie*. The *Malicious ParamGen* and *Malicious KeyGen* algorithms are identical to the Generalized ElGamal version and thus are ommited.

Signing Sessions: The possible signing sessions performed by D are described below. Let $i \geqslant 1$.

Session$_0(m_0, sk)$: To sign message $m_0 \in \mathbb{G}$, D does the following

$$k_0 \xleftarrow{\$} \mathbb{Z}_q^*, r_0 \leftarrow g^{k_0}, e_0 \leftarrow h(r_0 \| m_0), s_0 \leftarrow k_0 - xe_0 \bmod q.$$

The value k_0 is stored in D's volatile memory until the end of *Session*$_1$. Output the signature (r_0, s_0).

Session$_i(m_i, sk, pk_M)$: To sign message $m_i \in \mathbb{G}$, D does the following

$$z_i \leftarrow y_M^{k_{i-1}}, k_i \leftarrow H(z_i), r_i \leftarrow g^{k_i}, e_i \leftarrow h(r_i \| m_i), s_i \leftarrow k_i - xe_i \bmod q.$$

The value k_i is stored in D's volatile memory until the end of *Session*$_{i+1}$. Output the signature (r_i, s_i).

Recovering$(m_i, e_{i-1}, e_i, s_i, sk_M)$: Compute $r_{i-1} \leftarrow g^{s_{i-1}} y^{e_{i-1}}$, $\alpha \leftarrow r_{i-1}^{x_M}$ and $k_i \leftarrow H(\alpha)$. Recover x by computing $x \leftarrow e_i^{-1}(k_i - s_i) \bmod q$.

5.2 Threshold Kleptographic Attacks

In this section, we extend the SETUP attacks of Young and Yung on digital signatures. We introduce the first SETUP mechanism that leaks a user's secret key, only if ℓ out of n malicious parties decide to do this. We assume that the signature schemes are implemented in a black-box equipped with a volatile memory, erased whenever someone tampers with it.

In the following we give a few examples where a threshold kleptographic signature may be useful.

Since digitally signed documents are just as binding as signatures on paper, if a recipient receives a document signed by A he will act according to A's instructions. Finding A's

private key, can aid a law enforcement agency into collecting additional informations about A and his entourage. In order to protect citizens from abuse, a warrant must be issued by a legal commission before starting surveillance. To aid the commission and to prevent abuse, the manufacturer of A's device can implement an ℓ out of n threshold SETUP mechanism. Thus, A's key can be recovered only if there is a quorum in favor of issuing the warrant.

Digital currencies (*e.g.* Bitcoin) have become a popular alternative to physical currencies. Transactions between users are based on digital signatures. When a transaction is conducted, the recipient's public key is linked to the transfered money. Only the owner of the secret key can now spend the money. To protect his secret keys, a user can choose to store them in a tamper proof device, called a hardware wallet. Let's assume that a group of malicious entities manages to infect some hardware wallets and they implement an ℓ out of n threshold SETUP mechanism. When ℓ members decide, they can transfer the money from the infected wallets without the owner's knowledge. If $\ell - 1$ parties are arrested, the mechanism remains undetectable as long as the devices are not reverse engineered.

In accordance with the original works, we prove that the threshold SETUP mechanisms are polynomially indistinguishable from regular signatures. Depending on the infected signature, we obtain security in the standard or random oracle model (ROM). To do so, we make use of a public key encryption scheme (introduced in Section 3.5.2) and Shamir's secret sharing scheme [221]. ROM security proofs are easily deduced from the standard model security proofs provided in this section. Thus, are omitted.

Conventions. We further consider that the attacks presented from now on are implemented in a device D that digitally signs messages. The owner of the device is denoted by V and his public key by pk_V. We assume that his secret key sk_V is stored only in D's volatile memory[6]. The victim V thinks that D signs messages using the signature scheme described in Section 3.3.1. We stress that *KeyGen* and *Verification* algorithms are identical to the ones from Section 3.3.1. Thus, *KeyGen* and *Verification* are omitted when presenting the attacks.

5.2.1 A SETUP Attack on the Generalized ElGamal Signature

We further introduce a new SETUP mechanism. Compared to Young-Yung's attack, it is very easy to modify our mechanism to allow ℓ out of n malicious parties to recover V's

[6] If V knows his secret key, he is able to detect a SETUP mechanism using its description and parameters (found by means of reverse engineering a black-box, for example).

secret key[7]. The best we were able to do, using Young-Yung's mechanism, was to devise an ℓ out of ℓ threshold scheme[8]. We point out, that like Young-Yung's mechanism, our proposed mechanism leaks data continuously to the attacker.

5.2.1.1 Description

To implement the attack, M works in almost the same environment as in Section 5.1. Thus, we only mention the differences between the two environments.

Signing Sessions: The possible signing sessions performed by D are described below. Let $i \geqslant 1$.

Session$_0(m_0, sk_V)$: To sign message $m_0 \in \mathbb{G}$, D does the following

$$k_0 \xleftarrow{\$} \mathbb{Z}_q^*, r_0 \leftarrow g^{k_0}, s_0 \leftarrow k_0^{-1}[h(m_0) - a \cdot h(r_0)] \bmod q.$$

The value k_0 is stored in D's volatile memory until the end of *Session*$_1$. Output the signature (r_0, s_0).

Session$_i(m_i, sk_V, pk_M)$: To sign message $m_i \in \mathbb{G}$, D does the following

$$k_i \leftarrow k_{i-1} \cdot H(y_M^{k_{i-1}}), r_i \leftarrow g^{k_i}, s_i \leftarrow k_i^{-1}[h(m_i) - a \cdot h(r_i)] \bmod q.$$

The value k_i is stored in D's volatile memory until the end of *Session*$_{i+1}$. We remark that s_i is used as a data carrier for M. Output the signature (r_i, s_i).

Recovering$(m_{i-1}, m_i, r_{i-1}, r_i, s_{i-1}, s_i, sk_M)$: Compute $\alpha \leftarrow [s_i \cdot H(r_{i-1}^{x_M})]^{-1}$. Recover a by computing

$$a \leftarrow \left(\alpha \cdot h(m_i) - s_{i-1}^{-1} \cdot h(m_{i-1})\right) \cdot \left(\alpha \cdot h(r_i) - s_{i-1}^{-1} \cdot h(r_{i-1})\right)^{-1} \bmod q.$$

The correctness of the *Recovering* algorithm can be obtained as follows. From *Session*$_{i-1}$ and *Session*$_i$, we obtain the value of k_{i-1}

$$k_{i-1} \equiv s_{i-1}^{-1}[h(m_{i-1}) - a \cdot h(r_{i-1})] \mod q \tag{5.1}$$

$$k_{i-1} \equiv [s_i \cdot H(y_M^{k_{i-1}})]^{-1} \cdot [h(m_i) - a \cdot h(r_i)] \mod q. \tag{5.2}$$

From equalities (5.1) and (5.2) we obtain

$$a \cdot \left(\alpha \cdot h(r_i) - s_{i-1}^{-1} \cdot h(r_{i-1})\right) \equiv \alpha \cdot h(m_i) - s_{i-1}^{-1} \cdot h(m_{i-1}) \bmod q.$$

[7] We refer the reader to Section 5.2.2.
[8] We refer the reader to Appendix J.

Using the above equality and the fact that $y_M^{k_{i-1}} = r_{i-1}^{x_M}$, we obtain the correctness of the *Recovering* algorithm.

Remark 5.6. Let T be an honest generator for the values r used by the Generalized ElGamal signature scheme and let σ_i denote the i-th internal state and $\rho_i = g^{\sigma_i}$ the i-th output of T. The mechanism described above can be seen as a malicious PRNG $\tilde{T}$ based on the honest PRNG T. We define the internal states and outputs of $\tilde{T}$ by

- $\tilde{\sigma}_0 = \sigma_0$, $\tilde{\rho}_0 = \rho_0$;
- $\tilde{\sigma}_i = \tilde{\sigma}_{i-1} \cdot H(y_M^{\tilde{\sigma}_{i-1}})$, $\tilde{\rho}_i = g^{\tilde{\sigma}_i}$, where $i \geqslant 1$.

In the case of Dual-EC, if an attacker M knows output $\tilde{\rho}_{i-1}$ then he can compute the internal state $\tilde{\sigma}_i$. In the case of $\tilde{T}$, computing $\tilde{\sigma}_i$ also requires knowledge of the previous internal state $\tilde{\sigma}_{i-1}$. Since $\tilde{\sigma}_{i-1}$ is secret, the generator is not harmful on its own. But, if used to generate ephemeral keys g^k for ElGamal based signatures[9], it leads to a backdoor that enables M to break the security of the system.

5.2.1.2 Security Analysis

In this section we state the security margin for our variant of the ElGamal signature SETUP. We will defer the security proof of this scheme until the next section, since the scheme is a special case of the scheme described in Section 5.2.2.1. We denote by $GEGS$ the Generalized ElGamal Signature and by $N-GEGS$ the scheme described in the previous subsection.

Theorem 5.1. If the number of signatures is polynomial and HDH is hard in $\mathbb{G}$ then $GEGS$ and $N-GEGS$ are IND-SETUP in the standard model. Formally, let A be an efficient PPT IND-SETUP adversary. There exists an efficient algorithm B such that

$$ADV^{\text{IND-SETUP}}_{MEG,GEGS,N-GEGS}(A) \leqslant 4\Gamma ADV^{\text{HDH}}_{\mathbb{G},g,H}(B),$$

where Γ is the number of infected signatures.

Remark 5.7. Similarly to Theorem 5.1, we obtain that if T is a secure PRNG[10], then $\tilde{T}$ is a secure PRNG in the standard model.

Remark 5.8. As in the case of Dual-EC, it is easy to see that if in the $N-GEGS$ scheme, we replace y_M with $y'_M \xleftarrow{\$} \mathbb{G}$, the SETUP mechanism becomes benign. The security margin of the SETUP-free system remains the same as the one stated in Theorem 5.1.

[9] A well known vulnerability of ElGamal based signatures is that using the same k value twice, leads to secret key recovery [66].

[10] The outputs of T are computationally indistinguishable from the uniform distribution.

5.2.2 A Threshold SETUP Attack on the Generalized ElGamal Signature

In this section we introduce an ℓ out of n threshold SETUP attack, based on $\mathcal{N}-\mathcal{GEGS}$. In this secret sharing scenario, user V is the victim of n malicious parties (denoted by $\{M_i\}_{1\leqslant i\leqslant n}$) that somehow convince the manufacturer of D to implement the described SETUP mechanism. After D signs $n+1$ messages, any coalition of ℓ participants M_i can recover V's secret key. Once the key is obtained, V can be impersonated. We remark that starting from signature $\ell-1$ some coalitions of M_i can impersonate V.

5.2.2.1 Description

To ease description, we assume without loss of generality, that the first ℓ participants M_i decide to recover V's secret key and denote by $M=\{m_i\}_{0\leqslant i\leqslant \ell}$, $R=\{r_i\}_{0\leqslant i\leqslant \ell}$, $S=\{s_i\}_{0\leqslant i\leqslant \ell}$, $SK_M=\{sk_i\}_{1\leqslant i\leqslant \ell}$. We present our proposed threshold SETUP scheme below.

Malicious Parties KeyGen(pp): Let $H:\mathbb{G}\rightarrow\mathbb{Z}_q^*$ be a hash function. For each M_i, $1\leqslant i\leqslant n$, choose $x_i\xleftarrow{\$}\mathbb{Z}_q^*$ and compute $y_i\leftarrow g^{x_i}$. Output the public keys $pk_i=y_i$. The public keys pk_i and H will be stored in D's volatile memory. The secret keys are $sk_i=x_i$; they will only be known by the respective M_i and will not be stored in the black-box.

Signing Sessions: The possible signing sessions performed by D are described below. Let $1\leqslant i\leqslant n$ and $j>n$.

Session$_0$ (m_0, sk_V): To sign message $m_0\in\mathbb{G}$, D does the following

$$k_0\xleftarrow{\$}\mathbb{Z}_q^*, r_0\leftarrow g^{k_0}, s_0\leftarrow k_0^{-1}[h(m_0)-a\cdot h(r_0)] \bmod q.$$

The device also chooses $\{f_j\}_{1\leqslant j<\ell}$ at random from $\mathbb{Z}_q^*$ and forms the polynomial $f(z)=k_0+f_1\cdot z+\ldots+f_{\ell-1}\cdot z^{\ell-1}$. The polynomial $f(z)$ is stored in D's volatile memory until the end of *Session$_n$*. Output the signature (r_0, s_0).

Session$_i$ (m_i, sk_V, pk_i): To sign message $m_i\in\mathbb{G}$, D does the following

$$\begin{aligned} k_i &\leftarrow f(i)\cdot H(y_i^{k_0}),\ if\ f(i)\not\equiv 0 \bmod q;\\ k_i &\xleftarrow{\$} \mathbb{Z}_q^*,\ otherwise;\\ r_i &\leftarrow g^{k_i}, s_i\leftarrow k_i^{-1}[h(m_i)-a\cdot h(r_i)] \bmod q. \end{aligned}$$

We remark that s_i is used as a data carrier for M_i. Output the signature (r_i, s_i).

Session$_j(m_j, sk_V)$: To sign message $m_j \in \mathbb{G}$, D does the following

$$k_j \xleftarrow{\$} \mathbb{Z}_q^*, r_j \leftarrow g^{k_j}, s_j \leftarrow k_j^{-1}[h(m_j) - a \cdot h(r_j)] \bmod q.$$

Output the signature (r_j, s_j).

Recovering(M, R, S, SK_M): Compute $\alpha_i \leftarrow [s_i \cdot H(r_0^{x_i})]^{-1}$ and $\Delta_i \leftarrow \prod_{j \neq i} \frac{j}{j-i}$, $i, j \leqslant \ell$. Recover a by computing

$$a \leftarrow \left(\sum_{i=1}^{\ell} \alpha_i \cdot h(m_i) \cdot \Delta_i - s_0^{-1} \cdot h(m_0)\right) \cdot \left(\sum_{i=1}^{\ell} \alpha_i \cdot h(r_i) \cdot \Delta_i - s_0^{-1} \cdot h(r_0)\right)^{-1} \bmod q \quad (5.3)$$

The correctness of the *Recovering* algorithm can be obtained as follows. From *Session*$_0$, we obtain the value of k_0

$$k_0 \equiv s_0^{-1}[h(m_0) - a \cdot h(r_0)] \mod q. \quad (5.4)$$

From *Sessions*$_i$, we obtain M_i's share

$$f(i) \equiv [s_i \cdot H(y_i^{k_0})]^{-1} \cdot [h(m_i) - a \cdot h(r_i)] \mod q.$$

Using Lagrange interpolation we use the shares $f(i)$, $1 \leqslant i \leqslant \ell$ to recover k_0

$$k_0 \equiv \sum_{i=1}^{\ell} [s_i \cdot H(y_i^{k_0})]^{-1} \cdot [h(m_i) - a \cdot h(r_i)] \cdot \Delta_i \mod q. \quad (5.5)$$

From equalities (5.4) and (5.5) we obtain

$$a \cdot \left(\sum_{i=1}^{\ell} \alpha_i \cdot h(r_i) \cdot \Delta_i - s_0^{-1} \cdot h(r_0)\right) \equiv \sum_{i=1}^{\ell} \alpha_i \cdot h(m_i) \cdot \Delta_i - s_0^{-1} \cdot h(m_0) \bmod q.$$

Using the above equality and the fact that $y_i^{k_0} = r_0^{x_i}$, we obtain the correctness of the *Recovering* algorithm.

Remark 5.9. The probability that key recovery is not possible due to failure is $\epsilon = 1 - (1 - 1/q)^{n-\ell+1}$. Since q is a large prime number, we have that $\epsilon \simeq 0$.

Remark 5.10. When all n participants are required to recover V's secret key, the scheme described in Appendix J requires two infected signatures, while the above scheme requires

n infected signatures. Thus, the scheme described in this section is less efficient in this case. Unfortunately, we could not devise a method to extend the scheme described in Appendix J to an ℓ out of n threshold scheme.

Remark 5.11. The mechanism described in this section requires the malicious parties to directly compute V's secret key. In some cases this raises security concerns. For example, if the mechanism is used for surveillance purposes and a warrant is issued, if V's secret key is directly computed, when the warrant expires V can still be impersonated. In Appendix I we present a two party protocol extension of our scheme in order to mitigate this issue. We could not find an extension for the scheme described in Appendix J.

Remark 5.12. In the scheme described above, D plays the role of a trusted dealer, that leaks the shares using a subliminal channel to the n participants. This design choice was made in order to minimize communication between the malicious parties. The only moment when the participants communicate is when ℓ of them want to recover V's secret key.

Another possible scenario, was to use a secret sharing protocol with or without a trusted dealer between the n parties. After the participants agree on a shared public key $y_M = g^{x_M}$, the manufacturer implements, for example, Young-Yung SETUP attack on the generalized ElGamal signature [11]. Note that this approach works without any modifications to the SETUP mechanism.

Remark 5.13. Let P be an honest generator for the values r used by the Generalized ElGamal signature scheme and let σ_i denote the i-th internal state and $\rho_i = g^{\sigma_i}$ the i-th output of P. The mechanism described above can be seen as a malicious PRNG $\tilde{P}$ based on the honest PRNG P. We define the internal states and outputs of $\tilde{P}$ by

- $\tilde{\sigma}_0 = \sigma_0$, $\tilde{\rho}_0 = \rho_0$;
- $\tilde{\sigma}_i = f(i) \cdot H(y_i^{\sigma_0})$, $\tilde{\rho}_i = g^{\tilde{\sigma}_i}$, where $f(z) = \sigma_0 + \sigma_1 \cdot z + \ldots + \sigma_{\ell-1} \cdot z^{\ell-1}$ and $1 \leqslant i \leqslant n$;
- $\tilde{\sigma}_j = \sigma_j$, $\tilde{\rho}_j = \rho_j$, where $j > n$.

Because σ_0 and σ_j, where $j > n$, are identical for P and $\tilde{P}$ generator $\tilde{P}$ remains unpredictable. In the case $1 \leqslant i \leqslant n$, a group of ℓ malicious parties can prove that their $\tilde{\rho}_i$ are not random, but they cannot compute $\tilde{P}$'s internal states. Thus, when used on its own $\tilde{P}$ is mostly harmless. Unfortunately, if it is used to generate r for ElGamal based signatures, then ℓ malicious parties can recover the V's secret key.

[11] that uses y_M

5.2.2.2 Security Analysis

In this subsection we prove that the threshold version described above, denoted $S-GEGS$, is indistinguishable from $GEGS$ if the attacker corrupted at most $\ell-1$ out of n malicious parties M_i.

Theorem 5.2. If HDH is hard in $\mathbb{G}$ then $GEGS$ and $S-GEGS$ are IND-SETUP in the standard model as long as at most $\ell-1$ malicious parties are corrupted by A. Formally, let A be an efficient PPT IND-SETUP adversary. There exists an efficient algorithm B such that

$$ADV^{\text{IND-SETUP}}_{MEG,GEGS,S-GEGS}(A) \leqslant 4(n-\ell+1)ADV^{\text{HDH}}_{\mathbb{G},g,H}(B).$$

Proof. Let A be an IND-SETUP adversary that is trying to distinguish between $GEGS$ and $S-GEGS$. A has access to "random coins" sampled uniformly from a set R. Without loss of generality, we further assume that A has corrupted the first $\ell-1$ malicious participants.

Algorithm 39. The IND-SETUP game.

Function `init()`**:**
  Choose the secret keys $a, x_1, \ldots, x_n \xleftarrow{\$} \mathbb{Z}_q^*$
  Compute the public keys $y \leftarrow g^a$, $y_1 \leftarrow g^{x_1}, \ldots, y_n \leftarrow g^{x_n}$
  Set $\mathcal{L}_1 \leftarrow \left(\cup_{i=1}^{\ell-1}\{x_i\}\right) \cup (\cup_{i=1}^{n}\{y_i\})$ and $i \leftarrow 0$
Function $C_0(a,\ m)$**:**
  Choose $k \xleftarrow{\$} \mathbb{Z}_q^*$
  Compute $r \leftarrow g^k$ and $s \leftarrow k^{-1}[h(m) - a \cdot h(r)]$
  return (r, s)
Function $C_1(a,\ m)$**:**
  if $i = 0$ **then**
    Choose $k, f_1, \ldots, f_{\ell-1} \xleftarrow{\$} \mathbb{Z}_q^*$ and set $k_0 \leftarrow k$
  else if $0 < i \leqslant n$ *and* $f(i) \not\equiv 0 \bmod q$ **then**
    Compute $k \leftarrow f(i) \cdot H(y_i^{k_0})$
  else
    Choose $k \xleftarrow{\$} \mathbb{Z}_q^*$
  end
  Compute $r \leftarrow g^k$, $s \leftarrow k^{-1}[h(m) - a \cdot h(r)]$ and $i \leftarrow i + 1$
  return (r, s)
`init()`
Choose $b \xleftarrow{\$} \{0,1\}$ and $\rho \xleftarrow{\$} R$
return $A^{C_b(a,\cdot)}(\rho, y, \mathcal{L}_1)$

Algorithm 39 describes the IND-SETUP game. The first and second rows set up the public keys. Then the $GEGS$ and $S-GEGS$ oracles are described. The challenger then flips a bit b and reveals oracle C_b. A then computes its guess b' for b. A wins if $b = b'$.

We proceed by modifying oracle C_1 (described in Algorithm 39) into oracle C_2 (described in Algorithm 40). The only difference between the two oracles is that in C_2 the values k_i, $0 < i \leqslant \ell - 1$, are chosen at random. Since Shamir's secret sharing scheme is information theoretically secure, an adversary cannot distinguish between C_1 and C_2.

Algorithm 40. Oracle C_2.

Function $C_2(a,\ m)$**:**
  if $i = 0$ **then**
    Choose $k, f_1, \ldots, f_{\ell-1} \xleftarrow{\$} \mathbb{Z}_q^*$ and set $k_0 \leftarrow k$
  else if $\ell \leqslant i \leqslant n$ *and* $f(i) \not\equiv 0 \bmod q$ **then**
    Compute $k \leftarrow f(i) \cdot H(y_i^{k_0})$
  else
    Choose $k \xleftarrow{\$} \mathbb{Z}_q^*$
  end
  Compute $r \leftarrow g^k$, $s \leftarrow k^{-1}[h(m) - a \cdot h(r)]$ and $i \leftarrow i + 1$
  return (r, s)

Since MEG is IND$ an adversary cannot distinguish between C_0 and C_2. Note that the number of k values that A has to distinguish is $n - \ell + 1$. Thus, we obtain the security margin.

□

Remark 5.14. Similarly to Theorem 5.2, we obtain that if P is a secure PRNG, then $\tilde{P}$ is a secure PRNG in the standard model.

Remark 5.15. As in the case of Dual-EC, it is easy to see that if in the $S-GEGS$ scheme, we replace y_i with $y_i' \xleftarrow{\$} \mathbb{G}$, $1 \leqslant i \leqslant n$, the SETUP mechanism becomes benign. The security margin of the SETUP-free system remains the same as the one stated in Theorem 5.2.

5.2.3 Other Applications

The schemes described in Section 5.2.2 and Appendix J can either directly be used on other signatures (*e.g.* variations of the Generalized ElGamal signature [180], Pointcheval-Stern signature [208]) or indirectly, *i.e.* some work must be done to recover g^k (*e.g.* Schnorr signature [219] - see Example 5.1, DSA [23]).

Example 5.1. To be more precise, we describe the method used in the case of Schnorr signatures. We place ourselves in the subgroup of order q generated by a $g \in \mathbb{Z}_p^*$, where p is prime. The signature generation algorithm is

$$k \xleftarrow{\$} \mathbb{Z}_q^*, r \leftarrow h(g^k||m), s \leftarrow a \cdot r + k \bmod q.$$

In order to recover g^k, one must compute

$$g^s \cdot y^{-r} \equiv g^{s-ar} \equiv g^k.$$

After finding a method to recover g^k, either directly or by computing it from the signature, it is fairly easy to use the methods described in Section 5.2.2 and Appendix J. All the signatures presented in this section either have g^k directly embedded in them or the recovering mechanism is similar to the one presented in Example 5.1.

Some signature schemes that can be tampered with and also have the same security as $S - GEGS$ are: variations of the Generalized ElGamal signature [180], ECDSA [30], ECDSA variants [175], Katz-Wang signature [152], KCDSA [168], Elliptic Curve GOST [97], EDL signature Goh-Jarecki variant [118], EDL signature Chevallier variant [72] and Elliptic Curve Nyberg-Rueppel [183].

If $\mathbb{G}$ is generated by an element $g \in \mathbb{Z}_p^*$ of order q, we can apply the same methods and obtain security in the standard model[12] for the following algorithms: DSA [23], GOST [182], Nyberg-Rueppel [197], Nyberg-Rueppel IEEE variant [183], Pointcheval-Stern signature [208], Schnorr signature [219] and Girault-Poupard-Stern (GPS) signature [114], if parameter A used by the GPS signature is prime.

Schnorr [219] and Girault-Poupard-Stern signatures [114] are derived from identification schemes. As a consequence, we can apply similar methods to infect these identification schemes and compromise V's secret key. Another identification scheme that offers the possibility of embedding a secret trapdoor is Okamoto's scheme [200].

Signcrypt algorithms [268, 269] use a variation of the ElGamal signature in order to authenticate messages and use a key derivation function based on the recipient's secret key in order to encrypt messages. So, if we embed the threshold SETUP mechanism in the signature and manage to recover the signer's secret key, then we can also decrypt all the messages that the signer receives.

Changing the setting to identity based signatures (IBS), we observe that Cha-Cheon IBS [74], Hess IBS [142] and Paterson IBS [203] can be infected and the resulting schemes are

[12] We refer the reader to Remark 3.2.

secure in ROM[13]. A signature that can be tampered with and obtain the same security as $S - GEGS$, is Bellare-Namprempre-Neven IBS [46]. This signature offers an extra feature, we can also modify the extraction algorithm, permitting ℓ out of n legitimate users to obtain the master key (used by the central authority to generate keys for any legitimate user).

When random numbers are not available or of questionable quality (*e.g.* malicious RNG), one may use deterministic signatures. One such example is the deterministic variant of the Schnorr signature scheme introduced in [189]. The authors suggest to choose $k \leftarrow h(\kappa, m, pp)$, where κ is a fixed secret. Unfortunately, this approach does not protect V. When implementing a SETUP attack for this scheme, we must ensure the same functionality as in the SETUP-free version (*i.e.* signing the same message multiple times yields the same signature). In the following we give two attacks for this deterministic signature. In the first attack, a malicious party replaces κ by $\kappa' \leftarrow H(y_M^a)$ and recovers V's secret key by computing $a \leftarrow h(r)^{-1}[h(m) - k \cdot s]$. This attack can be easily extended to an ℓ out of ℓ attack[14], but we were not able to extend it to an ℓ out of n attack. In the second attack, D stores a list $\mathcal{L}$ containing the messages received as input and the associated signatures. When a message m is received, D will first search m in $\mathcal{L}$. If m is found, D will return the stored signature, else it will generate a new infected signature. If D runs out of memory, it reverts to $k \leftarrow h(\kappa, m, pp)$. To save memory an attacker could, for example, restrict D to maliciously signing only short messages.

5.2.4 Future Work

An interesting area of research would consist in finding a method to extend SETUP attacks applied to encryption schemes to threshold SETUP attacks. Also, it would be interesting to see if one can mount a successful SETUP attack or threshold SETUP attack if threshold signature schemes are used.

In Appendix J we describe an ℓ out of ℓ threshold SETUP mechanism that uses only two sessions in order to recover V's secret key. An extension to ℓ out of n may be more efficient than the approach from Section 5.2.2.1.

5.3 Unifying Framework

The initial model proposed by Young and Yung is the black-box model. For our intended purposes this model suffices, since the zero-knowledge protocols we attack were designed

[13] We refer the reader to Remark 3.2.
[14] We refer the reader to Appendix J.

for smartcards. An important property is that infected smartcards should have inputs and outputs indistinguishable from regular smartcards. However, if the smartcard is reverse engineered, the deployed mechanism may be detectable.

There are two methods to embed backdoors into a system: either you generate special public parameters (SPP) or you infect the random numbers (IRN) used by the system. In the case of discrete logarithm based systems, SPP and IRN were studied in [261, 262, 263, 264, 126, 110]. We only found SPP [83, 261, 262, 265, 264] and not IRN in the case of factorization based systems.

Using the same level of abstraction as in [176], we show how an attacker (called *Mallory*) can insert a backdoor into the UZK protocol and extract *Peggy*'s secret. When instantiated, this attack provides new insight into SETUP attacks. In particular, we provide the first IRN attack on a factoring based system and the first attack on systems based on e^{th}-root representations. We also provide the reader with new instantiations of Maurer's unified protocol: the Girault protocol, a new proof of knowledge for discrete logarithm representation in $\mathbb{Z}_n^*$ and a proof of knowledge of an e^{th}-root representation.

The second SETUP attack we introduce is a generalization of Young and Yung's work. When instantiated with the Schnorr protocol, we obtain their results. We also provide other examples not mentioned by Young and Yung.

Conventions. Compared to Section 3.2.1.1, we also assume that $\mathbb{G}$ is a cyclic group. Note that this implies that $\mathbb{G}$ is commutative. Let g be a generator of $\mathbb{G}$. We denote by αg the element $g \star \ldots \star g$ obtained by repeatedly applying the group operation $\alpha - 1$ times.

We further consider that the attacks presented from now on are implemented in a device D that is used by *Peggy* to prove the knowledge of x. We assume that x is stored only in D's volatile memory[15]. Note that *Peggy* believes that D works in accordance with the UZK protocol.

5.3.1 Main Unified SETUP Attack

The protocol described in this section is a SETUP attack against UZK that allows *Mallory* to extract *Peggy*'s knowledge of x. We only show how to infect two sessions of the protocol and assume that the rest of the sessions remain unmodified.

[15] If *Peggy* knows her secret she is able to detect the SETUP mechanism using its description and parameters (found by means of reverse engineering a black-box, for example).

5.3.1.1 Description

Before stating the results, we first make some preliminary assumptions. Let $h : \mathbb{H} \to \mathbb{G}$ be a hash function and let $i = 0, 1$ be an index. We assume that *Peggy* runs the protocols at least two times (*i.e.* once for $i = 0$ and once for $i = 1$). We denote by $y \leftarrow [g]^{x_M}$ *Mallory*'s public key, while $x_M \xleftarrow{\$} |\mathbb{G}|$ is his secret key. Note that y is stored on D's volatile memory. All the data we save will also be stored on D's volatile memory.

In Figure 5.1 we present the main protocol against UZK. We depict in red the modifications on UZK to obtain our SETUP attack. Note that after session 0 the index is incremented.

Peggy		*Victor*
Knows x. Computes $z = [x]$.		Knows z.
If $i = 0$ then: - choose $\alpha \xleftarrow{\$} \|\mathbb{G}\|$ and store α, - compute $k_0 \leftarrow \alpha g$ and $t_0 \leftarrow [g]^{\alpha}$. Else: - compute $k_1 \leftarrow k_0 \star h(y^{\alpha})$ and $t_1 \leftarrow [k_1]$, - erase α.		
	$\xrightarrow{t_i}$	
		Choose $c_i \xleftarrow{\$} \mathcal{C} \subset \mathbb{N}$.
	$\xleftarrow{c_i}$	
Compute $r_i \leftarrow k_i \star x^{c_i}$.		
	$\xrightarrow{r_i}$	
		If $[r_i] = t_i \otimes z^{c_i}$ return `true`. Else return `false`.

FIGURE 5.1: The main unified SETUP attack.

We further show how *Mallory* can extract *Peggy*'s knowledge if she uses a device that is infected with US-1.

Theorem 5.3. If *Peggy* uses US-1 and UZK satisfies the conditions from Theorem 3.1, then *Mallory* can compute an $\tilde{x}$ such that $[\tilde{x}] = z$. More precisely,

$$\tilde{x} = u^a \star (r_1^{-1} \star r_0 \star h(t_0^{x_M}))^b,$$

where a and b are computed using Euclid's extended gcd algorithm such that $\ell a + (c_0 - c_1)b = 1$.

Proof. From the definitions of r_0 and r_1 we obtain the following relations

$$[r_0] = [k_0 \star x^{c_0}] = t_0 \otimes z^{c_0} \text{ and } [r_1] = [k_1 \star x^{c_1}] = [k_0 \star h(y^\alpha) \star x^{c_1}] = t_0 \otimes [h(y^\alpha)] \otimes z^{c_1}.$$

Let $\beta = h(y^\alpha) = h(t_0^{x_M})$. We make use of

$$[r_1^{-1} \star r_0] = [r_1^{-1}] \otimes [r_0] = z^{-c_1} \otimes [\beta]^{-1} \otimes z^{c_0} = z^{c_0 - c_1} \otimes [\beta]^{-1}$$

and Theorem 3.1 to see that *Mallory* can compute an $\tilde{x}$ such that $[\tilde{x}] = z$

$$\begin{aligned}
[\tilde{x}] &= [u^a \star (r_1^{-1} \star r_0 \star \beta)^b] \\
&= [u]^a \otimes ([r_1^{-1} \star r_0] \otimes [\beta])^b \\
&= (z^\ell)^a \otimes (z^{c_0 - c_1} \otimes [\beta]^{-1} \otimes [\beta])^b \\
&= z^{\ell a + (c_0 - c_1)b} = z.
\end{aligned}$$

□

Remark 5.16. UZK can be transformed into a signature scheme using the Fiat-Shamir transform [108]. Thus, obtaining a unified signature scheme. Note that the SETUP attacks described for UZK are preserved by the Fiat-Shamir transform, therefore *Mallory* can recover *Peggy*'s signing key by using either of them.

5.3.1.2 Security Analysis

We continue by stating the security margin for the IND-SETUP between UZK and US-1.

Theorem 5.4. If HDH is hard in $\langle [g] \rangle$ then UZK and US-1 are IND-SETUP in the standard model. Formally, let A be an efficient PPT IND-SETUP adversary. There exists an efficient algorithm B such that

$$ADV^{\text{IND-SETUP}}_{\text{UZK,US-1}}(A) \leqslant 2ADV^{\text{HDH}}_{\langle [g] \rangle, [g], h}(B).$$

Proof. Let A be an IND-SETUP adversary trying to distinguish between UZK and US-1. We show that A's advantage is negligible. We construct the proof as a sequence of games in which all the required changes are applied to US-1. Let W_i be the event that A wins game i.

Game 0. The first game is identical to the IND-SETUP game[16]. Thus, we have

$$|2Pr[W_0] - 1| = ADV^{\text{IND-SETUP}}_{\text{UZK,US-1}}(A). \tag{5.6}$$

Game 1. In this game, $h(y^\alpha)$ from *Game 0* becomes $[g]^z$, where $z \xleftarrow{\$} |\mathbb{G}|$. Since this is the only change between *Game 0* and *Game 1*, A will not notice the difference assuming the HDH assumption holds. Formally, this means that there exists an algorithm B such that

$$|Pr[W_0] - Pr[W_1]| = ADV^{\text{HDH}}_{\langle [g] \rangle, [g], h}(B). \tag{5.7}$$

Game 2. The last change we make is $k_0, k_1 \xleftarrow{\$} \mathbb{G}$. Adversary A will not notice the difference, since

- α is a random exponent and $\mathbb{G}$ is cyclic
- multiplying k_0 with a random element yields a random element.

Formally, we have that

$$Pr[W_1] = Pr[W_2]. \tag{5.8}$$

The changes made to US-1 in *Game 1* and *Game 2* transformed it into UZK. Thus, we have

$$Pr[W_2] = 1/2. \tag{5.9}$$

Finally, the statement is proven by combining the equalities (5.10) – (5.13).

□

5.3.2 A Supplementary SETUP Attack

Compared to US-1, the supplementary protocol only allows *Mallory* to compute x in some specific instantiations of UZK. Again, we only show how to infect two sessions of the protocol.

[16] as in Remark 5.2

5.3.2.1 Description

In Figure 5.2 we present a supplementary protocol against UZK. Again, we depict in red the modifications made to UZK to obtain our SETUP attack. Note that after session 0 the index is incremented.

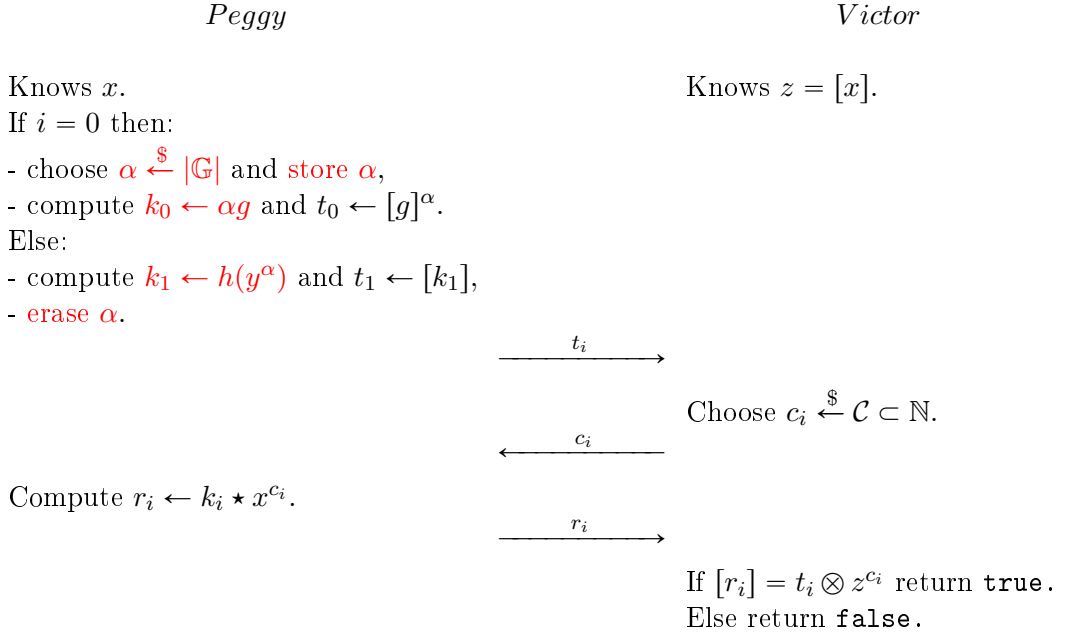


FIGURE 5.2: A supplementary unified SETUP attack.

Unlike US-1, with US-2 *Mallory* cannot extract *Peggy*'s knowledge except for some particular instantiations of UZK. More precisely, if *Mallory* knows or can compute the cardinal of $\mathbb{G}$ then he can extract *Peggy*'s knowledge.

Theorem 5.5. If *Peggy* uses US-2 and $|\mathbb{G}|$ is publicly known, then *Mallory* can compute an $\tilde{x}$ such that $[\tilde{x}] = z$, with probability $\varphi(|\mathbb{G}|)/|\mathbb{G}|$. More precisely,

$$\tilde{x} = (r_1 \star (h(t_0^{x_M}))^{-1})^{c_1^{-1}}.$$

Proof. Let $\beta = h(y^{\alpha}) = h(t_0^{x_M})$. From the definition of r_1 we can easily extract x by computing

$$x = (r_1 \star k_1^{-1})^{c_1^{-1}} = (r_1 \star \beta^{-1})^{c_1^{-1}}.$$

□

5.3.2.2 Security Analysis

We further state the security margin for the IND-SETUP between UZK and US-2. We omit the proof due to its similarity to Theorem 5.4.

Theorem 5.6. If HDH is hard in $\langle [g] \rangle$ then UZK and US-2 are IND-SETUP in the standard model. Formally, let A be an efficient PPT IND-SETUP adversary. There exists an efficient algorithm B such that

$$ADV^{\text{IND-SETUP}}_{\text{UZK,US-2}}(A) \leqslant 2ADV^{\text{HDH}}_{\langle [g] \rangle,[g],h}(B).$$

5.3.3 Special Cases of the Unified SETUP Attacks

In this section we describe a number of attacks based on US-1 and US-2 for different instantiations UZK.

5.3.3.1 Proofs of Knowledge of a Discrete Logarithm

Let $p = 2q + 1$ be a prime number such that q is also prime. Select an element $h \in \mathbb{H}_p$ of order q in some multiplicative group of order $p-1$. The discrete logarithm of an element $z \in \mathbb{H}_p$ is an exponent x such that $z = h^x$. We further describe a protocol for proving the knowledge of a discrete logarithm.

The Schnorr protocol [219][17] is a special case of UZK where $(\mathbb{G}, \star) = (\mathbb{Z}_q, +)$ and $\mathbb{H} = \langle h \rangle$. The one-way group homomorphism is defined by $[x] = h^x$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, q-1]$. According to [176], the conditions of Theorem 3.1 are satisfied for $\ell = q$ and $u = 0$.

Standard instantiation of the Schnorr protocol define $\mathbb{H}_p$ either as $\mathbb{Z}_p^*$ or as an elliptic curve, so according to Remark 3.2, we can safely apply both SETUP attacks. Thus, for the first attack we have the following parameters

$$g \leftarrow 1, k_0 \leftarrow \alpha, t_0 \leftarrow h^\alpha, k_1 \leftarrow k_0 + h(y^\alpha), t_1 \leftarrow h^{k_1}.$$

According to Theorem 5.3, *Peggy*'s secret can be recovered by computing

$$\tilde{x} = (c_0 - c_1)^{-1}(r_0 - r_1 + h(t_0^{x_M})).$$

[17] This proof can be seen as a more efficient version of a proposal made by Chaum *et al.* [68].

For the second attack the only change in the protocol is $k_1 \leftarrow h(y^{\alpha})$. According to Theorem 5.5, *Mallory* can recover *Peggy*'s secret by computing

$$\tilde{x} = c_1^{-1}(r_1 - h(t_0^{x_M})).$$

Remark 5.17. Recovering x when *Peggy* uses US-2 was first described in a series of papers by Young and Yung [261, 262, 263, 264]. Remark that in this setting computing x is a little bit more efficient than in the case of US-1.

We further describe a variation of the Schnorr protocol introduced by Girault [113][17]. Thus, let $p = 2fp' + 1$ and $q = 2fq' + 1$ be prime numbers such that f, p' and q' are distinct primes. Select an element $h \in \mathbb{Z}_n^*$ of order f, where $n = pq$. Note that p and q are secret.

Using the UZK notations we have $(\mathbb{G}, \star) = (\mathbb{Z}_f, +)$ and $\mathbb{H} = \langle h \rangle$. The one-way group homomorphism is defined by $[x] = h^x$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, f-1]$. It is easy to see that $\ell = f$ and $u = 0$ satisfy the two conditions of Theorem 3.1.

Since HDH is hard in $\mathbb{H}$[18] then both attacks can be mounted. Note that the attacks can be easily derived from the attacks on the Schnorr protocol.

5.3.3.2 Proofs of Knowledge of an e^{th}-root

Let p and q be two safe prime numbers such that $(p-1)/2$ and $(q-1)/2$ are also prime. Compute $n = pq$ and choose a prime e such that $\gcd(e, \varphi(n)) = 1$. An e^{th}-root of an element $z \in \mathbb{Z}_n^*$ is a base x such that $z \equiv x^e \bmod n$. Note that the e^{th}-root is not unique. We further describe a protocol for proving the knowledge of an e^{th}-root.

The Guillou-Quisquater protocol [131] is a special case of UZK where $(\mathbb{G}, \star) = (\mathbb{H}, \otimes) = (\mathbb{Z}_n^*, \cdot)$. The one-way group homomorphism is defined by $[x] = x^e \bmod n$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, e-1]$. According to [176], the conditions of Theorem 3.1 are satisfied for $\ell = e$ and $u = z$. Note that when $e = 2$ we obtain the protocol introduced by Fiat and Shamir [108].

Remark 5.18. Before stating the parameters for the SETUP attacks we must first address two issues. The first issue is that both SETUP attacks assume that a generator g is known to *Mallory*. This is needed in order to set-up *Mallory*'s public key. But n is generated internally by *Peggy*'s device and no generator for $\mathbb{Z}_n^*$ is publicly available in the general case. To remove this impediment we always choose $p, q \equiv 3$ or $5 \bmod 8$.

[18] See Remark 3.2

According to [177] this ensures us that 2 is a generator for both $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$. Hence, 2 is also a generator for $\mathbb{Z}_n^*$. If p and q are stored only in *Peggy*'s device, then she cannot distinguish this particular choice of primes from other randomly chosen primes, since she only has access to n.

The last issue that we have to address is the selection of *Mallory*'s secret key. Let's assume that n is a λ-bit integer. Since $\phi(n)$ is unknown to *Mallory*, instead of choosing $x_M \xleftarrow{\$} |\mathbb{Z}_n^*|$, he will choose $x_M \xleftarrow{\$} [0, 2^\lambda]$. It is easy to see that the statistical distance between the two distributions is $(\phi(n) - 2^\lambda)/\phi(n)$. Thus, it is negligible.

Since HDH is hard in $\mathbb{H}$[18] and it is infeasible to compute $|\mathbb{G}|$, then only US-1 can be applied. Thus, we have the following parameters for US-1

$$g \leftarrow 2, k_0 \leftarrow 2^\alpha, t_0 \leftarrow 2^{\alpha e}, k_1 \leftarrow k_0 h(y^\alpha), t_1 \leftarrow h^{k_1}.$$

According to Theorem 5.3, *Peggy*'s secret can be recovered by computing

$$\tilde{x} \equiv z^a \cdot (r_1^{-1} r_0 \cdot h(t_0^{x_M}))^b \bmod n.$$

5.3.3.3 Proofs of Knowledge of a Discrete Logarithm Representation

Let $p = 2q + 1$ be a prime number such that q is also prime. Select m elements $h_1, \ldots, h_m \in \mathbb{H}_p$ of order q in some multiplicative group of order $p - 1$. A discrete logarithm representation of an element $z \in \langle h_1, \ldots, h_m \rangle$ is a list of exponents $(x_1, \ldots, x_m)$ such that $z = h_1^{x_1} \ldots h_m^{x_m}$. Note that discrete logarithm representations are not unique. We further describe a protocol for proving the knowledge of a discrete logarithm representation.

A protocol for proving the knowledge of a representation is presented in [176][17]. To instantiate UZK and obtain Maurer's protocol we set $\mathbb{G} = \mathbb{Z}_q^m$ with $\star$ defined as addition applied component-wise and $\mathbb{H} = \langle h_1, \ldots, h_m \rangle$. The one-way group homomorphism is defined by $[(x_1, \ldots, x_m)] = h_1^{x_1} \ldots h_m^{x_m}$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, q - 1]$. According to [176], the conditions of Theorem 3.1 are satisfied for $\ell = q$ and $u = (0, \ldots, 0)$. Note that when $m = 2$ we obtain a protocol introduced by Okamoto [200].

The SETUP attacks for this protocol can be easily derived from the attacks on the Schnorr protocol and, thus, are omitted.

Chaum *et al.* [68] also provide a variant for their protocol when n is composite. Thus, by adapting the Girault protocol and tweaking the Maurer protocol, we can obtain a more

efficient version of the Chaum *et al.* protocol. Using the notations from the Girault protocol, we set $\mathbb{G} = \mathbb{Z}_f^m$ and $\mathbb{H} = \langle h_1, \ldots, h_m \rangle$, where $h_1, \ldots, h_m \in \mathbb{Z}_n^*$ are elements of order f. The one-way group homomorphism is defined by $[(x_1, \ldots, x_m)] = h_1^{x_1} \ldots h_m^{x_m}$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $\mathbb{Z}_f$. It is easy to see that $\ell = f$ and $u = (0, \ldots, 0)$. Note that US-1 and US-2 can also be mounted in this setting.

5.3.3.4 Proofs of Knowledge of an e^{th}-root Representation

Let p and q be two prime numbers such that $(p-1)/2$ and $(q-1)/2$ are also prime. Compute $n = pq$ and choose primes $e_1, \ldots, e_m$ such that $\gcd(e_i, \varphi(n)) = 1$, for $1 \leqslant i \leqslant m$. An e^{th}-root representation of an element $z \in \mathbb{Z}_n^*$ is a list of bases $(x_1, \ldots, x_m)$ such that $z \equiv x_1^{e_1} \ldots x_m^{e_m} \bmod n$. Note that e^{th}-root representations are not unique. We further describe a protocol for proving the knowledge of an e^{th}-root representation.

A protocol for proving the knowledge of an e^{th}-root representation can be obtained from UZK if we set $\mathbb{G} = (\mathbb{Z}_n^*)^m$ with $\star$ defined as multiplication applied component-wise and $(\mathbb{H}, \otimes) = (\mathbb{Z}_n^*, \cdot)$. The one-way group homomorphism is defined by $[(x_1, \ldots, x_m)] = x_1^{e_1} \ldots x_m^{e_m} \bmod n$ and the challenge space $\mathcal{C}$ can be any arbitrary subset of $[0, e-1]$, where e is a prime such that $\gcd(e, \phi(n)) = 1$. Since all e_i are coprime then there exist α_is such that $\alpha_1 e_1 + \ldots + \alpha_m e_m = 1$. Then, it is easy to see that $\ell = 1$ and $u = (z^{\alpha_1}, \ldots, z^{\alpha_m})$.

The US-1 SETUP attack for this protocol can be easily derived from the attack on the Guillou-Quisquater protocol and, thus, is omitted.

5.3.4 Future Work

In Section 5.2 we can find an extensive list of signature schemes that are vulnerable to SETUP attacks. Thus, an interesting direction of research is abstracting digital signatures[19] and devising a method for attacking all of them at once, instead of tweaking the attacks for each individual signature.

5.4 Kleptographic Subscription Plans

One of the classical business models for kleptographic attacks is the following: a client[20] C pays up front a manufacturer M, whom will later implement a certain backdoor

[19] not only the ones obtained using the Fiat-Shamir transform
[20] by definition a malicious entity

in a tamper proof device and deliver that device to a victim. This model puts the manufacturer at an advantage, because he can charge the customer and not implement the requested backdoor. Since this transaction is illegal, the customer can not file a complain and legally retrieve his money. Thus, this might scare off some of the potential clients.

Another classical model is the following: a client pays the manufacturer half the money up front and the rest after checking the correctness of the backdoor. If the manufacturer does not take certain precautions, then the client is at an advantage. For example, C checks the correctness of the backdoor, but fails to pay the second installment. This can be easily avoided if a backdoor deactivation method is put in place by M[21]. A possible deactivation strategy is for M to send D a special input that instructs the device to erase all incriminating evidence. A similar approach is used in [88, 109] to trigger backdoors.

Both classical approaches have an inherent risk for the manufacturer: the client can easily prove that M backdoored D either by decrypting all the messages send through that device or by revealing the private keys stored in D. Thus, to make the risk worth while the manufacturer must charge C a high embedding fee. This will certainly scare away certain resource constrained clients (*e.g.* small businesses that do not have the resources of a large corporation). To address this issue, we introduce a subscription based model suitable for the ElGamal encryption algorithm.

Our model draws inspiration from the subscription services offered by companies like Netflix [6], Amazon [7] and HBO [8]. These companies give access to streaming content in exchange for a monthly pay. In our case, a client pays for a backdoor that gives him access to a limited number of private messages. Subsequently, the client has to renew his subscription. This balances the risk and reward factors for the manufacturer[22] and, in consequence, M can lower embedding fees. A risk still remains: no guarantees of output delivery for the clients. But, this is minimum in a subscription based model because the goal of the manufacturer is to keep clients satisfied, so they further renew their subscription[23].

Compared to the classical models, our proposed model has a different issue that needs to be tackled. Clients want access to their services as soon as they pay. But, illegal transactions mostly use cryptocurrencies [75] and the average confirmation time for this type of transactions is large in some cases (*e.g.* for Bitcoin, it takes on average an hour per transaction [2]). Thus, to give the manufacturer sufficient time for deactivating the

[21] As in the previous model, the transaction is illegal and thus, M can not take legal action against C.

[22] M is exposed only for a limited period of time

[23] Cheating a client will only bring M a small amount of revenue.

backdoor[24] if the transaction is not valid, we employ a mechanism similar to time-lock puzzles [213] .

Note that generic kleptographic countermeasures [214, 215, 135] can protect tamper proof device's users against our proposed mechanisms. Unfortunately, unless users do not explicitly require the implementation of these defences, a manufacturer is not obliged to deploy them. Thus, M is free to implement any kleptographic mechanism.

5.4.1 Preliminaries

Conventions. All kleptographic subscriptions presented from now on are implemented in a device D. The owner of the device is denoted by V and we assume that he is in possession of his secret key. Note that V thinks that D contains an implementation of the ElGamal scheme as described in Section 7.3. When one of the original ElGamal algorithms is not modified by the SETUP attack, the scheme will be omitted when presenting the respective attack.

Throughout the paper, when presenting kleptographic subscriptions, we make use of the following additional algorithms:

- *Device's/Manufacturer's/Customer's KeyGen* – used by the device/manufacturer/customer to generate its/his keys;
- *Token* – used by the customer/manufacturer to extract the access token;
- *Extract* – used by the customer to recover the messages sent by V.

The previously mentioned algorithms are not implemented in D. For simplicity, kleptographic parameters will further be implicit when describing a scheme.

5.4.1.1 Security Assumptions

Definition 5.4 (Pseudorandom Function - PRF)**.** A function $F : \mathbb{G} \times [1, n] \rightarrow S$ is a PRF if:

- Given a key $K \in \mathbb{G}$ and an input $X \in [1, n]$ there is an efficient algorithm to compute $F_K(X) = F(X, K)$.

[24] by means of special triggers

- Let A be a PPT algorithm with access to an oracle $\mathcal{O}$ that returns 1 if $\mathcal{O} = F_K(\cdot)$. The PRF-advantage of A, defined as

$$ADV_F^{\text{PRF}}(A) = \left|Pr[A^{F_K(\cdot)} = 1 | K \xleftarrow{\$} \mathbb{G}] - Pr[A^{F(\cdot)} = 1 | F \xleftarrow{\$} \mathcal{F}]\right|$$

  must be negligible for any PPT algorithm A, where $\mathcal{F} = \{F : [1, n] \to S\}$.

Definition 5.5 (Pseudorandom Permutation - PRP)**.** A PRF $P : \mathbb{G} \times [1, n] \to [1, n]$ is a PRP if P is one-to-one and $\mathcal{F}$ from Definition 5.4 is changed into $\mathcal{F} = \{F : [1, n] \to [1, n] \mid F \text{ is one-to-one}\}$. The PRP-advantage of A is denoted $ADV_P^{\text{PRP}}(A)$.

5.4.2 Free Subscription

The first type of subscription (denoted by FS) is an analog of public television channels. Thus, anyone who is in possession of the transmitted ciphertexts can decrypt them after a certain amount of traffic has been sent. This protocol will form the basis for the mechanisms presented in Sections 5.4.3 and 5.4.4.

Although, this kind of subscription does not bring any revenue, it can still be useful in certain situations. For example, a disgruntled employee can embed it in the source code of certain products before leaving the company. Then, he can anonymously point out that the respective company implemented backdoors in their products. The scope of this scenario is to damage the company's reputation.

5.4.2.1 Description

Let n be the maximum number of messages that a client needs to wait before recovering all of V's communications. Also, let $F : \mathbb{G} \times \{0, 1\}^* \to \mathbb{Z}_q^*$. When searching for the access token, we make use of an auxiliary function *Check* that returns true if the decrypted message is correct. We further present the algorithms for the free subscription SETUP attack.

Device's KeyGen(pp): Choose $x_D \xleftarrow{\$} \mathbb{Z}_q^*$ and $p \xleftarrow{\$} [0, n]$. Output the device's secret key $sk_D = (x_D, p)$.

Encryption Sessions: The possible encryption sessions performed by D are described below. Let $i \neq p$.

Encryption$_i(m_i, pk, sk_D)$: To encrypt a message $m_i \in \mathbb{G}$, first compute $k_i \leftarrow F(g^{x_D}, i)$. Then compute the values $c_i \leftarrow g^{k_i}$ and $d_i \leftarrow m_i \cdot y^{k_i}$. Output the pair (c_i, d_i).

Encryption$_p(m_p, pk, sk_D)$: To encrypt a message $m_p \in \mathbb{G}$, compute the values $c_p \leftarrow g^{x_D}$ and $d_p \leftarrow m_p \cdot y^{x_D}$. Output the pair (c_p, d_p). Erase p from D's memory.

Token$(c_1, d_1, \ldots, c_n, d_n, pk)$: Let $i = 1$. Compute $k_{i+1} \leftarrow F(c_i, i \bmod n + 1)$, $m_{i+1} \leftarrow d_{i+1} \cdot y^{-k_{i+1}}$ and $i \leftarrow i + 1$, until $Check(m_i) = \texttt{true}$. Output the token p.

The ith Extract(c_i, d_i, p): To recover the ith message compute $k_i \leftarrow F(c_p, i)$ and $m_i \leftarrow d_i \cdot y^{-k_i}$.

Remark 5.19. It is easy to see that message m_p can only be retrieved by the recipient.

5.4.2.2 Security Analysis

We further state the security margin without proof due to its similarity to the more involved proof of Theorem 5.8.

Theorem 5.7. If F is a PRF and $i \in [1, p-1]$ then EG and FS are IND-SETUP. Formally, let A be an efficient PPT IND-SETUP adversary. There exists an efficient algorithm B such that

$$ADV^{\text{IND-SETUP}}_{\text{EG, FS}}(A) \leqslant 2ADV^{\text{PRF}}_{F}(B).$$

5.4.3 Paid Subscription

In this subsection, we describe a kleptographic analogue of payed television (denoted by PS). Thus, C pays M for a session's access token, that only M can extract from D. Note that these tokens are unique per session. So, a group of users can pay for only one token and all of them will have access to that session's private messages. Although this can be considered cheating, it is also a reality in other systems (*e.g.* paying for a Netflix account and sharing the credentials with one's friends). We will rectify this problem in the next subsection.

5.4.3.1 Description

Let t be a security parameter and $P : \mathbb{G} \times [1, n] \rightarrow [1, n]$. After the first message is transmitted the manufacturer will send the clients a set of t positions p_j needed to compute the access token. Note that M has a window of at least $t-1$ messages to receive his payments. If one payment is declined, M can deactivate the backdoor before the t-th message has been issued. A downside of this scheme is that if one of the clients fails to pay for the token, then he deprives all users of their access.

We further state one session of the protocol. After a predetermined number of messages (greater than n) have elapsed, D can generate new keys and start a new session.

Manufacturer's KeyGen(pp): Choose $x_M \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $y_M \leftarrow g^{x_M}$. Output the manufacturer's public key $pk_M = y_M$. The secret key is $sk_M = x_M$. Store pk_M in D's internal memory.

Device's KeyGen(pp): Choose $k_0 \xleftarrow{\$} \mathbb{Z}_q^*$. For each $j \in [1,t]$ compute $p_j \leftarrow P(y_M^{k_0}, j)$ and choose $x_j \xleftarrow{\$} \mathbb{Z}_q^*$. Compute $x_D \leftarrow x_1 + \ldots + x_t$. Store the device's secret key $sk_D = (k_0, p_1, \ldots, p_t, x_1, \ldots, x_t, x_D)$.

Encryption Sessions: The possible encryption sessions performed by D are described below. Let $i \in [0,n]$ and $i \neq p_j$, for each $j \in [1,t]$. The algorithm for *Encryption*$_i$ are identical to the public subscription and thus are omitted.

Encryption$_0(m_0, pk)$: To encrypt a message $m_0 \in \mathbb{G}$ compute the values $c_0 \leftarrow g^{k_0}$ and $d_0 \leftarrow m_0 \cdot y^{k_0}$. Output the pair (c_0, d_0). Erase k_0 from D's memory.

Encryption$_{p_j}(m_{p_j}, pk, sk_D)$: To encrypt a message $m_{p_j} \in \mathbb{G}$, compute the values $c_{p_j} \leftarrow g^{x_j}$ and $d_{p_j} \leftarrow m_{p_j} \cdot y^{x_j}$. Output the pair (c_{p_j}, d_{p_j}). Erase (p_j, x_j) from D's memory.

Token(c_0, sk_M): For each $j \in [1,t]$ compute $p_j \leftarrow P(c_0^{x_M}, j)$. Output the token $p = (p_1, \cdots, p_t)$.

The ith Extract(c_i, d_i, p): To recover the ith message compute $c_p \leftarrow c_{p_1} \cdot \ldots \cdot c_{p_t}$ and $k_i \leftarrow F(c_p, i)$ and $m_i \leftarrow d_i \cdot y^{-k_i}$.

Remark 5.20. It is easy to see that messages $m_0, m_{p_1}, \ldots, m_{p_t}$ can not be retrieved by the customers.

5.4.3.2 Security Analysis

Theorem 5.8. If DDH is hard in $\mathbb{G}$, P is a PRP, F is a PRF and $(C_n^t)^{-1}$ is negligible then EG and PS are IND-SETUP. Formally, let A be an efficient PPT IND-SETUP adversary. There exist three efficient algorithms B_1, B_2 and B_3 such that

$$ADV^{\text{IND-SETUP}}_{\text{EG, PS}}(A) \leqslant 2ADV^{\text{DDH}}_{\mathbb{G},g}(B_1) + 2ADV^{\text{PRP}}_{P}(B_2) + 2ADV^{\text{PRF}}_{F}(B_3) + (C_t^n)^{-1}.$$

Proof. Let A be an IND-SETUP adversary trying to distinguish between EG and PS. We show that A's advantage is negligible. We construct the proof as a sequence of games in which all the required changes are applied to PS. Let W_i be the event that A wins game i.

Game 0. The first game is identical to the IND-SETUP game[25]. Thus, we have

$$|2Pr[W_0] - 1| = ADV^{\text{IND-SETUP}}_{\text{EG,PS}}(A). \tag{5.10}$$

Game 1. In this game, instead of using $y_M^{k_0}$ as a key to P we use $r_P \xleftarrow{\$} \mathbb{G}$. More precisely, for each $j \in [1,t]$ we compute $p_j \leftarrow P(r_P, j)$. Since this is the only change between *Game 0* and *Game 1*, A will not notice the difference assuming the DDH assumption holds. Formally, this means that there exists an algorithm B_1 such that

$$|Pr[W_0] - Pr[W_1]| = ADV^{\text{DDH}}_{\mathbb{G},g}(B_1). \tag{5.11}$$

Game 2. Since P is a PRP then we can choose $p_j \xleftarrow{\$} [1,n]$, without A detecting the change. Formally, this means that there exists an algorithm B_2 such that

$$|Pr[W_1] - Pr[W_2]| = ADV^{\text{PRP}}_{P}(B_2). \tag{5.12}$$

Game 3. In each $Encryption_{p_j}$ algorithm we make the change $c_{p_j} \leftarrow g^{k_j}$ and $d_{p_j} \leftarrow m_{p_j} y^{k_j}$, where $k_j \xleftarrow{\$} \mathbb{Z}_q^*$. Since k_js and x_js have the same distribution, and the b_js are uniformly distributed in $[1,n]$, then A can only detect the change using a brute-force attack[26]. Formally, we have

$$|Pr[W_2] - Pr[W_3]| = (C_t^n)^{-1}. \tag{5.13}$$

Game 4. The last change we make is $k_i \xleftarrow{\$} \mathbb{Z}_q^*$. Adversary A will not notice the difference, since F is a PRF. Formally, this means that there exists an algorithm B_3 such that

$$|Pr[W_3] - Pr[W_4]| = ADV^{\text{PRF}}_{P}(B_3). \tag{5.14}$$

The changes made to PS in *Game 1* $-$ *Game 4* transformed it into EG. Thus, we have

$$Pr[W_4] = 1/2. \tag{5.15}$$

Finally, the statement is proven by combining the equalities (5.10) – (5.15).

□

[25] as in Remark 5.2

[26] *i.e.* by trying each t-combination c_{try} of c_is, until on input c_{try} the *Extract* algorithm outputs a message m such that $Check(m) = \texttt{true}$.

5.4.4 Targeted Subscription

As mentioned in the previous subsection, a coalition of clients can pay for only one token[27]. To avoid this problem we bind a specific session to a certain client. We could not find a method that allows multiple bindings per session. We further present the proposed solution for binding users and sessions (denoted by TS).

5.4.4.1 Description

Customer's KeyGen(pp): Choose $x_C \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $y_C \leftarrow g^{x_C}$. Output the customer's public key $pk_C = y_C$. The secret key is $sk_C = x_C$. Store pk_C in D's internal memory.

Encryption Sessions: The possible encryption sessions performed by D are described below. Let $i \in [0, n]$ and $i \neq p_j$, for each $j \in [1, t]$. The algorithms for $Encryption_0$ and $Encryption_{p_j}$ are identical to the paid subscription and thus are omitted.

$Encryption_i(m_i, pk, pk_C, sk_D)$: To encrypt a message $m_i \in \mathbb{G}$, first compute $k_i \leftarrow F(y_C^{x_D}, i)$. Then compute the values $c_i \leftarrow g^{k_i}$ and $d_i \leftarrow m_i \cdot y^{k_i}$. Output the pair (c_i, d_i).

The ith Extract(c_i, d_i, p): To recover the ith message compute $c_p \leftarrow c_{p_1} \cdot \ldots \cdot c_{p_t}$ and $k_i \leftarrow F(c_p^{x_C}, i)$ and $m_i \leftarrow d_i \cdot y^{-k_i}$.

5.4.4.2 Security Analysis

Theorem 5.8 assures us that the client has negligible probability of reading V's messages without M's help. We further prove a similar result for any PPT IND-SETUP adversaries.

Theorem 5.9. If DDH is hard in $\mathbb{G}$, P is a PRP and F is a PRF then EG and TS are IND-SETUP. Formally, let A be an efficient PPT IND-SETUP adversary. There exist three efficient algorithms B_1, B_2 and B_3 such that

$$ADV^{\text{IND-SETUP}}_{\text{EG, TS}}(A) \leqslant 4ADV^{\text{DDH}}_{\mathbb{G},g}(B_1) + 2ADV^{\text{PRP}}_{P}(B_2) + 2ADV^{\text{PRF}}_{F}(B_3).$$

Proof. *Game 0* $-$ *Game 2* and *Game 4* are identical to the games presented in the proof of Theorem 5.8 and thus, are omitted. Since only the customer is in position of x_C, we can not use the strategy presented in Theorem 5.8, *Game 3*. Thus, we present a modified version of *Game 3*.

[27] further used by the whole group to access messages

Game 3'. In this game, we replace $y_C^{x_D}$ by $r_F \xleftarrow{\$} \mathbb{Z}_q^*$. Due to the fact that DDH is hard in $\mathbb{G}$, A will not notice the change. Formally, this means that there exists an algorithm B_1' such that

$$|Pr[W_2] - Pr[W_{3'}]| = ADV^{\text{DDH}}_{\mathbb{G},g}(B_1'). \tag{5.16}$$

Finally, the statement is proven by combining the equalities (5.10) – (5.12) and (5.14) – (5.16).

□

5.4.5 Future Work

A couple of interesting open problems are the extension of subscription based services to digital signatures and the implementation of multi-targeted subscriptions for one session.

5.5 Hash Channels

Most subliminal channels or SETUP attacks use random numbers to convey information undetected. In consequence, all the proposed countermeasures focus on sanitizing the random numbers used by a system. In the case of digital signatures, a different but laborious method for inserting a subliminal channel in a system is presented in [256]. Instead of using random numbers as information carriers, *Alice* uses the hash of the message to convey the message for *Bob*. In order to achieve this, *Alice* makes small changes to the message until the hash has the desired properties. Note that the method presented in [256] bypasses all the countermeasures mentioned so far.

This section studies a generic method that allows the prisoners to communicate through the subliminal-free signatures found in [214, 215, 73, 135, 32, 57]. To achieve our goal we work in a scenario where all messages are time-stamped before signing. Note that we do not break any of the assumptions made by the subversion-free proposals. This work is motivated by the fact that most end-users to do not verify the claims made by manufacturers[28]. Moreover, users often do not know which should be the outputs of a device [163]. A notable incident in which users where not aware of the correct outputs and trusted the developers is the Debian incident [50].

[28] Manufacturers might implement subversion-free signatures just for marketing purposes, while still backdooring some of the devices produced.

Conventions. The encryption of a message $m \in \{0,1\}$ using one-time pad is denoted by $\omega \leftarrow m \oplus b$, where b is a random bit used only once.

We consider that the covert channels presented from now on are implemented in a device D that digitally signs messages. In the case of subliminal channels, the prisoners are denoted as *Alice* (sender) and *Bob* (receiver), while *Walter* is the guard. In the case of SETUP attacks, the owner of the device is referred to as *Charlie* and the attacker is usually *Mallory*. When the secret key sk is not known to the PPT algorithm A we assume that sk is stored only in D's volatile memory. Note that *Walter* and *Charlie* believe that D signs messages using the original specifications of the signature scheme implemented in D. When one of the original signature's algorithm is not modified by the covert channel, the algorithm will be omitted when presenting the respective channel.

5.5.1 A Schnorr based Hash Channel

In order to be valid, legal documents need a timestamp appended to them before being digitally signed [26, 132]. According to [26] the timestamp must include seconds. Note that if the timestamp module is independent from the *Sign* module, then *Walter* or *Charlie* can inject false timestamps into the signing module. Thus, we assume that the timestamp module is integrated in the signing module. Using this framework we achieve a subliminal channel by adapting and simplifying the idea from [256]. Note that we embed our proposal in the Schnorr signature.

Let lim be an upper limit for the number of trials and u_t the smallest time unit used by the time stamping algorithm (*e.g.* seconds, milliseconds). We further present our proposed subliminal channel.

Time Stamp(u_t): Output the current time τ including u_t.

Subliminal Sign(m, ω, sk): Generate a random number $k \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $r \leftarrow g^k$. Let $counter = 1$. Generate τ using the *Time Stamp* algorithm and compute $e \leftarrow h(r\|m\|\tau)$ and $counter = counter + 1$, until $e \equiv \omega \bmod 2$ or $counter = lim$. Compute $s \leftarrow k - xe \bmod q$. Output the signature (e, s).

Extract(m, e, s, pk) : To extract the embedded message compute $\omega \leftarrow e \bmod 2$. Remark that the probability of event $e \equiv \omega \bmod 2$ is $1 - 1/2^{lim}$.

The security of the Schnorr signature scheme is preserved, since we are not modifying the scheme itself, but the way messages are processed. Let τ_h be the average time it takes device D to compute $h(r\|m\|\tau)$ for fixed bit-size bit_m messages. To avoid detection

by *Walter* or *Charlie* the manufacturer writes in D's specification that for a message of size bit_m it takes $lim \cdot \tau$ to sign bit_m messages. Thus, D remains consistent with the specifications (*i.e.* IND-COVERT secure). The main restriction when choosing lim is users' usability. Due to the hash-rate statistics reported for SHA-256 in [12, 14] we can assume $\tau_h < 1$ second. Thus, the bottleneck becomes the time stamp (*i.e.* D can not output a signature for time t at time $t-1$). This can be mitigated by including finer time units into the timestamp (*e.g* milliseconds).

Remark 5.21. Let $z \leftarrow g^t$ be the public key of *Bob* and b a bit *Alice* wants to send to *Bob*. Then, we can easily transition to a public key subliminal channel by using the HKE protocol and computing $\omega \leftarrow b \oplus H(z^k)$, where $\mathbb{G}_m = \{0,1\}$. Since k is fresh for each signature, *Alice* can continuously leak data to *Bob*. Note that we are using HKE to encrypt the message so, ω is indistinguishable from a random bit. Thus, the scheme is IND-COVERT secure under the HDH assumption. We further denote this public key subliminal channel by hash_p.

Remark 5.22. When dealing with longer messages there is a simpler way to transmit them. Thus, let m_i be the ith bit of m and $\mathbb{G}_m = \{0,1\}^{|m|}$. The device D can leak m to *Bob* through $|m|$ signing sessions by computing $c \leftarrow m \oplus H(z^{k_0})$ and setting $\omega \leftarrow c_i$ for the ith signing session, where $0 \leqslant i < |m|$. Note that m is successfully transmitted with a probability of $(1 - 1/2^{lim})^{|m|}$. This channel is further denoted by hash_ℓ. Remark that if we replace *Bob* with *Mallory* and set $m \leftarrow x$, hash_ℓ is transformed into a SETUP attack.

Remark 5.23. If adversary A has access to x, then he can compute all k numbers. Thus, hash_p and hash_ℓ can be detected if x can be retrieved from D, while the regular hash channel it is not. Hence, in the public key setting we assume that x is only stored in D's volatile memory[29].

5.5.2 Stochastic Detection

In [161], the authors show that the execution time of the Young-Yung attack can be used to distinguish honest devices from backdoored devices. Using Kucner *et. al.* observations as a starting point, we run a series of experiments to see if our proposed methods can be detected by measuring their execution time.

We implemented in C using the GMP library [19] the Schnorr signature (normal), the trivial channel (trivial), the Young-Yung attack (yy), the hash channel (hash), the public key hash channel (hash_p) and hash_p's extension to long messages (hash_ℓ). The programs

[29] The same assumption is make in Young-Yung's attack, since their mechanism can also be detected when x is known to the attacker.

were run on a CPU Intel i7-4790 4.00 GHz and compiled with GCC with the O3 flag activated. In our experiments for each prime size of 2048, 3072, 4096 and 8192 bits, we ran the algorithms with 100 safe prime numbers from [16]. For each prime we measured the average running time for 128 random 2040 byte messages[30] using the function *omp_get_wtime*() [15]. Before signing each message we added a 8 byte timestamp with the current system time in milliseconds (*clock_gettime*()). The hash function used internally by the algorithms is either SHA256 or SHA512 [11].

When we implemented the hash channels we took advantage of the Merkle-Damgard structure of SHA256 or SHA512. Thus, we computed and stored the intermediary value h_{it} obtained after processing 1984 (SHA256) or 1920 (SHA512) bytes. Then for each trial we used h_{it} to process the last block of the message. Note that the size of the messages was selected such that after h_{it} the SHA functions must process one full message block and a full padding block (worst case scenario). Also, in our experiments *lim* tends towards infinity.

The results of our experiments are presented in Figures 5.3 to 5.10. We can see from the plots that the Schnorr signature and the hash channel have similar execution times. We further investigated this by computing the absolute time difference between a normal execution (t_{n_1}) and a hash channel execution (t_h) or another normal execution (t_{n_2}). The results are presented in Table 5.1. Note that the empirical evidence suggests that the normal and hash channel executions are indistinguishable due to the noise added by the operating system.

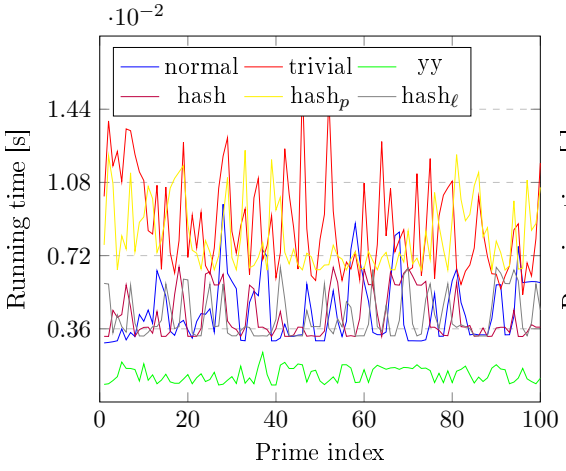


FIGURE 5.3: Prime's size 2048 bits with SHA256.

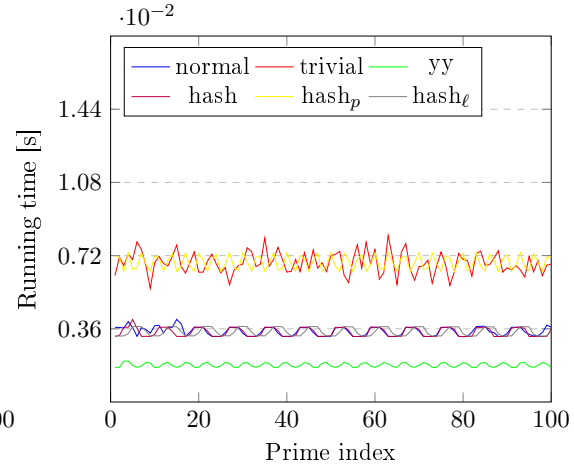


FIGURE 5.4: Prime's size 2048 bits with SHA512.

When we implemented hash_ℓ we distributed the HKE protocol execution over 128 Schnorr signature computations. The downside of this method is that first we need to use 128

[30] By choosing 128 messages we simulated the following scenario: the secret key x is generated using a PRNG with a seed of 128 bits and D leaks the seed.

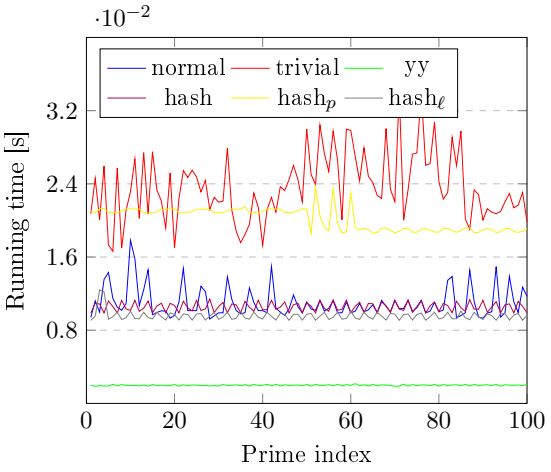


FIGURE 5.5: Prime's size 3072 bits with SHA256.

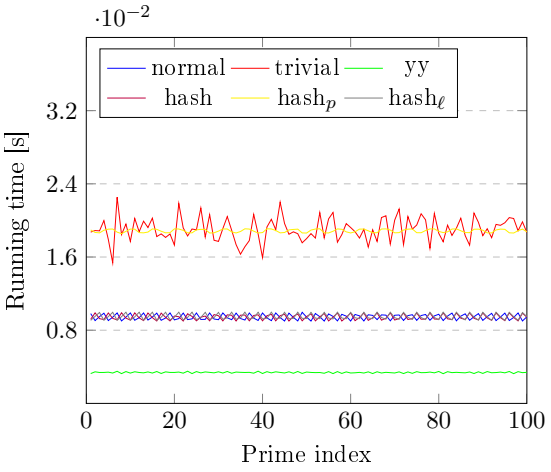


FIGURE 5.6: Prime's size 3072 bits with SHA512.

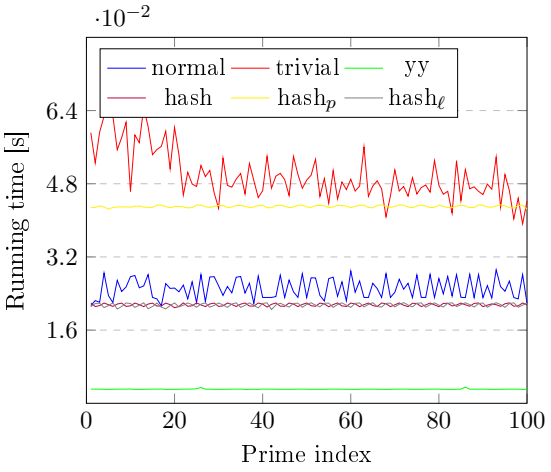


FIGURE 5.7: Prime's size 4096 bits with SHA256.

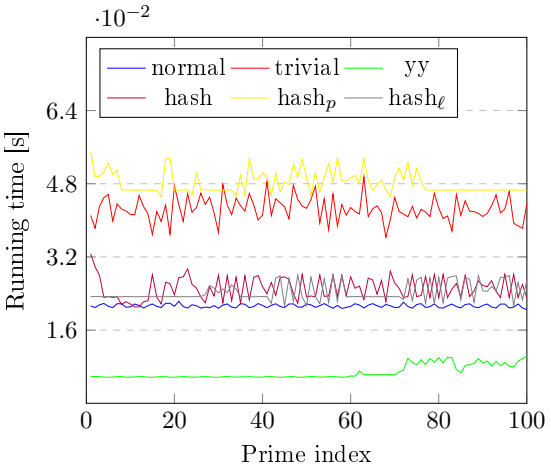


FIGURE 5.8: Prime's size 4096 bits with SHA512.

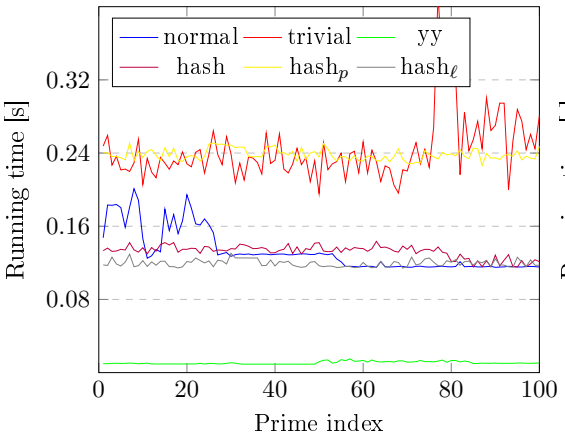


FIGURE 5.9: Prime's size 8192 bits with SHA256.

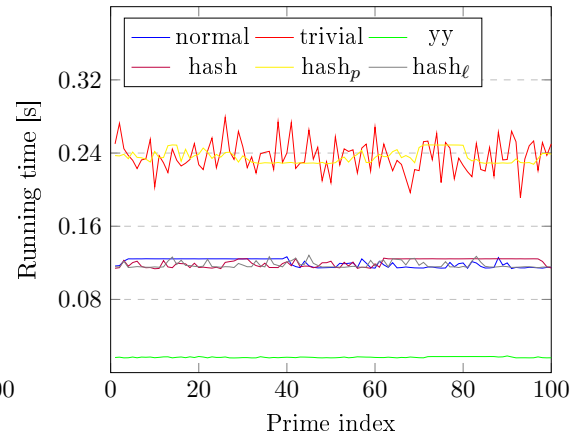


FIGURE 5.10: Prime's size 8192 bits with SHA512.

Prime's size	SHA	$\lvert t_{n_1} - t_{n_2}\rvert$	$\lvert t_{n_1} - t_h\rvert$	$\lvert t_{n_1} - t_\ell\rvert$
2048	256	0.185621	0.138921	0.149650
	512	0.122050	0.097460	0.101445
3072	256	0.462406	0.105996	0.150646
	512	0.156232	0.156667	0.160375
4096	256	0.523229	0.354953	0.358049
	512	0.118666	0.134868	0.085795
8192	256	1.381028	1.548863	1.586020
	512	0.483661	0.629464	0.468742

TABLE 5.1: Time comparison.

Schnorr signatures for masking the HKE and then 128 hash channel signatures for leaking the message. The results presented in this section are only for the first part, since experimental data for hash channels is already presented. Note that the first part of hash_ℓ is indistinguishable from the normal execution. As in the case of the hash channel, we further investigated the indistinguishability claim by computing the absolute time difference between a normal execution (t_{n_1}) and a hash_ℓ channel execution (t_ℓ). The results are presented in Table 5.1. Note that the empirical evidence suggests that the normal and hash_ℓ channel executions are indistinguishable due to the noise added by the operating system.

Another remark is that the rest of the channels can be easily detected by measuring their execution time. Thus, noise must be added to the Young-Yung attack or to the Schnorr signature in order to make the subliminal channels undetectable. Note that the trivial channel and the public key hash channel have similar execution times. Thus, any technique used to mask the execution time of the trivial channel can also be used for the public key hash channel.

Let T be the computation time for one signature. We denote by $E[T]$ and $\sigma[T]$ the expected value and the standard deviation of T. Kucner *et. al.* introduce the $R[T] = \sigma[T]/E[T]$ characteristic in order to measure computation time independently of the actual speed of the processor. We computed $R[T]$ for all channels and the results are presented in Table 5.2. We can observe from our experiments that the $R[T]$ characteristic fluctuates in practice. Also, from Table 5.2, it is easy to observe that the $R[T]$ characteristic for the Schnorr signature is always smaller than the one for the trivial channel and the Young-Yung attack. Thus, we can distinguish these two channels from an honest execution. Unfortunately, the results are inconclusive for the rest of the channels.

Prime's size	SHA	normal	trivial	yy	hash	hash_p	hash_ℓ
2048	256	0.128196	0.722334	0.322048	0.076789	0.138364	0.147544
	512	0.035308	0.695684	0.131866	0.017010	0.033390	0.094461
3072	256	0.094644	0.751065	0.430531	0.044940	0.063633	0.095584
	512	0.024800	0.718313	0.207609	0.024917	0.044754	0.092278
4096	256	0.131263	0.700937	0.582434	0.043187	0.045583	0.101174
	512	0.051705	0.691449	0.326993	0.125705	0.089238	0.140214
8192	256	0.116920	0.704363	1.156188	0.172762	0.101328	0.132343
	512	0.056456	0.708207	0.594154	0.057846	0.086518	0.121710

TABLE 5.2: $R[T]$ characteristic.

5.5.3 Marketing Backdoors

In this section we provide the reader with state-of-the-art countermeasures used to obtain subliminal-free signatures and show that three proposals are vulnerable to the hash and hash_ℓ channels without masking the channels' execution time, while for the rest the channels must be masked. Thus, a manufacturer can market a product as being subliminal free[31], while in reality it is not. Note that our proposed scenario does not violate the assumptions made by the subversion-free protocols.

5.5.3.1 Russel *et al.* Subversion-Free Proposal

The authors of [214, 215] assume that all the random numbers used by a signature scheme are generated by a malicious RNG (including the key generation step). Based on this assumption, the authors describe and prove secure a generic method for protecting users. Note that both the trivial channel and the Young-Yung attack can be modeled as malicious RNGs. Unfortunately, in the hash channel scenario the security of their method breaks down.

The philosophy behind Russel *et al.* method is to split every generation algorithm into two parts: a random string generation part RG and a deterministic part DG. By extensively testing DG the user can be ensured that the deterministic part is *almost consistent* with the specifications. By using two independent RNG modules $Source_1, Source_2$ and hashing their concatenated outputs, any backdoors implemented in the RNGs will not propagate into DG. We further describe an instantiation of [215] using the Schnorr signature scheme.

$Random(Source_1, Source_2)$: Generate $s_1 \xleftarrow{\$} Source_1$ and $s_2 \xleftarrow{\$} Source_2$. Output $h(s_1 \| s_2)$.

[31] by implementing one of these countermeasures

KeyGen(pp): Generate x using the *Random* algorithm and compute $y \leftarrow g^x$. Output the public key $pk = y$. The secret key is $sk = x$.

Sign(m, sk): To sign a message $m \in \{0,1\}^*$, first generate k using the *Random* algorithm. Then, compute the values $r \leftarrow g^k$, $e \leftarrow h(r\|m)$ and $s \leftarrow k - xe \bmod q$. Output the signature (e, s).

5.5.3.2 Hanzlik *et al.* Controlled Randomness Proposal

A method for controlling the quality of k is proposed in [135]. In order to do this, the authors use a blinding factor $U \leftarrow g^u$ that is installed by the owner of the device and a counter i. The owner accepts a signature produced by D if and only if *Check* returns `true`. Note that the Young-Yung SETUP attack is not possible due to the blinding factor. We further present their modifications on the *Sign* algorithm.

Sign(m, U, i, sk): To sign a message $m \in \{0,1\}^*$, first generate $k_0 \xleftarrow{\$} \mathbb{Z}_q^*$, compute $r' \leftarrow g^{k_0}$, $k_1 \leftarrow H(U^{k_0}, i)$ and increment i. Let $k \leftarrow k_0 k_1$. Compute the values $r \leftarrow g^k$, $e \leftarrow h(r\|m)$ and $s \leftarrow k - xe \bmod q$. Output the signature (e, s) and the control data (r', i).

Check(e, s, r', u): Compute $r \leftarrow g^s y^e$ and $\alpha \leftarrow H(r'^u, i)$. Output `true` only if and only if $r = r'^\alpha$. Otherwise, output `false`.

The authors underline that a subliminal channel[32] exists, but due to the limited memory of the signing device, hiding the time needed to implement their proposed channel is difficult. Note that our timestamp method proposed in Section 5.5.1 is much faster[33] and, thus, in some cases is feasible for bypassing Hanzlik *et al.* mechanism.

5.5.3.3 Choi *et al.* Tamper-Evident Digital Signatures

Choi *et al.* [73] introduce the notion of tamper-evidence for digital signatures in order to prevent corrupted nodes to covertly leak secret information. We further provide the tamper-evident Schnorr signature.

ParamGen(λ): Generate two large prime numbers p, q, such that $q \geqslant 2^\lambda$ and $q|p-1$. Select a cyclic group $\mathbb{G}$ of order $p-1$ and let $g \in \mathbb{G}$ be an element of order q. Let $h : \{0,1\}^* \to \mathbb{Z}_q^*$ be a hash function and let ℓ be the number of permitted signatures. Output the public parameters $pp = (p, q, g, \mathbb{G}, h, \ell)$.

[32] similar to the trivial channel described in Section 5.1

[33] *i.e.* computing a hash is faster than computing a modular exponentiation

KeyGen(pp, κ): Choose $x \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $y \leftarrow g^x$. Also, choose $\omega_\ell \xleftarrow{\$} \{0,1\}^\kappa$. For $1 \leqslant i \leqslant \ell$, generate $k_i \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $\omega_{i-1} \leftarrow h(g^{k_i} \| \omega_i)$. Output the public key $pk = (y, \omega_0)$. The secret key is $sk = (x, k_1, \ldots, k_\ell, \omega_1, \ldots, \omega_\ell)$.

Sign(m_i, sk): To sign the ith message $m_i \in \{0,1\}^*$, compute the values $r_i \leftarrow g^{k_i}$, $e_i \leftarrow h(r_i \| \omega_i \| m_i)$ and $s_i \leftarrow k_i - xe_i \bmod q$. Output the signature (e_i, s_i, ω_i).

Verification$(m_i, e_i, s_i, \omega_i, pk)$: To verify the signature (e_i, s_i, ω_i) of message m_i, compute $r_i \leftarrow g^{s_i} y^{e_i}$ and $u \leftarrow h(r_i \| \omega_i \| m_i)$. Output `true` if and only if $u = e_i$ and $\omega_{i-1} \leftarrow h(r_i \| \omega_i)$. Otherwise, output `false`.

The authors work in the honest key generation model. Thus, the nodes can not manipulate the k_is in any way. Fortunately, our hash channels use messages to leak confidential data. Thus, the nodes can still subliminally transmit data by using our proposed channels.

5.5.3.4 Ateniese *et al.* and Bohli *et al.* Subversion-Free proposals

The authors of [32] propose the usage of re-randomizable signatures and unique signatures as countermeasures to backdoors induced by malicious RNGs. These proposals are secure according to their security model [32]. A similar approach can be found in [57], where the authors convert the Digital Signature Algorithm into a deterministic signature. Note that both approaches assume honest key generation.

All these signature schemes work on fixed length messages and internally use a number theoretic hash function[34]. In order to work on variable length messages a standard hash function is used to process the message and the resulting hash is used as input for the Naor-Reingold function. Thus, for each small change in the message we have to recompute the hash $h(m)$, multiply $|h(m)/2|$ integers from $\mathbb{Z}_q^*$ and perform an exponentiation in $\mathbb{G}$. So, our proposed hash channel on average doubles the time necessary to process a message. In this case, the execution time of a hash channel is no longer similar to an honest implementation and, thus, noise must be added to mask the backdoor.

[34] more precisely, the Naor-Reingold pseudo-random function [192, 193]

Chapter 6

(Pseudo-)Random Number Generators

One of the most essential building blocks of cryptography are random numbers generators. In particular, for ensuring privacy or authenticity is vital that cryptographic keys are randomly generated. Additionally, most cryptographic algorithms are randomized.

Generating random numbers by means of physical processes is usually time consuming and expensive, thus in practice most applications use pseudo-random numbers generators. Such a generator is a deterministic algorithm that takes as input a small random seed and expands it into a much longer sequence of bits. Not all PRNGs are suitable for cryptographic application. One such example is the generator used by Adobe Flash Player. Some of the basic PRNG security requirements are: not to be able to distinguish it from a real RNG and not to be able to recover its internal state from its output. We describe a seed recovering algorithm for the Flash Player PRNG in the first part of this chapter.

A popular method for generating cryptographic keys or other random inputs is to have an entropy pool that accumulates data from a physical noise source and a PRNG that periodically reseeds from the pool and outputs data at a constant rate. To ensure proper operation, before adding data to the entropy pool some lightweight tests are applied to it. In the second part of this chapter we study a possible architecture for adding data to the pool. Therefore, we provide the reader with experimental results and the theoretical framework for our proposed architecture.

6.1 Flash Player PRNG

JIT compilers (*e.g.* JavaScript and ActionScript) translate source code or bytecode into machine code at runtime for faster execution. Due to the fact that the purpose of JIT compilers is to produce executable data, they are normally exempt from data execution prevention (DEP[1]). Thus, a vulnerability in a JIT compiler might lead to an exploit undetectable by DEP. One such attack, called JIT spraying, was proposed in [56]. By coercing the ActionScript JIT engine, Blazakis shows how to write shellcode into the executable memory and thus, bypass DEP. The key insight is that the JIT compiler is predictable and must copy some constants to the executable page. Hence, these constants can encode small instructions and then control flow to the next constant's location.

To defend against JIT spraying attacks, Adobe employs a technique called *constant blinding*. This method prevents an attacker from loading his instructions into constants and thus, blocks the delivery of his malicious script. The idea behind constant blinding is to avoid storing constants in memory in their original form. Instead, they are first XORed with some randomly generated secret cookie and then stored inside the memory. If the secret cookie is generated by means of a weak PRNG[2], the attacker regains his ability to inject malicious instructions.

Instead of using an already proven secure PRNG, the Flash Player designers tried to implement their own PRNG. Unfortunately, in [253, 1] it is shown that the design of the generator is flawed. In [1] a brute force attack is implemented, while in [253] a refined brute force attack is presented. These results have been reported to Adobe under the code CVE-2017-3000 [21] and the vulnerability has been patched in version 25.0.0.127.

In this section, we refine the attack presented in [253] from a time complexity of $\mathcal{O}(2^{21})$ to one of $\mathcal{O}(2^{11})$. We also show that no matter the parameters used by the PRNG, the flaw remains. More precisely we show that for any parameters the worst brute force attack takes $\mathcal{O}(2^{21})$ operations. In [253] the authors do not present the full algorithm for reversing the PRNG, while in [1] we found the full algorithm, but it was not optimized. For completeness, in Appendix K we also present an optimized version of the full algorithm. Note that in this section we only focus on the Flash Player PRNG. For more details about JIT spraying attacks and constant blinding we refer the reader to [33, 56, 212, 253].

[1] The DEP mechanism performs additional checks on memory to help prevent malicious code from running on a system.

[2] *i.e.*, the seed used to generate the cookie can be recovered in reasonable time

6.1.1 Preliminaries

6.1.1.1 Constant Blinding in Flash Player

In this subsection we describe the implementation of the Flash Player PRNG, as presented in [17]. The generator has four components (described in Listing 6.1): a seed initialization function (*RandomFastInit*), a seed update function (*RandomFastInit*), a hash function (*RandomPureHasher*) and a cookie generation function (*GenerateRandomNumber*). According to the source code, the hash function is adapted from [254]. Note that the variable uValue is initialized by a function found in the Windows API (*VMPI_ getPerformanceCounter*).

The role of the hash function is to make attackers unable to retrieve the seed value (uValue) in reasonable time. Note that the default timeout in Flash Player is $15s$. Thus, an attacker must succeed in finding the seed, predicate the secret value into the next round and embed the desired value in the executable heap in $15s$.

6.1.1.2 Shifting Signed Integers

According to [10], if we left shift a signed integer (*e.g.* iSeed) the result is unpredictable and if we right shift a signed negative integer the result is implementation dependent. Thus, we will make a clear distinction between implementation independent or dependent attack strategies against the Flash Player PRNG. In some cases, the attacks devised for a particular implementation are faster than the corresponding implementation independent strategy (see Section 6.1.3).

For simplicity, when talking about targeted attacks we consider the behavior of shifts implemented in Microsoft Visual Studio [10] and GCC [20] on x86 and x64 architectures. Thus, left shifts are sign independent (*e.g.* `0b11000000` $\ll 1 =$ `0b10000000`) and right shifts of signed integers use the sign bit to fill vacated bit positions (*e.g.* `0b11000000` $\gg_s 1 =$ `0b11100000` and `0b01000000` $\gg_s 1 =$ `0b00100000`).

6.1.1.3 Previous Cryptanalysis Results

By abstracting the code described in Listing 6.1, we identify the three main components of the cookie generation function, *i.e.*:

$$f(x) = (x \ll 13) \oplus x - (x \gg_s 21),$$
$$g(x) = (c_3 \cdot x^3 + c_2 \cdot x + c_1) \ \& \ \texttt{0x7fffffff} + x,$$
$$h(x) = 71 \cdot x \bmod 2^{32}.$$

If these functions are reversed, then the PRNG is broken. In [253], the authors propose an algorithm for reversing f (Algorithm 41) and a backtracking algorithm for reversing g (the complete description is presented in Algorithm 59). For completeness, we provide in Appendix K the full algorithm (Algorithm 60) for reversing the PRNG (which includes the inverse of h). Note that Algorithm 41 has a time complexity of $\mathcal{O}(2^{21})$ and is implementation independent.

```
#define c3   15731L
#define c2   789221L
#define c1   1376312589L
#define kRandomPureMax 0x7fffffffL

void RandomFastInit(pTRandomFast pRandomFast)
{
    int32_t n = 31;
    pRandomFast->uValue = (uint32_t)(VMPI_getPerformanceCounter());
    pRandomFast->uSequenceLength = (1L << n) - 1L;
    pRandomFast->uXorMask = 0x14000000L;
}

#define RandomFastNext(_pRandomFast)\
(\
    ((_pRandomFast)->uValue & 1L)\
    ? ((_pRandomFast)->uValue = (((_pRandomFast)->uValue >> 1) ^ (
    _pRandomFast)->uXorMask)\
    : ((_pRandomFast)->uValue = (((_pRandomFast)->uValue >> 1))\
)

int32_t RandomPureHasher(int32_t iSeed)
{
    int32_t    iResult;

    iSeed = ((iSeed << 13) ^ iSeed) - (iSeed >> 21);

    iResult = (iSeed*(iSeed*iSeed*c3 + c2) + c1) & kRandomPureMax;
    iResult += iSeed;
```

```
29      iResult = ((iResult << 13) ^ iResult) - (iResult >> 21);
30
31      return iResult;
32  }
33
34  int32_t GenerateRandomNumber(pTRandomFast pRandomFast)
35  {
36      if (pRandomFast->uValue == 0)
37      {
38          RandomFastInit(pRandomFast);
39      }
40      long aNum = RandomFastNext(pRandomFast);
41      aNum = RandomPureHasher(aNum * 71L);
42
43      return aNum & kRandomPureMax;
44  }
```

LISTING 6.1: ActionScript PRNG implementation.

Algorithm 41. The algorithm for reversing f.

Input: The value to reverse v.
Output: The set of possible solutions S.

```
1  S ← ∅;
2  for low ∈ [0, 0x7ff] do
3  |   temp ← v & 0x7ff;
4  |   if temp > low then
5  |   |   high = (1 ≪ 11) + low − temp;
6  |   else
7  |   |   high = low − temp;
8  |   end
9  |   for mid ∈ [0, 0x3ff] do
10 |   |   s ← (high ≪ 21) | (mid ≪ 11) | low;
11 |   |   if f(s) == v then
12 |   |   |   S ← S ∪ s;
13 |   |   end
14 |   end
15 end
16 return S;
```

6.1.2 Reinterpreting

Let n be the word size in bits. As Algorithm 59 can be used to reverse any generic polynomial g and the linear function h can be easily reversed, we only focus on reversing the generic function

$$f(x) = (x \ll \ell) \oplus x - (x \gg_s r),$$

	Conditions	Time complexity
1	$n - r \leqslant \ell$ and $n \leqslant 2r$	$\mathcal{O}(2^r)$
2	$n - r \leqslant \ell$ and $n \geqslant 2r$	$\mathcal{O}(c_{24}2^r)$
3	$n - r \geqslant \ell$ and $n \leqslant 2r$	$\mathcal{O}(2^r)$
4	$n - r \geqslant \ell$ and $n \geqslant 2r$	$\mathcal{O}(c_{24}2^r)$

TABLE 6.1: Attack parameters for Lemma 6.1.

where $1 \leqslant \ell, r \leqslant n$ are integers. We further denote by v the output of $f(x)$.

Degenerate Cases. Let $ct = 10^{n-1} \gg_s n$. We consider the cases $\ell, r \in \{0, n\}$ as degenerate due to different inherent weakness induced by these choices. Thus, in our study we do not take in consideration degenerate cases. We further present the weakness associated with the degenerate cases:

- when $r = n$ and $0 < \ell \leqslant n$, the function $f(x) = x \oplus (x \ll \ell) + ct$ leaks ℓ bits of its seed;
- when $\ell = 0$ and $0 \leqslant r \leqslant n$, the function $f(x) = -(x \gg_s r)$ leaks $n - r$ bits of its seed and v has the rest of the bits constant;
- when $r = 0$ and $0 < \ell \leqslant n$, the function $f(x) = x \oplus (x \ll \ell) - x$ always outputs a v with ℓ trailing zeros;
- when $\ell = n$ and $0 < r \leqslant n$, the function $f(x) = x - (x \gg_s r)$ leaks r bits of its seed.

We further present a series of attacks that are implementation independent. In the case $n - r \leqslant \ell, n \leqslant 2r$ we generalized a different algorithm than Algorithm 41, due to a more direct adaptation to an implementation dependent version.

Lemma 6.1. Let $c_{24} = \lfloor n/r \rfloor + 1$. For each (set of) condition(s) presented in Column 2 of Table 6.1 there exists an attack whose corresponding time complexity is presented in Column 3 of Table 6.1.

Proof. When $n - r \leqslant \ell$, we can explicitly write the function f as shown in Figure 6.1. Note that the bits used to fill vacated positions are represented as question marks. As we want a compiler independent attack we consider the ? bits as unknown and tailor our attacks accordingly.

In the first case, we first recover the most significant $n - r$ bits ($high$) and then extract the least significant $n - r$ bits (low) from $v + high$. For the rest of $2r - n$ bits (mid) we do an exhaustive search. This leads to a time complexity of $\mathcal{O}(2^{n-r}2^{2r-n}) = \mathcal{O}(2^r)$.

$$\begin{array}{cccccccccc} & a_1 & \dots & a_{n-\ell} & a_{n-\ell+1} & \dots & a_r & a_{r+1} & \dots & a_n \\ \oplus & a_{\ell+1} & \dots & a_n & 0 & \dots & 0 & 0 & \dots & 0 \\ = & t_1 & \dots & t_{n-\ell} & t_{n-\ell+1} & \dots & t_r & t_{r+1} & \dots & t_n \\ + & ? & \dots & ? & ? & \dots & ? & a_1 & \dots & a_{n-r} \end{array}$$

FIGURE 6.1: Bit representation of $f(x)$.

Algorithm 42. The algorithm for reversing f (Case 1).

Input: The value to reverse v.
Output: The set of possible solutions S.

```
1  S ← ∅;
2  for high ∈ [0, 1^{n−r}] do
3  |   temp ← v + high;
4  |   low ← temp & 1^{n−r};
5  |   for mid ∈ [0, 1^{2r−n}] do
6  |   |   s ← (high ≪ r) | (mid ≪ (n − r)) | low;
7  |   |   if f(s) == v then
8  |   |   |   S ← S ∪ s;
9  |   |   end
10 |   end
11 end
12 return S;
```

In the second case, we can do better than simply using Algorithm 42. We first recover the least significant r bits (low) and then use low to gradually recover the rest of the bits (mid). This leads to the complexity $\mathcal{O}(2^r(q+1))$.

When $n-r \geqslant \ell$, we can explicitly write the function f as depicted in Figure 6.2. Note that some of the bits resulted from the left shift overlap with some from the right shift. Thus, in the third case we recover the least significant $n-r$ bits (low), add the overlapping bits, and then recover the most significant $n-r$ bits ($high$) from v. For the rest of $2r-n$ bits (mid) we do an exhaustive search. So, similarly to the first case, we obtain a complexity of $\mathcal{O}(2^r)$.

In the last case, we slightly modify the algorithm used in the second case to take into account the overlapping bits. Thus, the resulting attack has the same complexity $\mathcal{O}(2^r(q+1))$.

$$\begin{array}{cccccccccc} & a_1 & \dots & a_r & a_{r+1} & \dots & a_{n-\ell} & a_{n-\ell+1} & \dots & a_n \\ \oplus & a_{\ell+1} & \dots & a_{\ell+r} & a_{\ell+r+1} & \dots & a_n & 0 & \dots & 0 \\ = & t_1 & \dots & t_r & t_{r+1} & \dots & t_{n-\ell} & t_{n-\ell+1} & \dots & t_n \\ + & ? & \dots & ? & a_1 & \dots & a_{n-r-\ell} & a_{n-r-\ell+1} & \dots & a_{n-r} \end{array}$$

FIGURE 6.2: Bit representation of $f(x)$.

Algorithm 43. The algorithm for reversing f (Case 2 and 4).

Input: The value to reverse v.
Output: The set of possible solutions S.

Function $Minus(temp_1, temp_2, size)$**:**
  if $temp_2 > temp_1$ **then**
    $high = (1 \ll (size+1)) + temp_1 - temp_2$;
  else
    $high = temp_1 - temp_2$;
  end
  return $high \ \& \ 1^{size}$;
Function $ComputeMid(low)$**:**
  $q \leftarrow \lfloor (n-r)/r \rfloor$;
  $m \leftarrow n - r \bmod r$;
  $mid \leftarrow 0$;
  for $i \in [1, q]$ **do**
    $temp_1 \leftarrow (mid \ll r) \mid low$;
    $temp_1 \leftarrow (temp_1 \oplus (temp_1 \ll \ell)) \ \& \ 1^{ir}$; //only for Case 4
    $temp_2 \leftarrow v \ \& \ 1^{ir}$;
    $mid \leftarrow Minus(temp_1, temp_2, ir)$;
  end
  if $m \neq 0$ **then**
    $temp_1 \leftarrow (mid \ll r) \mid low$;
    $temp_1 \leftarrow (temp_1 \oplus (temp_1 \ll \ell)) \ \& \ 1^{n-r}$; //only for Case 4
    $temp_2 \leftarrow v \ \& \ 1^{n-r}$;
    $mid \leftarrow Minus(temp_1, temp_2, n-r)$;
  end
  return mid
Function $Main(v)$**:**
  $S \leftarrow \varnothing$;
  for $low \in [0, 1^r]$ **do**
    $mid \leftarrow ComputeMid(low)$;
    $s \leftarrow (mid \ll r) \mid low$;
    if $f(s) == v$ **then**
      $S \leftarrow S \cup s$;
    end
  end
  return S;

□

Corollary 6.1. There exists an attack on the Flash Player PRNG with time complexity $\mathcal{O}(2^{21})$.

Algorithm 44. The algorithm for reversing f (Case 3).

Input: The value to reverse v.
Output: The set of possible solutions S.

```
1  S ← ∅;
2  for low ∈ [0, 1^{n−r}] do
3  |  temp_1 ← (low ⊕ (low ≪ ℓ)) & 1^{n−r};
4  |  temp_2 ← v & 1^{n−r};
5  |  high ← Minus(temp_1, temp_2, n − r);
6  |  for mid ∈ [0, 1^{2r−n}] do
7  |  |  s ← (high ≪ r) | (mid ≪ (n − r)) | low;
8  |  |  if f(s) == v then
9  |  |  |  S ← S ∪ s;
10 |  |  end
11 |  end
12 end
13 return S;
```

	Conditions	Time complexity
1	$n - r \leqslant \ell$ and $n \leqslant 2r$	$\mathcal{O}(c_{13}2^{n-r})$
3	$n - r \geqslant \ell$ and $n \leqslant 2r$	$\mathcal{O}(c_{13}2^{n-r})$

TABLE 6.2: Attack parameters for Lemma 6.2.

$$\begin{array}{cccccccccc} & a_1 & \dots & a_{n-\ell} & a_{n-\ell+1} & \dots & a_r & a_{r+1} & \dots & a_n \\ \oplus & a_{\ell+1} & \dots & a_n & 0 & \dots & 0 & 0 & \dots & 0 \\ = & t_1 & \dots & t_{n-\ell} & t_{n-\ell+1} & \dots & t_r & t_{r+1} & \dots & t_n \\ + & a_1 & \dots & a_1 & a_1 & \dots & a_1 & a_1 & \dots & a_{n-r} \end{array}$$

FIGURE 6.3: Bit representation of $f(x)$.

6.1.3 Improving

In this subsection we consider implementation dependent attacks. For simplicity we assume the behavior of the Microsoft Visual Studio and GCC compilers on x86 and x64 architectures. Other compilers' behaviors can be modeled similarly.

Lemma 6.2. Let $c_{13} = \lfloor r/\ell \rfloor + 1$. For each (set of) condition(s) presented in Column 2 of Table 6.1 there exists an attack whose corresponding time complexity is presented in Column 3 of Table 6.1.

Proof. When $n - r \leqslant \ell$, we can explicit the function f as shown in Figure 6.3. Note that a_1 is the sign bit used to fill the gaps. With this in mind, we use the existing knowledge (low) to gradually recover the $2r - n$ bits (mid). Thus, we improve the exhaustive search of the mid part from Algorithm 42. This leads to a time complexity of $\mathcal{O}(2^r(q+1))$.

Algorithm 45. The improved algorithm for reversing f (Case 1).

Input: The value to reverse v.
Output: The set of possible solutions S.

Function $Add(v, high)$**:**
  if $high \in [0, 1^{n-r-1}]$ **then**
    $temp \leftarrow v + high$;
  else
    $temp \leftarrow v + high \oplus 1^r 0^{n-r}$;
  end
  return $temp$;
Function $SpeedMid(low, temp, size, step)$**:**
  $q \leftarrow \lfloor size \,/\, step \rfloor$;
  $m \leftarrow size \bmod step$;
  $temp_1 \leftarrow low$;
  for $i \in [0, q-1]$ **do**
    $offset \leftarrow (i+1) \cdot step$;
    $temp_2 \leftarrow (temp_1 \oplus (temp \gg offset)) \;\&\; 1^{n-r}$;
    $mid \leftarrow mid \mid (temp_1 \ll (i \cdot step))$;
    $temp_1 \leftarrow temp_2$
  end
  $offset \leftarrow (q+1) \cdot step$;
  $temp_2 \leftarrow (temp_1 \oplus (temp \gg offset)) \;\&\; 1^m$;
  $mid \leftarrow mid \mid (temp_1 \ll (q \cdot step))$;
  return mid;
Function $Main(v)$**:**
  $S \leftarrow \varnothing$;
  for $high \in [0, 1^{n-r}]$ **do**
    $temp \leftarrow Add(v, high)$;
    $low \leftarrow temp \;\&\; 1^\ell$;
    $mid \leftarrow SpeedMid(low, temp, r-\ell, \ell)$;
    $s \leftarrow (high \ll r) \mid (mid \ll \ell) \mid low$;
    if $f(s) == v$ **then**
      $S \leftarrow S \cup s$;
    end
  end
  return S;

When $n-r \geqslant \ell$, Figure 6.2 becomes Figure 6.4. In the third case, we adapt the algorithm used in Case 1 to take into account overlapping bits. Thus, we obtain the same time complexity.

$$\begin{array}{cccccccccc}
 & a_1 & \dots & a_r & a_{r+1} & \dots & a_{n-\ell} & a_{n-\ell+1} & \dots & a_n \\
\oplus & a_{\ell+1} & \dots & a_{\ell+r} & a_{\ell+r+1} & \dots & a_n & 0 & \dots & 0 \\
= & t_1 & \dots & t_r & t_{r+1} & \dots & t_{n-\ell} & t_{n-\ell+1} & \dots & t_n \\
+ & a_1 & \dots & a_1 & a_1 & \dots & a_{n-r-\ell} & a_{n-r-\ell+1} & \dots & a_{n-r}
\end{array}$$

FIGURE 6.4: Bit representation of $f(x)$.

Algorithm 46. The improved algorithm for reversing f (Case 3).

Input: The value to reverse v.
Output: The set of possible solutions S.

$S \leftarrow \varnothing$; $e \leftarrow n - r - \ell$;
for $low \in [0, 1^{n-r}]$ **do**
  $temp_1 \leftarrow (low \oplus (low \ll \ell))$ & 1^{n-r};
  $temp_2 \leftarrow v$ & 1^{n-r};
  $high \leftarrow Minus(temp_1, temp_2, n - r)$;
  $temp \leftarrow Add(v, high)$;
  $mid \leftarrow SpeedMid(low, temp, 2r - n + e, \ell)$;
  $mid \leftarrow mid \gg e$;
  $s \leftarrow (high \ll r) \mid (mid \ll (n - r)) \mid low$;
  if $f(s) == v$ **then**
    $S \leftarrow S \cup s$;
  end
end
return S;

□

Corollary 6.2. There exist an attack on the Flash Player PRNG with time complexity $\mathcal{O}(2^{11})$.

Corollary 6.3. For any choice of ℓ and r there exists an attack whose time complexity is at most $\mathcal{O}(n2^{n/2})$.

Proof. According to Lemma 6.1, Cases 2 and 4 there exists an attack with complexity $\mathcal{O}(2^r) \leqslant \mathcal{O}(2^{n/2})$. In Cases 1 and 3 we make use of the attacks presented in Lemma 6.2. Thus, there exists an attack with complexity $\mathcal{O}(c_{13}2^{n-r}) \leqslant \mathcal{O}(c_{13}2^{n/2}) \leqslant \mathcal{O}(n2^{n/2})$. As a result, in the general case we obtain our statement. □

Corollary 6.4. There exists an attack on the Flash Player PRNG with time complexity at most $\mathcal{O}(2^{21})$ independent of ℓ and r.

	Case 1 ($l = 13, r = 21$)			Case 2 ($l = 23, r = 11$)
	Algorithm 41	Algorithm 42	Algorithm 45	Algorithm 43
$f(x)$	$2.16055s$	$2.82102s$	$0.00717478s$	$0.00608366s$
PRNG	$10.6442s$	$13.8981s$	$0.036917s$	$0.0334592s$

TABLE 6.3: Running times for reversing the function f and the PRNG (Cases 1 and 2).

	Case 3 ($l = 9, r = 21$)		Case 4 ($l = 19, r = 11$)
	Algorithm 44	Algorithm 46	Algorithm 43
$f(x)$	$2.77496s$	$0.00708749s$	$0.00809854s$
PRNG	$14.6386s$	$0.0432187s$	$0.0437757s$

TABLE 6.4: Running times for reversing the function f and the PRNG (Cases 3 and 4).

6.1.4 Experimental Results

We implemented Algorithms 42 to 46 and used 32 random seed values to test if our algorithms succeed in recovering the seed for all $1 \leqslant r < 32$ and $1 \leqslant l < 32$. The compilers we worked with are Microsoft Visual Studio 2017 version 15.7.5 with the C++14 extension activated and GCC version 5.4.0 with the C++11 extensions activated. The tests were a success.

In another experiment we run Algorithms 42 to 46 and Algorithm 60 with 2000 random seed values and used the function *omp_get_wtime()* [15] to compute the running time necessary to invert the function f and the corresponding PRNG. The programs were run on a CPU Intel i7-4790 4.00 GHz and compiled with GCC with the O3 flag activated. The results for the 2000 iterations can be found in Tables 6.3 and 6.4. Note that the average time for brute forcing one value is $2.88861s$ for f and $13.2578s$ for PRNG.

6.2 Bias Amplifiers

In [264] the authors propose an interesting mechanism that blurs the line between what constitutes a Trojan horse and what does not. To detect their mechanism, a program needs to somehow differentiate between a naturally unstable random number generator (RNG) and artificially unstable one (obtained by means of certain mathematical transformations). To our knowledge, [264] is the only previous work that discuses this topic.

More precisely, in [264] a digital filter is described. Usually, digital filters are applied to RNGs to correct biases[3], but this filter has an opposite purpose. When applied to a stream of unbiased bits the filter is benign. On the other hand, if applied to a stream of biased bits the filter amplifies their bias. Thereby, making the RNG worse.

In this section we extend the filter from [264][4], provide a new class of filters and discuss some new possible applications. When designing bias amplifiers, a couple of rules must be respected. The first one states that if the input bits are unbiased or have a maximum bias (*i.e.* the probability of obtaining 1 is either 0 or 1) the filter must maintain the original bias. For unbiased bits this rule keeps the amplifiers transparent to a user, as long as the noise source functions according to the original design parameters. For maximum bias the rule is a functional one. Since the RNG is already totally broken, changing the bias does not make sense (from a designing point of view). The second rule states that the filter should amplify the bias in the direction that it already is. This rule helps the designer amplify the bias in an easier manner.

The main application we propose for these filters is RNG testing (*e.g.*, boosting health tests implemented in a RNG). Recent standards [158, 249] require a RNG to detect failures and one such method for early detection can be to apply an amplifier and then do some lightweight testing[5]. Based on the results obtained in Sections 6.2.2 and 6.2.3, we introduce a generic architecture for implementing health tests in Section 6.2.4.1. More precisely, using a lightweight test on the amplified bits the architecture can detect deviations from the uniform distribution. To validate our architecture, we first run a series of experiments on RNGs that generate uniformly independent and identically distributed bits. We also show that our architecture can detect deviation from the initial parameters of the u.i.i.d. source. In Section 6.2.5 we extend the preliminary results to noise sources that have a Bernoulli distribution and show that the architecture can detect, starting from the design phase, badly broken sources. To support our results we develop a theoretical model and provide the reader with simulations based on our model. Note that our theoretical model also explains why our architecture can detect deviation from the initial parameters

Due to recent events [36, 205, 50, 69] RNGs have been under a lot of scrutiny. Thus, wondering what kind of mechanisms can be implemented by a malicious third party in order to weaken or destabilize a system becomes natural. Amplifying filters provide a novel example of how one can achieve this. Based on the failure detection mechanisms

[3] They are called randomness extractors [95].

[4] The filter presented in [264] corresponds to the greedy amplifier with parameter $n = 3$ described in Section 6.2.2.

[5] for example the tests described in [134]

proposed in Section 6.2.4.1, we show, for example, how a manufacturer can manipulate the architecture to become malicious.

6.2.1 Preliminaries

Conventions. In this section, we consider binary strings composed of independent bits that follow a Bernoulli distribution $B(\tilde{p})$, where $\tilde{p}$ is the probability of obtaining a 1. The probability of obtaining a 0 is denoted by $\tilde{q} = 1 - \tilde{p}$.

Let u be a binary string and $A \subseteq \mathbb{Z}_2^n$. Then $w(u)$ denotes the hamming weight of u and $w(A)$ the set $\{w(a) \mid a \in A\}$. Note that since we are working with i.i.d. bits, for any $u, v \in \mathbb{Z}_2^n$ such that $w(u) = w(v)$, the equality $Pr[u] = Pr[v]$ holds. Thus, from a probabilistic point of view, it does not matter which element of the set $\{u \in A \mid w(u) = k\}$ we choose to work with.

The element $\min(A)$ ($\max(A)$) is the smallest (biggest) integer of the set A, while $\min_w(A)$ ($\max_w(A)$) is an element from A that has the smallest (biggest) hamming weight. We say that a pair of sets (S_0, S_1) is an equal partition of the set S if the following hold: $S = S_1 \cup S_2$, $S_1 \cap S_2 = \varnothing$ and $|S_1| = |S_2|$.

6.2.1.1 Digital Filters

In this section, we consider a digital filter to be a mapping from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2$. If we continuously apply a filter to data generated by a RNG[6], then three types of filters arise:

- `bias amplifier` - the output data has a bigger bias than the input data;
- `neutral filter` - the output data has the the same bias as the input data;
- `bias corrector`[7] - the output data has a smaller bias than the input data.

Let (S_0, S_1) be an equal partition of a set S. Let D be a digital filter such that it maps S_0 and S_1 to 0 and 1, respectively (see Table 6.5). Also, let ε_D be the output bias of D. We say that a `bias amplifier` is maximal if ε_D is maximal over all the equal partitions of $\mathbb{Z}_2^n$. To compare `bias amplifiers` we measure the distance between $Pr[S_1]$ and $Pr[S_0]$.

Before stating our results, some restrictions are needed. If the input bits are unbiased (*i.e.* $\tilde{p} = \tilde{q} = 1/2$) or have a maximum bias (*i.e.* $\tilde{p} = 0$ or $\tilde{q} = 0$) we require the filter to maintain the original bias. If one replaces a `bias corrector` with a `bias amplifier`,

[6] Note that except for $n = 1$ the bit rate of the RNG will drop.
[7] We prefer to use this notion instead of randomness extractor, because it simplifies our framework.

Bit 0	**Bit** 1
S_0	S_1

TABLE 6.5: Conversion table.

the amplifier must behave as the corrector when the RNG has bias 0 or 1/2. The last requirement is that the filter amplifies the bias in the direction that it already is. Without loss of generality, we assume that the bias is towards 1.

6.2.1.2 Combinatorial Results

To ease description, we use the notation C_k^n to denote binomial coefficients. Pascal's identity states that $C_k^n = C_k^{n-1} + C_{k-1}^{n-1}$, where $1 \leqslant k \leqslant n$. Note that $|\{u \in \mathbb{Z}_2^n \mid w(u) = k\}| = C_k^n$. We further state a lemma from [67].

Lemma 6.3. Let s_i, $i \in [1, b]$ be integers such that $s = s_1 + \ldots + s_b \leqslant a$. Then, the number of integer solutions of the equation $x_1 + \ldots + x_b = a$ with the restrictions $x_i \geqslant s_i$ is $C_{b-1}^{b+a-s-1}$.

6.2.2 Greedy Bias Amplifiers

In this section we generalize and improve the `bias amplifier` described in [264]. We first present a `neutral filter` and based on it we develop a maximal `bias amplifier`. We can easily transform one into the other by changing the conversion table.

Lemma 6.4. Let $S_0 = \{u \in \mathbb{Z}_2^n \mid u = 0\|v, v \in \mathbb{Z}_2^{n-1}\}$ and $S_1 = \{u \in \mathbb{Z}_2^n \mid u = 1\|v, v \in \mathbb{Z}_2^{n-1}\}$. Then $Pr[S_0] = \tilde{q}$ and $Pr[S_1] = \tilde{p}$.

Proof. Since we are working with i.i.d. random bits the following holds

$$Pr[S_0] = \sum_{v \in \mathbb{Z}_2^{n-1}} Pr[0\|v] = \sum_{v \in \mathbb{Z}_2^{n-1}} \tilde{q} Pr[v] = \tilde{q} \sum_{v \in \mathbb{Z}_2^{n-1}} Pr[v] = \tilde{q}.$$

Similarly, we obtain $Pr[S_1] = \tilde{p}$.

□

Using Lemma 6.4 we can devise a `neutral filter` N by mapping all the elements of S_0 and S_1 to 0 and 1, respectively. Starting from the equal partition (S_0, S_1) (Lemma 6.4), using a greedy algorithm (Algorithm 47), we devise a new equal partition that serves as the core of a maximal `bias amplifier`.

Number of switches	Weight of S_0 elements	Weight of S_1 elements
C_0^{n-1}	$n-1$	1
C_1^{n-1}	$n-2$	2
	...	
C_{i-1}^{n-1}	$n-i$	i
	...	

TABLE 6.6: Operations performed during the while loop.

Algorithm 47.

Input: An integer n

Output: An equal partition of $\mathbb{Z}_2^n$

Set $S_0 = \{u \in \mathbb{Z}_2^n \mid u = 0\|v, v \in \mathbb{Z}_2^{n-1}\}$ and $S_1 = \{u \in \mathbb{Z}_2^n \mid u = 1\|v, v \in \mathbb{Z}_2^{n-1}\}$

Set $\alpha = \max_w(S_0)$ and $\beta = \min_w(S_1)$

while $w(\alpha) < w(\beta)$ **do**

  Set $S_0 = (S_0 \backslash \{\alpha\}) \cup \{\beta\}$ and $S_1 = (S_1 \backslash \{\beta\}) \cup \{\alpha\}$

  Update $\alpha = \max_w(S_0)$ and $\beta = \min_w(S_1)$

end

return (S_0, S_1)

Lemma 6.5. Let k be a positive integer and let (S_0, S_1) be the output of Algorithm 47. Then the following properties hold

1. If $n = 2k+1$ then $S_0 = \{u \mid 0 \leqslant w(u) \leqslant k\}$ and $S_1 = \{u \mid k+1 \leqslant w(u) \leqslant n\}$. Also, $Pr[S_0] = \sum_{i=0}^{k} C_i^n \tilde{p}^i \tilde{q}^{n-i}$ and $Pr[S_1] = \sum_{i=0}^{k} C_i^n \tilde{p}^{n-i} \tilde{q}^i$.
2. If $n = 2k$ then $S_0 = \{u \mid 0 \leqslant w(u) \leqslant k-1\} \cup T_0$ and $S_1 = \{u \mid k+1 \leqslant w(u) \leqslant n\} \cup T_1$, where (T_0, T_1) is an equal partition of $\{u \in \mathbb{Z}_2^n \mid w(u) = k\}$. Also, $Pr[S_0] = \sum_{i=0}^{k-1} C_i^n \tilde{p}^i \tilde{q}^{n-i} + \frac{C_k^n}{2}(\tilde{p}\tilde{q})^k$ and $Pr[S_1] = \sum_{i=0}^{k-1} C_i^n \tilde{p}^{n-i} \tilde{q}^i + \frac{C_k^n}{2}(\tilde{p}\tilde{q})^k$.
3. If $\varepsilon = 0$ then $Pr[S_0] = Pr[S_1] = \frac{1}{2}$ and if $\varepsilon = \frac{1}{2}$ then $Pr[S_0] = 0$ and $Pr[S_1] = 1$.

Proof. During the while loop Algorithm 47 swaps the elements whose weight is written in Column 2, Table 6.6 with the elements that have their weight written in Column 3, Table 6.6.

The while loop ends when $w(\alpha) \geqslant w(\beta)$. According to Table 6.6, this is equivalent with $n - i \geqslant i$. When $n = 2k+1$ we obtain that the while loop stops when $i \leqslant k+1$. When $n = 2k$ the loop stops when $i \leqslant k$. Thus, we obtain the sets S_0 and S_1. The probabilities $Pr[S_0]$ and $Pr[S_1]$ are a direct consequence of the structure of the sets and the fact that $C_k^n = C_{n-i}^n$. The last item is simply a matter of computation.

□

Bit 0	**Bit** 1	$\tilde{p}$	**Corr.**	**Amp.**
000, 001, 010, 100	111, 110, 101, 011	$\frac{1}{2}+\frac{3}{2}\varepsilon-2\varepsilon^3$	-	✓
000, 001, 010, 011	111, 110, 101, 100	$\frac{1}{2}+\varepsilon$	-	-
000, 001, 101, 011	111, 110, 010, 100	$\frac{1}{2}+\frac{1}{2}\varepsilon+2\varepsilon^3$	✓	-
000, 110, 101, 011	111, 001, 010, 100	$\frac{1}{2}+4\varepsilon^3$	✓	-
000, 001, 010, 111	100, 110, 101, 011	$\frac{1}{2}+\frac{1}{2}\varepsilon+2\varepsilon^2-2\varepsilon^3$	✓	-
011, 110, 010, 100	000, 001, 101, 111	$\frac{1}{2}+2\varepsilon^2$	✓	-
011, 001, 010, 100	000, 110, 101, 111	$\frac{1}{2}+\frac{1}{2}\varepsilon+2\varepsilon^2-2\varepsilon^3$	✓	-

TABLE 6.7: Trigraph conversion table.

In Table 6.7 we present all the possible partitions of $\mathbb{Z}_2^3$. We mapped these partitions to 0 and 1 in such a way that $\tilde{p} > \frac{1}{2}$. Note that bias amplification happens only for the partition presented in the first entry of the table[8]. This is not true in general. For example, when $n = 5$, we obtain another bias amplifier if we map the set $S_1' = (S_1 \backslash \{11100\}) \cup \{11000\}$[9] and $S_0' = \mathbb{Z}_2^5 \backslash S_1'$ to 1 and 0. We will now prove that (S_0', S_1') is the basis of an amplifier. Note that $Pr[S_1'] = \frac{1}{2} + \frac{7}{4}\varepsilon - 4\varepsilon^3 + 4\varepsilon^5$. Thus, we have

$$Pr[S_1'] - \tilde{p} = \frac{\varepsilon}{4}\left(16\varepsilon^4 - 16\varepsilon^2 + 3\right) > 0,$$

which is equivalent with

$$16t^2 - 16t + 3 > 0, \tag{6.1}$$

where $t = \varepsilon^2$. Equation (6.1) has two solution $t_1 = \frac{1}{4}$ and $t_2 = \frac{3}{4}$. Thus, the sign of the quadratic function is negative only when $\frac{1}{4} < t < \frac{3}{4}$. By taking into account the complement rule and that $0 < \varepsilon < \frac{1}{2}$, we obtain an amplifier by converting (S_0', S_1') to $(0, 1)$.

Lemma 6.6. Let (S_0, S_1) be the output of Algorithm 47. If we map all the elements of S_0 and S_1 to 0 and 1, respectively, then we obtain a maximal `bias amplifier` G.

Proof. According to Lemma 6.5 all the lowest and highest probability elements are in S_0 and S_1, respectively. Thus, the statement is true. □

[8] The authors of [264] call it a RNG biasing Trojan horse.
[9] S_1 is the set from Lemma 6.5.

Lemma 6.7. Let (S_0^n, S_1^n) be the output of Algorithm 47 for $n = 2k + 1$. Then the following hold

1. $Pr[S_0^n] = Pr[S_0^{n+1}]$ and $Pr[S_1^n] = Pr[S_1^{n+1}]$.
2. $Pr[S_0^n] - Pr[S_0^{n+2}] = Pr[S_1^{n+2}] - Pr[S_1^n] = 2\varepsilon C_k^n(\tilde{p}\tilde{q})^{k+1}$.
3. $Pr[S_0^n] > Pr[S_0^{n+2}]$ and $Pr[S_1^n] < Pr[S_1^{n+2}]$.
4. $Pr[S_1^n] - Pr[S_0^n] < Pr[S_1^{n+2}] - Pr[S_0^{n+2}]$.

Proof. We prove the first statement using induction. When $k = 1$ we have $S_0^1 = \{0\}$, $S_1^1 = \{1\}$, $S_0^2 = \{00, 01\}$ and $S_1^2 = \{10, 11\}$. Using Lemma 6.4, we obtain $Pr[S_0^1] = \tilde{q} = Pr[S_0^2]$ and $Pr[S_1^1] = \tilde{p} = Pr[S_1^2]$. Thus, proving the statement for the case $k = 1$.

We now assume that the statement is true for k (*i.e.* $Pr[S_0^n] = Pr[S_0^{n+1}]$ and $Pr[S_1^n] = Pr[S_1^{n+1}]$) and we it for $k + 1$. Applying Pascal's identity twice to $Pr[S_0^{n+2}]$ we obtain

$$\begin{aligned} Pr[S_0^{n+2}] &= \sum_{i=0}^{k+1} C_i^{n+2}\tilde{p}^i\tilde{q}^{n+2-i} = \tilde{q}^{n+2} + (n+2)\tilde{p}\tilde{q}^{n+1} \\ &\quad + \sum_{i=2}^{k+1}(C_i^n + 2C_{i-1}^n + C_{i-2}^n)\tilde{p}^i\tilde{q}^{n+2-i}. \end{aligned} \tag{6.2}$$

We rewrite Equation (6.2) as a sum of S^1, S^2, S^3 (described next):

$$\begin{aligned} S^1 &= \tilde{q}^{n+2} + n\tilde{p}\tilde{q}^{n+1} + \sum_{i=2}^{k+1} C_i^n\tilde{p}^i\tilde{q}^{n+2-i} \\ &= \tilde{q}^2 Pr[S_0^n] + C_{k+1}^n\tilde{p}^{k+1}\tilde{q}^{n+1-k}, \end{aligned} \tag{6.3}$$

$$\begin{aligned} S^2 &= 2\tilde{p}\tilde{q}^{n+1} + 2\sum_{i=2}^{k+1} C_{i-1}^n\tilde{p}^i\tilde{q}^{n+2-i} \\ &= 2\sum_{i=0}^{k} C_i^n\tilde{p}^{i+1}\tilde{q}^{n+1-i} = 2\tilde{p}\tilde{q}Pr[S_0^n], \end{aligned} \tag{6.4}$$

$$\begin{aligned} S^3 &= \sum_{i=2}^{k+1} C_{i-2}^n\tilde{p}^i\tilde{q}^{n+2-i} = \sum_{i=0}^{k-1} C_i^n\tilde{p}^{i+2}\tilde{q}^{n-i} \\ &= \tilde{p}^2 Pr[S_0^n] - C_k^n\tilde{p}^{k+2}\tilde{q}^{n-k}. \end{aligned} \tag{6.5}$$

Reassembling Equations (6.3) to (6.5) we obtain

$$\begin{aligned} Pr[S_0^{n+2}] &= Pr[S_0^n] + C_{k+1}^n\tilde{p}^{k+1}\tilde{q}^{n+1-k} - C_k^n\tilde{p}^{k+2}\tilde{q}^{n-k} \\ &= Pr[S_0^n] - 2\varepsilon C_k^n(\tilde{p}\tilde{q})^{k+1}. \end{aligned} \tag{6.6}$$

Applying Pascal's identity twice to $Pr[S_0^{n+3}]$ we obtain

$$Pr[S_0^{n+3}] = \sum_{i=0}^{k+1} C_i^{n+3}\tilde{p}^i\tilde{q}^{n+3-i} + \frac{C_{k+2}^{n+3}}{2}(\tilde{p}\tilde{q})^{k+2} = \tilde{q}^{n+3} + (n+3)\tilde{p}\tilde{q}^{n+2} + \sum_{i=2}^{k+1}(C_i^{n+1} + 2C_{i-1}^{n+1} + C_{i-2}^{n+1})\tilde{p}^i\tilde{q}^{n+3-i} + \frac{C_{k+2}^{n+3}}{2}(\tilde{p}\tilde{q})^{k+2}. \quad (6.7)$$

Let $\alpha = \sum_{i=0}^{k} C_i^{n+1}\tilde{p}^i\tilde{q}^{n+1-i}$. We rewrite Equation (6.7) as a sum of S^4, S^5, S^6 (described next):

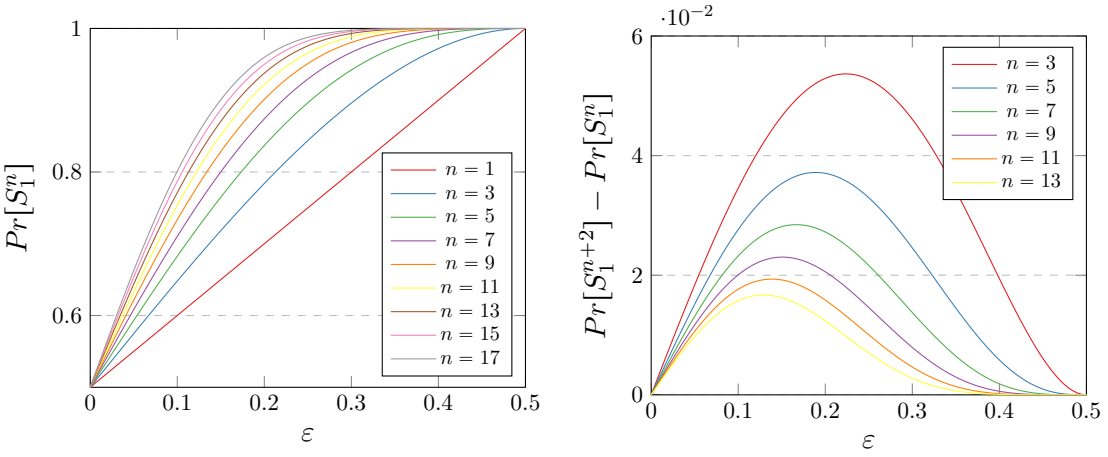
$$\begin{aligned} S^4 &= \tilde{q}^{n+3} + (n+1)\tilde{p}\tilde{q}^{n+2} + \sum_{i=2}^{k+1} C_i^{n+1}\tilde{p}^i\tilde{q}^{n+3-i} \\ &= \tilde{q}^2\alpha + C_{k+1}^{n+1}\tilde{p}^{k+1}\tilde{q}^{n+2-k}, \quad (6.8) \\ S^5 &= 2\tilde{p}\tilde{q}^{n+2} + 2\sum_{i=2}^{k+1} C_{i-1}^{n+1}\tilde{p}^i\tilde{q}^{n+3-i} \\ &= 2\sum_{i=0}^{k} C_i^{n+1}\tilde{p}^{i+1}\tilde{q}^{n+2-i} = 2\tilde{p}\tilde{q}\alpha, \quad (6.9) \\ S^6 &= \sum_{i=2}^{k+1} C_{i-2}^{n+1}\tilde{p}^i\tilde{q}^{n+3-i} = \sum_{i=0}^{k-1} C_i^{n+1}\tilde{p}^{i+2}\tilde{q}^{n+1-i} \\ &= \tilde{p}^2\alpha - C_k^{n+1}\tilde{p}^{k+2}\tilde{q}^{n+1-k}. \quad (6.10) \end{aligned}$$

Reassembling Equations (6.8) to (6.10) we obtain

$$\begin{aligned} Pr[S_0^{n+3}] &= Pr[S_0^{n+1}] + C_{k+1}^{n+1}\tilde{p}^{k+1}\tilde{q}^{n+2-k} - C_k^{n+1}\tilde{p}^{k+2}\tilde{q}^{n+1-k} \\ &\quad - \frac{C_{k+1}^{n+1}}{2}(\tilde{p}\tilde{q})^{k+1} + \frac{C_{k+2}^{n+3}}{2}(\tilde{p}\tilde{q})^{k+2} \\ &= Pr[S_0^{n+1}] - C_k^n(\tilde{p}\tilde{q})^{k+1}\left\{\frac{n+1}{k+1}\left[\tilde{q}^2 - \frac{1}{2}\right]\right. \\ &\quad \left. + \tilde{p}\tilde{q}\left[-\frac{n+1}{k+2} + \frac{(n+1)(n+2)(n+3)}{2(k+1)(k+2)(k+2)}\right]\right\} \\ &= Pr[S_0^{n+1}] - C_k^n(\tilde{p}\tilde{q})^{k+1}\left\{2\left[\tilde{q}^2 - \frac{1}{2}\right] + 2\tilde{p}\tilde{q}\right\} \\ &= Pr[S_0^{n+1}] - 2\varepsilon C_k^n(\tilde{p}\tilde{q})^{k+1}. \quad (6.11) \end{aligned}$$

Applying the induction step to Equations (6.6) and (6.11) we obtain that $Pr[S_0^{n+2}] = Pr[S_0^{n+3}]$. The following equality is a consequence of the complement rule

$$Pr[S_1^{n+2}] = 1 - Pr[S_0^{n+2}] = 1 - Pr[S_0^{n+3}] = Pr[S_1^{n+3}].$$



(A) Probability of S_1^n.

(B) The distance $Pr[S_1^{n+2}] - Pr[S_1^n]$.

FIGURE 6.5: Greedy amplifier.

This completes the proof the first statement. The remaining statements are a direct consequence of Equation (6.6) and the complement rule.

□

Corollary 6.5. Let (S_0^n, S_1^n) be the output of Algorithm 47 for $n = 2k + 1$. Then $Pr[S_0^n] - Pr[S_0^{n+2}] > Pr[S_0^{n+2}] - Pr[S_0^{n+4}]$ and $Pr[S_1^{n+2}] - Pr[S_1^n] > Pr[S_1^{n+4}] - Pr[S_1^{n+2}]$.

Proof. Using Lemma 6.7 we obtain that $Pr[S_0^n] - Pr[S_0^{n+2}] > Pr[S_0^{n+2}] - Pr[S_0^{n+4}]$ is equivalent with $2\varepsilon C_k^n (\tilde{p}\tilde{q})^{k+1} > 2\varepsilon C_{k+1}^{n+2} (\tilde{p}\tilde{q})^{k+2}$. Rewriting the inequality we obtain

$$1 > \frac{(2k+2)(2k+3)}{(k+1)(k+2)} \tilde{p}\tilde{q}.$$

The proof is concluded by observing that

$$\frac{(2k+2)(2k+3)}{(k+1)(k+2)} \tilde{p}\tilde{q} < 4\left(\frac{1}{4} - \varepsilon^2\right) = 1 - 4\varepsilon^2 \leqslant 1.$$

□

Figure 6.5a and Figure 6.5b are a graphical representation of Lemma 6.7 ($n \leqslant 17$) and Corollary 6.5 ($n \leqslant 15$), respectively. The x-axis represents the original bias ϵ, while the y-axis represents $Pr[S_1^n]$ (Figure 6.5a) and $Pr[S_1^{n+2}] - Pr[S_1^n]$ (Figure 6.5b).

Using the properties stated in Lemmas 6.5 and 6.7, we will next describe an equivalent and simplified version of Algorithm 47. Note that devising a greedy `bias amplifier` only makes sense when n is odd.

Algorithm 48.

Input: An odd integer n

Output: An equal partition of $\mathbb{Z}_2^n$

```
Set S_0 = S_1 = ∅
for i = 0, ..., 2^n − 1 do
    if w(i) ⩽ k then
        S_0 = S_0 ∪ {i}
    end
    else
        S_1 = S_1 ∪ {i}
    end
end
return (S_0, S_1)
```

6.2.3 Von Neumann Bias Amplifier

Von Neumann introduced in [252] a simple, yet effective method for correcting the bias of a RNG. Each time the RNG generates two random bits b_1 and b_2, the filter outputs b_1 if and only if $b_1 \neq b_2$. It is easy to see that $Pr[b_1b_2 = 01] = Pr[b_1b_2 = 10] = \tilde{p}\tilde{q}$. Thus, the bias of the output data is 0. We further generalize Von Neumann's method and explain how to replace it's conversion table in order to obtain a maximal `bias amplifier`. Through this section we will restrict n to be of the form $2k$, where k is a positive integer.

Lemma 6.8. Let $V = \{u \in \mathbb{Z}_2^n \mid w(u) = k\}$ and let (V_0, V_1) be an equal partition of V. Then $Pr[V_0] = Pr[V_1] = \frac{C_k^n}{2}(\tilde{p}\tilde{q})^k$.

Proof. Since (V_0, V_1) is an equal partition of V, we obtain that $|V_0| = |V_1| = \frac{|V|}{2} = \frac{C_k^n}{2}$. Note that $Pr[u] = (\tilde{p}\tilde{q})^k$, for any $u \in V$. Combining these two facts we obtain the statement of the lemma.

□

Using Lemma 6.8 we can devise a `corrector filter`[10] V_c by mapping all the elements of V_0 and V_1 to 0 and 1, respectively. In Algorithm 49 we provide an example of how to generate a pair (V_0, V_1).

Algorithm 49.

Input: An integer n

Output: An equal partition of V

```
Set V_0 = V_1 = ∅ and V = {u ∈ Z_2^n | w(u) = k}
Set α = max(V) and β = min(V)
for i = 1, ..., C_k^n/2 do
    Set V_0 = V_0 ∪ {β} and V_1 = V_1 ∪ {α}
    Update V = V\{α, β}
    Set α = max(V) and β = min(V)
end
return (V_0, V_1)
```

We further show that the probabilities V_0 and V_1 get smaller if we increase n. This translates in a lower bit rate if we apply V_c. Note that increasing n does not change the bias of the output data, thus making V_c[11] useless in practice if used only for correcting biases.

Lemma 6.9. Let (V_0^n, V_1^n) be the output of Algorithm 49 for $n = 2k$. Then $Pr[V_0^n] > Pr[V_0^{n+2}]$.

Proof. We remark that $Pr[V_0^n] > Pr[V_0^{n+2}]$ is equivalent with

$$1 > \frac{(2k+1)(2k+2)}{(k+1)(k+1)}\tilde{p}\tilde{q}.$$

The proof is now similar to Corollary 6.5 and thus is omitted.

□

Figure 6.6 is a graphical representation of Lemma 6.9 ($n \leqslant 18$). The x-axis represents the original bias ϵ, while the y-axis represents $Pr[V_0^n]$.

Note that when $\tilde{p} = 0$ or $\tilde{q} = 0$ we obtain $Pr[V_0] = Pr[V_1] = 0$. When constructing a `bias amplifier` V_a we must have the same behavior. Thus, the strings we use to construct V_a need to contain at least a 0 and an 1. When $n = 2$ the only strings that contain 0 and 1 are 01 and 10, but these are the basis for the Von Neumann `bias`

[10] with the bias of the output data 0

[11] for $n \geqslant 4$

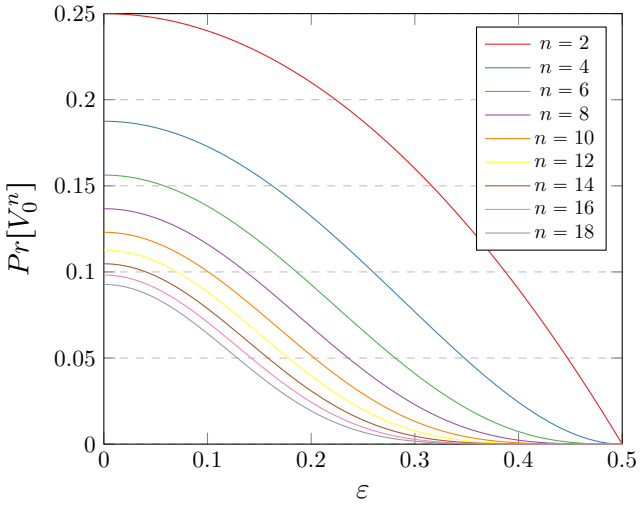


FIGURE 6.6: Von Neumann corrector.

`corrector`. Hence, when $n = 2$ there are no `bias amplifiers`. This leads to the restriction $n \geqslant 4$. We again use a greedy approach (Algorithm 50) and devise a core for V_a.

Algorithm 50.

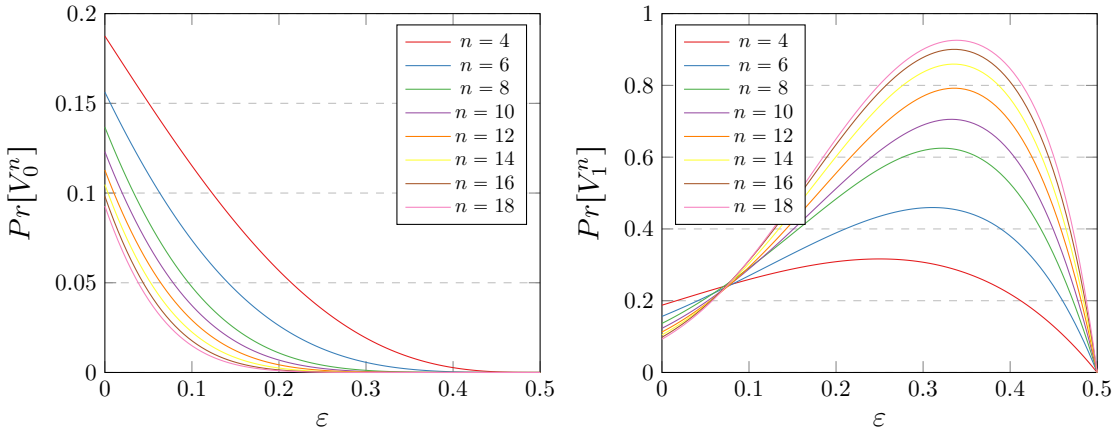
Input: An integer n
Output: Two sets V_0 and V_1

Set $V_0 = V_1 = \varnothing$ and $W = \mathbb{Z}_2^n \backslash \{0^n, 1^n\}$
Set $\alpha = \min_w(W)$ and $\beta = \max_w(W)$
for $i = 1, \ldots, C_k^n/2$ **do**
    Set $V_0 = V_0 \cup \{\alpha\}$ and $V_1 = V_1 \cup \{\beta\}$
    Update $W = W \backslash \{\alpha, \beta\}$
    Set $\alpha = \min_w(W)$ and $\beta = \max_w(W)$
end
return (V_0, V_1)

Lemma 6.10. Let x be an integer such that $\sum_{i=1}^{x} C_i^n < C_k^n/2 < \sum_{i=1}^{x+1} C_i^n$. Define $y = C_k^n/2 - \sum_{i=1}^{x} C_i^n$, $W_0 \subset \{u \in \mathbb{Z}_2^n \mid w(u) = x+1\}$, $W_1 \subset \{u \in \mathbb{Z}_2^n \mid w(u) = n-x-1\}$, such that $|W_0| = |W_1| = y$. Also, let (V_0, V_1) be the output of Algorithm 50. Then the following properties hold

1. $V_0 = \{u \in \mathbb{Z}_2^n \mid 1 \leqslant w(u) \leqslant x\} \cup W_0$ and $V_1 = \{u \in \mathbb{Z}_2^n \mid n-x \leqslant w(u) \leqslant n-1\} \cup W_1$.
2. $Pr[V_0] = \sum_{i=1}^{x} C_i^n \tilde{p}^i \tilde{q}^{n-i} + y\tilde{p}^{x+1}\tilde{q}^{n-x-1}$ and $Pr[V_1] = \sum_{i=1}^{x} C_i^n \tilde{p}^{n-i}\tilde{q}^i + y\tilde{\tilde{p}}^{n-x-1}q^{x+1}$.
3. If $\varepsilon = 0$ then $Pr[S_0] = Pr[S_1] = \frac{1}{2}$ and if $\varepsilon = \frac{1}{2}$ then $\tilde{p} = 1$ and $\tilde{q} = 0$.

Proof. The proof is a direct consequence of Algorithm 50 and thus is omitted.



(A) Probability of V_0^n.

(B) Probability of V_1^n.

FIGURE 6.7: Von Neumann amplifier.

□

Figure 6.7 is a graphical representation of Lemma 6.10 ($n \leqslant 18$). The x-axis represents the original bias ϵ, while the y-axis in Line 8 and Line 8 represents $Pr[V_0^n]$ and $Pr[V_1^n]$, respectively.

Lemma 6.11. Let (V_0, V_1) be the output of Algorithm 50. If we map all the elements of V_0 and V_1 to 0 and 1, respectively, then we obtain a maximal `bias amplifier` V_a.

Proof. According to Lemma 6.10 all the lowest and highest probability elements are in V_0 and V_1, respectively. Thus, the statement is true.

□

Unfortunately, due to the nature of x and y, the best we could do is to heuristically provide a graphical representation of Conjecture 6.1 (Figure 6.8). We could not theoretically prove it in general.

Conjecture 6.1. Let n be even, (S_0^{n-1}, S_1^{n-1}) be the output of Algorithm 47 for $n-1$ and (V_0^n, V_1^n) be the output of Algorithm 50 for n. Denote by $M^n = (Pr[V_1^n] - Pr[V_0^n])/(Pr[V_1^n] + Pr[V_0^n])$. Then $M^n < M^{n+2}$ and $Pr[S_1^{n-1}] - Pr[S_0^{n-1}] < M^n$.

Note that in the case of greedy amplifiers the metric $(Pr[S_1^{n-1}] - Pr[S_0^{n-1}])/(Pr[S_1^{n-1}] + Pr[S_0^{n-1}])$ is equal to $Pr[S_1^{n-1}] - Pr[S_0^{n-1}]$. Thus, Conjecture 6.1 states that the Von Neumann amplifier for a given n is better at amplifying ε than its greedy counterpart. We chose to state the conjecture such that it is true for all $n \geqslant 4$, but, from Figure 6.8, we can observe that as n grows the Von Neumann amplifier becomes better at amplifying

ϵ[12]. Note that in Figure 6.8 the x-axis represents the original bias ϵ, while the y-axis represents the values $Pr[S_1^{n-1}] - Pr[S_0^{n-1}]$ (interrupted line) and M^n (continuous line).

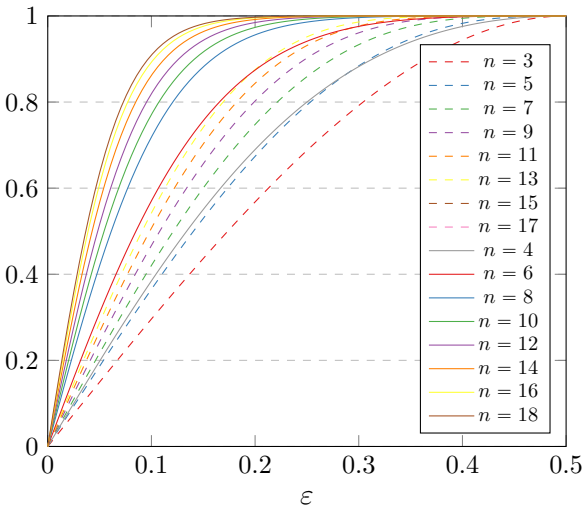


FIGURE 6.8: Comparing greedy amplifiers (interrupted line) with Von Neumann amplifiers (continuous line).

6.2.4 Applications

6.2.4.1 The Good

RNG standards [158, 249] require manufactures to implement some early detection mechanism for entropy failure. Health tests represent one such method for detecting major failures. There are two categories of health tests: startup tests and continuous tests. The former are one time tests conducted before the RNG starts producing outputs, while the latter are tests performed in the background during normal operation.

We propose a generic architecture for implementing health tests (Figure 6.9). We first store data D (obtained from the noise source) in a buffer, then we apply a `bias amplifier` to it and obtain data D_a. Next, we apply some lightweight tests on D_a. If the tests are passed, the RNG outputs D, otherwise D is discarded. Note that the `bias amplifier` can be implemented as a lookup table, thus obtaining no processing overhead at the expense of $\mathcal{O}(2^n)$ memory.

In our instantiations we used the health tests implemented in Intel's processors [134]. Intel's health tests H_i use a sliding window and count how many times each of the six different bit patterns (Column 1, Table 6.8) appear in a 256 bit sample. An example of allowable margins for the six patterns can be found in Column 2, Table 6.8. The

[12] *e.g* the Von Neumann amplifier for $n = 8$ is better than the greedy amplifiers for $n = 3, \ldots, 17$

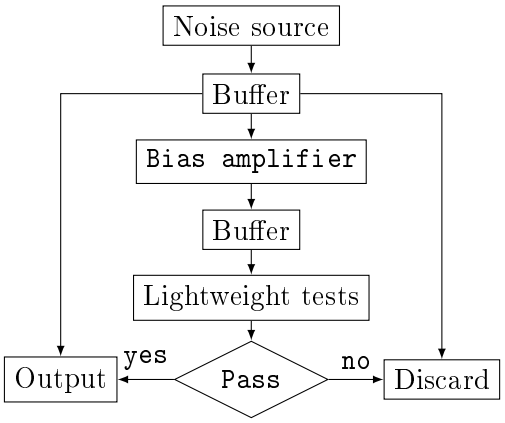


FIGURE 6.9: Generic architecture for implementing health tests.

thresholds mentioned in Tables 6.8 and 6.10 were computed using 10^6 256 bit samples generated using the default RNG from the GMP library [19].

We first propose a continuous test using the greedy amplifiers described in Section 6.2.2. Depending on the available memory we can use one of the greedy amplifiers and then apply H_i. Note that n should be odd due to Lemma 6.7. If the health test are implemented in a processor it is much easier to use $n = 4, 8, 16$. From the health bounds presented in Table 6.8, we can observe that the differences between data without amplification and data with amplification are not significant. Thus, one can easily update an existing good RNG[13] by adding an extra buffer and an amplification module, while leaving the health bounds intact. Note that due to the unpredictable number of output bits produced by a Von Neumann amplifier, greedy amplifiers are better suited for continuous testing.

Bit	**Allowable number of occurrences per sample**			
pattern	**without amp.**	$n = 3$ **amp.**	$n = 5$ **amp.**	$n = 7$ **amp.**
1	$90 < m < 165$	$88 < m < 165$	$89 < m < 167$	$90 < m < 165$
01	$45 < m < 83$	$45 < m < 82$	$46 < m < 83$	$45 < m < 83$
010	$8 < m < 59$	$9 < m < 62$	$10 < m < 58$	$7 < m < 60$
0110	$1 < m < 38$	$2 < m < 34$	$2 < m < 35$	$2 < m < 34$
101	$10 < m < 59$	$10 < m < 61$	$10 < m < 60$	$9 < m < 63$
1001	$1 < m < 35$	$2 < m < 36$	$0 < m < 35$	$1 < m < 35$

TABLE 6.8: Health bounds for greedy amplifiers (amp.).

To test our proposed configuration and obtain some metrics (Table 6.9) we conducted a series of experiments with an u.i.i.d. noise source. More precisely, we generated 105000 256 bit samples using the Bernoulli distribution instantiated with the Mersenne Twister engine (mt19937) found in the C++ random library [3]. Then, we applied the bias

[13] that already has H_i implemented

amplifying filters from Table 6.8 and counted how many samples are marked `pass`. In the case of raw data, a sample is marked `pass`[14] if it passes the H_i test from Column 1, Table 6.8. In the case of bias amplification, if a 256 bit buffer b_a from D_a passes H_i, all the input buffers that where used to produce b_a are marked `pass`. Note that to implement our filters we used lookup tables and thus we had no performance overhead.

From Table 6.9 we can easily see that when the bias is increased, the number of samples that are marked `pass` is lower than H_i in the case of greedy amplifiers. Also, note that the rejection rate is higher as n increases. Thus, enabling us to have an early detection mechanism for RNG failure.

ϵ	**Number of samples marked pass**			
	without amp.	$n = 3$ **amp.**	$n = 5$ **amp.**	$n = 7$ **amp.**
0.00	104999	104997	105000	105000
0.01	104999	104991	104990	105000
0.02	104996	104979	104945	104965
0.03	104988	104925	104685	104384
0.04	104949	104631	103545	101661
0.05	104856	103620	99370	91413
0.06	104598	100668	88845	69832
0.07	104002	93840	69810	41286
0.08	102763	81660	46110	17724
0.09	100411	64332	23460	5404
0.10	96381	44262	9005	1043
0.11	89967	26142	2625	105
0.12	80849	12882	570	0
0.13	69164	5253	65	0
0.14	55856	1704	0	0
0.15	41777	420	0	0
0.16	29039	87	0	0
0.17	18410	21	0	0
0.18	10470	6	0	0
0.19	5331	0	0	0
0.20	2393	0	0	0
0.21	992	0	0	0
0.22	335	0	0	0
0.23	102	0	0	0
0.24	32	0	0	0
0.25	11	0	0	0
0.26	2	0	0	0

TABLE 6.9: Greedy amplifiers (amp.) metrics.

If the design of the RNG has a Von Neumann module, then Von Neumann amplifiers

[14] The terminology used by Intel is that the sample is "healthy".

can be used to devise a startup test. Before entering normal operation, the Von Neumann module can be instantiated using the conversion table of the corresponding amplifier. For example, when $n = 4$ one would use $V_0 = \{0001, 0010, 0100\}$ and $V_1 = \{0111, 1011, 1101\}$[15] instead of $V_0 = \{0011, 0101, 0110\}$ and $V_1 = \{1001, 1010, 1100\}$[16]. The resulting data can then be tested using H_i and if the test pass the RNG will discard the data and enter normal operation. Note that the first buffer from Figure 6.9 is not necessary in this case. Note that Von Neumann amplifiers require $n > 2$, thus the speed of the RNG will drop. This can be acceptable if the data speed needed for raw data permits it, the RNG generates data much faster than the connecting cables are able to transmit or the raw data is further used by a pseudo-random number generator (PRNG).

Bit	**Allowable number of occurrences per sample**			
pattern	$n = 4$ **corr.**	$n = 4$ **amp.**	$n = 6$ **corr.**	$n = 6$ **amp.**
1	$88 < m < 166$	$91 < m < 167$	$89 < m < 167$	$90 < m < 168$
01	$43 < m < 83$	$44 < m < 83$	$44 < m < 85$	$45 < m < 82$
010	$9 < m < 59$	$10 < m < 60$	$7 < m < 58$	$9 < m < 60$
0110	$1 < m < 33$	$1 < m < 36$	$2 < m < 35$	$2 < m < 33$
101	$10 < m < 58$	$11 < m < 61$	$10 < m < 57$	$8 < m < 60$
1001	$0 < m < 34$	$2 < m < 35$	$1 < m < 34$	$1 < m < 34$

TABLE 6.10: Health bounds for Von Neumann correctors (corr.) and amplifiers (amp.).

We also conducted a series of experiments to test the performance of the proposed startup test. This time, we generated u.i.i.d data until we obtained 1000 256-bit samples, applied the bias correcting/amplifying filters from Table 6.10 and counted how many of these samples pass the H_i test from Column 1, Table 6.8. Another metric that we computed is the number of input bits required to generate one output bit.

Note that in Table 6.11 we only wrote the $n = 2$ corrector, since for $n = 4, 6$ the results are almost identical. From Table 6.11 we can easily observe that when the bias is increased the number of samples that pass H_i is lower than the corrector in the case of Von Neumann amplifiers. As in the case of greedy amplifiers, we can observe that the rejection rate is higher as n increases. The experimental data also shows that Von Neumann amplifiers perform better than the greedy amplifiers when rejecting bad samples.

In Table 6.12 we can see that more data is required to generate one bit as n grows. When the bias increases, we can observe that compared to Von Neumann correctors the throughput of the corresponding amplifiers is better. Thus, besides having an early

[15] the sets used to define the maximal Von Neumann amplifier
[16] the sets used to define the Von Neumann corrector

ϵ	**Number of samples that pass H_i**		
	$n = 2$ **corr.**	$n = 4$ **amp.**	$n = 6$ **amp.**
0.00	1000	1000	1000
0.01	1000	1000	1000
0.02	1000	1000	995
0.03	1000	998	940
0.04	1000	981	721
0.05	1000	919	322
0.06	1000	806	79
0.07	1000	567	7
0.08	1000	310	0
0.09	1000	134	0
0.10	1000	53	0
0.11	1000	11	0
0.12	1000	2	0

TABLE 6.11: Von Neumann correctors (corr.) and amplifiers (amp.) metrics.

detection mechanism, it also takes less time to detect if an RNG is broken if we use a Von Neumann amplifier.

TABLE 6.12: Von Neumann correctors (corr.) and amplifiers (amp.) throughput.

ϵ	**Number of input bits per number of output bits**				
	$n = 2$ **corr.**	$n = 4$ **corr.**	$n = 4$ **amp.**	$n = 6$ **corr.**	$n = 6$ **amp.**
0.00	3.9958	10.6646	10.6751	19.1374	19.2776
0.01	3.9978	10.6690	10.6817	19.1873	19.2548
0.02	4.0044	10.6852	10.6885	19.2513	19.2017
0.03	4.0106	10.7272	10.6873	19.3623	19.0892
0.04	4.0202	10.7956	10.6900	19.5129	18.9534
0.05	4.0352	10.8755	10.6952	19.7228	18.7933
0.06	4.0531	10.9713	10.6980	20.0087	18.5889
0.07	4.0755	11.1025	10.6876	20.3259	18.3405
0.08	4.1013	11.2489	10.6709	20.7180	18.0855
0.09	4.1264	11.3916	10.6841	21.1418	17.8104
0.10	4.1594	11.5733	10.6823	21.6591	17.5187
0.11	4.1956	11.7824	10.6862	22.2298	17.2154
0.12	4.2362	12.0062	10.7001	22.8814	16.9006

6.2.4.2 The Bad

One can easily turn the benign architecture presented in Figure 6.9 into a malicious architecture (Figure 6.10). In the new proposed configuration, health tests always output `pass` and instead of outputting D the system outputs D_a.

The malicious configuration can be justified as a bug and can be obtained from the original architecture either by commenting some code lines (similarly to [50]) or by manipulating data buffers (similarly to [69]). Note that code inspection or reverse engineering

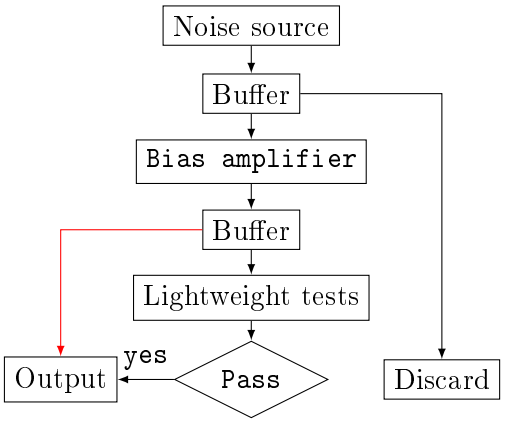


FIGURE 6.10: Generic architecture for infecting RNGs.

will reveal these so called bugs. A partial solution to detection can be implementing the architecture in a tamper proof device and deleting the code if someone tinkers with the device. Another partial solution is embedding the architecture as a submodule in a more complex architecture (similarly to [50]). This solution is plausible due to the sheer complexity of open-source software and the small number of experts who review them [47].

Another problem is that the RNG will output D_as instead of Ds and this translates to lower data rates. A possible solution to this problem is to use D_a as a seed for a PRNG and then output the data produced by the PRNG. Thus, raw data is never exposed. A problem with this approach is that in most cases the PRNG will also mask the bias. The only case that is compatible with this approach is when the bias is large. Therefore one can simply use an intelligent brute force to find the seed. Hence, breaking the system.

A more suitable approach to the aforementioned problem is to use a pace regulator [105]. This method uses an intermediary buffer to store data and supplies the data consumer with a constant stream of bits. Unfortunately, if data requirements are high, then the regulator will require a lot of memory and in some cases the intermediary buffer will be depleted. Thus, failing to provide data.

A solution specific to greedy amplifiers is to implement in all devices a `neutral filter` after D and output the resulting data D_n. Thus, when a malicious version of the RNG is required, one can simply replace the conversion table of the `neutral filter` with the conversion table of the corresponding `bias amplifier`. For example, when $n = 3$ one would change $S_0 = \{000, 001, 010, 100\}$ and $S_1 = \{111, 110, 101, 100\}$[17] with $S_0 = \{000, 001, 010, 100\}$ and $S_1 = \{111, 110, 101, 011\}$[18]. It is easy to see that in this case both D_n and D_a have the same frequency.

[17] the sets used to define the `neutral filter`
[18] the sets used to define the maximal greedy amplifier

Since we are modifying the statistical properties of the raw data, a simple method for detecting changes is black box statistical testing (for example using [13]). Thus, if a user is getting suspicious he can detect the "bugs". Again, a partial solution is to implement the malicious architecture as a submodule inside a more complex architecture either in tamper proof devices, either in complex software. Thus, eliminating the user's access to raw data.

6.2.5 Empirical Investigation into Bernoulli Noise Sources

In this section we extend the experimental results from Section 6.2.4.1 to Bernoulli sources. In order to implement Intel's health tests, we experimentally computed the initial thresholds used in H_i.[19] The results are presented in Table 6.13 and were computed using 10^6 256 bit samples generated based on the Bernoulli distribution instantiated with the Mersenne Twister engine (mt19937) found in the C++ random library [3]. When the data used to generate the thresholds follows a $B(\tilde{p})$ distribution, we denote by $H_i(\tilde{p})$ the resulting health test.

Note that ε might be different for each individual noise source (*e.g.* due to manufacturing variations) and since our scope is to automatically detect large deviations, we had to experimentally determine the initial bounds. A similar process needs to be carried out internally by each RNG during a setup phase. Remark that since the bias is unknown, using theoretical estimates increases design complexity.

Bit pattern	**Allowable number of occurrences per sample**				
	$\tilde{p}=0.1$	$\tilde{p}=0.2$	$\tilde{p}=0.3$	$\tilde{p}=0.4$	$\tilde{p}=0.5$
1	$5-50$	$24-87$	$45-115$	$67-138$	$92-167$
01	$5-44$	$20-64$	$32-75$	$42-80$	$45-83$
010	$3-43$	$13-57$	$14-64$	$12-66$	$10-58$
0110	$0-12$	$0-21$	$0-27$	$2-32$	$1-35$
101	$0-14$	$0-27$	$1-39$	$5-50$	$9-61$
1001	$0-15$	$0-23$	$0-31$	$1-34$	$2-35$

TABLE 6.13: Health bounds for $H_i(\tilde{p})$.

When the architecture presented in Figure 6.9 is instantiated with $H_i(\tilde{p})$ we denote it by $A_t(\tilde{p})$. To analyze the behavior of $A_t(\tilde{p})$ we conducted a series of experiments. Thus, we generated 450450 256 bit samples using the Bernoulli distribution $B(\hat{p})$[20] instantiated with mt19937. Then, we applied the greedy bias amplifying filters from Section 6.2.2 with amplifying factors $n = 1, 3, 5, 7, 9, 11, 13$ and counted how many samples are marked `pass`. The probability P_{pass} of a sequence to be marked `pass` is derived by dividing the

[19] Intel also experimentally generated, using their noise source, the initial thresholds.
[20] Note that in our experiments $\tilde{p}$ is fixed, while $\hat{p}$ drifts from 0.01 to 0.99.

counter with 450450. The results are presented in Figure 6.17. Note that for $\tilde{p} \in [0.5, 1.0]$ the resulting plots are mirrored version of the plots obtained for $\tilde{p} \in [0.0, 0.5]$ and thus are omitted. We further consider $\tilde{p} \leqslant 0.5$.

Remark 6.1. Let $n = 9, 11, 13$. We can easily see that the number of samples that are marked `pass` is close to zero for $\tilde{p} \leqslant 0.3$ and is considerably lower ($P_{pass} < 0.60$) when $0.3 \leqslant \tilde{p} \leqslant 0.4$. We can also observe that when $\tilde{p} \leqslant 0.3$, $\hat{p}$ needs to drift at least 0.05 to have $P_{pass} < 0.40$. When $\tilde{p} = 0.4$, $\hat{p}$ needs to drift at least 0.01 to have $P_{pass} < 0.85$. Thus, if we instantiate $A_t(\tilde{p})$ with greedy amplifiers with $n = 9, 11, 13$ the architecture can detect catastrophic RNG failure (*i.e.* $\tilde{p} \leqslant 0.4$).

Remark 6.2. Let $\tilde{p} = 0.5$. We can easily see that when $n = 9, 11, 13$ and $\hat{p} \notin (0.46, 0.54)$ we have $P_{pass} < 0.97$. Thus, the architecture enables us to detect when a good source deviates[21] with more than 0.04 from 0.5.

We also conducted a series of experiments to test the performance of $A_t(\tilde{p})$ instantiated with the Von Neumann bias amplifying filters from Section 6.2.3 with amplifying factors $n = 1, 4, 6, 8, 10, 12, 14$. So, we generated data with $B(\hat{p})$ until we obtained 10000 256-bit samples[22], then we applied the Von Neumann bias amplifying filters and counted how many of these samples pass the $H_i(\tilde{p})$ test. The results are presented in Figure 6.18. Note that in this case P_{pass} is obtained by dividing the counter with 10000. Another metric that we computed is the number of input bits required to generate one output bit. The results are presented in Figure 6.11.

Remark 6.3. Let $n \geqslant 6$. We can easily see that the number of samples that are marked `pass` is close to zero for $\tilde{p} \leqslant 0.4$. We can also observe that when $\tilde{p} \leqslant 0.3$, $\hat{p}$ needs to drift at least 0.08 to have $P_{pass} < 0.42$. When $\tilde{p} = 0.4$, $\hat{p}$ needs to drift at least 0.03 to have $P_{pass} < 0.84$. Thus, if we instantiate $A_t(\tilde{p})$ with Von Neumann amplifiers with $n = 6, 8, 10, 12, 14$ the architecture can detect catastrophic RNG failure. Also, remark that the drift for Von Neumann amplifiers is larger than in the case of greedy amplifiers.

Remark 6.4. Let $\tilde{p} = 0.5$. We can easily see that when $n = 6$ and $\hat{p} \notin (0.47, 0.53)$ we have $P_{pass} < 0.975$, while for $n \geqslant 8$ and $\hat{p} \notin (0.48, 0.52)$ we have $P_{pass} < 0.985$. Thus, the architecture enables us to detect when a good source deviates with more than 0.03 and, respectively, 0.02 from 0.5. Hence, Von Neumann amplifiers provide us with a better detection method than the greedy counterparts.

Remark 6.5. Although, Von Neumann amplifiers are better suited to detect deviations than greedy amplifiers, we can observe that the data requirements fluctuate and even in

[21] The deviation might be an effect of components' ageing or malfunctioning.

[22] We generated less data than the greedy counterpart due to the amplifier's high bit requirements (see Figure 6.11).

the uniform case efficiency can get to as low 0.01495 $bits_{out}/bits_{in}$. This translates into longer testing times that in the case of greedy amplifiers where the data requirements are fixed. Thus, when choosing between greedy and Von Neumman amplifiers one need to consider what is more important: faster testing times or better detection of source deviations.

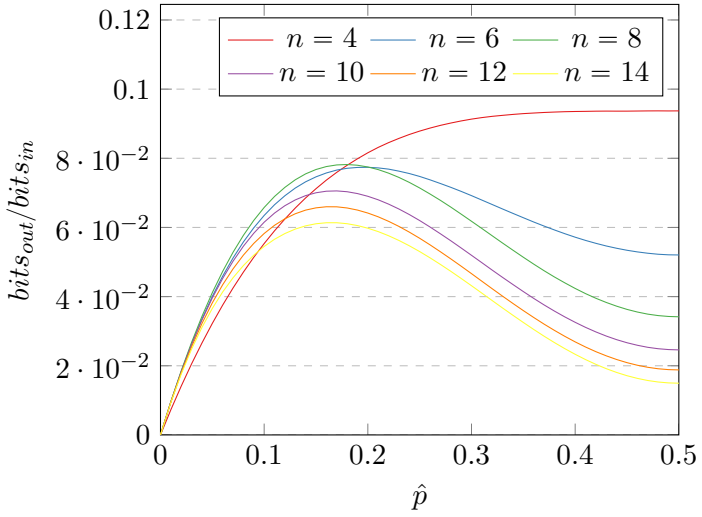


FIGURE 6.11: Bit requirements for Von Neumann amplifiers.

6.2.6 Theoretical Model

In this section we develop the theoretical framework that supports the findings presented in Section 6.2.5. First we derive a series of lemmas that are later used for estimating P_{pass}. Then, we provide the reader with a series of simulations.

6.2.6.1 Description

We first state a known result regarding the number of 1s (denoted by c_1) in a sequence of length m. Then, we determine the number of overlapping 01s (denoted by c_{01}), 010s (denoted by c_{010}), 101s (denoted by c_{101}), 0110s (denoted by c_{0110}) and 1001s (denoted by c_{1001}) in a sequence of length m. Note that we assume that all the sequences are generated by a Bernoulli noise source $B(p)$.

Lemma 6.12. Let k a positive integer. Then

$$Pr[c_1 = k] = C_k^m \cdot p^k \cdot q^{m-k}.$$

Remark 6.6. Note that when the Hamming weight ω of a sequence is either 0 or m, we have $c_{01} = c_{010} = c_{101} = c_{0110} = c_{1001} = 0$. Thus, when computing the probability P of

k occurrences of a pattern, the cases $\omega = 0$ and $\omega = m$ add to P a term $q^m + p^m$ only when $k = 0$. For uniformity, we further consider the term $q^m + p^m$ as being implicit.

Lemma 6.13. Let k be a positive integer. Then

$$Pr[c_{01} = k] = \sum_{\omega=1}^{m-1} C_k^{\omega} \cdot C_k^{m-\omega} \cdot p^{\omega} \cdot q^{m-\omega}.$$

Proof. First we form a sequence Γ of k concatenated 01s. Thus, for a given Hamming weight ω we are left with $\omega - k$ 1s and $m - \omega - k$ 0s that are unused. When inserting the $m - 2k$ bits into Γ, for ease of description, we always insert 0s and 1s before a 0 and, respectively, a 1 that is already in Γ. Remark that we can insert a number of 1s and 0s at the beginning and, respectively, the end of Γ without changing the number of 01 patterns.

After inserting in Γ the $m - 2k$ bits we obtain the sequence

$$\underbrace{1\ldots1}_{y_0}\underbrace{0\ldots0}_{x_1}0\,1\underbrace{\ldots1}_{y_1}1\ldots0\underbrace{\ldots0}_{x_k}0\,1\underbrace{\ldots1}_{y_k}1\,0\underbrace{\ldots0}_{x_{k+1}}$$

with the restrictions

$$x_1 + \ldots + x_{k+1} = m - \omega - k, x_i \geqslant 0, i \in [1, k+1], \tag{6.12}$$

$$y_0 + \ldots + y_k = \omega - k, y_i \geqslant 0, i \in [0, k]. \tag{6.13}$$

According to Lemma 6.3, the number of solutions that satisfy Equation (6.12) and Equation (6.13) is $C_k^{m-\omega}$ and, respectively, C_k^{ω}. Using the number of solutions and the law of total probability we obtain the desired result. □

Lemma 6.14. Let k a positive integer. Then

$$Pr[c_{010} = k] = \sum_{\omega=1}^{m-1} \sum_{r=k}^{\omega} C_r^{m-\omega} \cdot C_k^r \cdot C_{r-k}^{\omega-r} \cdot p^{\omega} \cdot q^{m-\omega}.$$

Proof. Let r be the maximum number of 01 patterns. Using a similar reasoning to the proof of Lemma 6.13 we obtain the sequence

$$\underbrace{1\ldots1}_{y_0}\underbrace{0\ldots0}_{x_1}0\,1\underbrace{\ldots1}_{y_1}1\ldots0\underbrace{\ldots0}_{x_r}0\,1\underbrace{\ldots1}_{y_r}1\,0\underbrace{\ldots0}_{x_{r+1}}$$

with the restrictions

$$x_1 + \ldots + x_{r+1} = m - \omega - r, x_i \geqslant 0, i \in [1, r+1], \tag{6.14}$$

$$y_0 + \ldots + y_r = \omega - r, y_0 \geqslant 0. \tag{6.15}$$

According to Lemma 6.3 the number of solutions that satisfy Equation (6.14) is $C_r^{m-\omega}$.

To ensure that there are exactly k 010 patterns Equation (6.15) that have to satisfy the following condition: exactly k out of r $y_1, \ldots, y_r$ must be 0. We further assume that $y_1 = \ldots = y_k = 0$ and $y_{k+1}, \ldots, y_r \geqslant 1$. Note that the number of solutions obtained under this assumption must be multiplied with a factor of C_k^r. Equation (6.15) now becomes

$$y_0 + y_{k+1} + \ldots + y_r = \omega - r, y_0 \geqslant 0, y_i \geqslant 1, i \in [k+1, r] \tag{6.16}$$

According to Lemma 6.3 the number of solutions for Equation (6.16) is $C_{r-k}^{\omega-r}$. By adding everything together and using the law of total probability we obtain the desired result. □

Lemma 6.15. Let k be a positive integer. Then

$$Pr[c_{101} = k] = \sum_{\omega=1}^{m-1} \sum_{r=k}^{m-\omega} C_r^{\omega} \cdot C_k^r \cdot C_{r-k}^{m-\omega-r} \cdot p^{\omega} \cdot q^{m-\omega}.$$

Proof. In this case, we consider r as the maximum number of 10 patterns and Γ as the sequence composed of k concatenated 10s. Remark that we can insert a number of 0s and 1s at the beginning and, respectively, the end of Γ without affecting r. Thus, after inserting in Γ the $m - 2k$ bits, we obtain the sequence

$$\underbrace{0\ldots0}_{x_0}\underbrace{1\ldots1}_{y_1}\underbrace{1}\underbrace{0\ldots0}_{x_1}\underbrace{0\ldots1\ldots1}_{y_r}\underbrace{1}\underbrace{0\ldots0}_{x_r}\underbrace{0}\underbrace{1\ldots1}_{y_{r+1}}$$

with the restrictions

$$x_0 + \ldots + x_r = m - \omega - r, x_0 \geqslant 0, \tag{6.17}$$

$$y_1 + \ldots + y_{r+1} = \omega - r, y_i \geqslant 0, i \in [1, r+1]. \tag{6.18}$$

According to Lemma 6.3 the number of solutions that satisfy Equation (6.18) is C_r^{ω}.

To ensure that there are exactly k 101 patterns Equation (6.17) that have to satisfy the following condition: exactly k out of r $x_1, \ldots, x_r$ must be 0. We further assume that $x_1 = \ldots = x_k = 0$ and $x_{k+1}, \ldots, x_r \geqslant 1$. Note that the number of solutions obtained

under this assumption must be multiplied with a factor of C_k^r. Equation (6.17) now becomes

$$\begin{aligned} &x_0 + x_{k+1} + \ldots + x_r = m - \omega - r, \\ &x_0 \geqslant 0, x_i \geqslant 1, i \in [k+1, r] \end{aligned} \tag{6.19}$$

According to Lemma 6.3 the number of solutions for Equation (6.19) is $C_{r-k}^{m-\omega-r}$. By adding everything together and using the law of total probability we obtain the desired result. □

Remark 6.7. In [234], an analysis for $Pr[c_{0110} = k]$ is presented. But, the authors consider bits that have a $B(0.5)$ distribution and that are arranged in a circle. Thus, in our case, we need to reanalyze $Pr[c_{0110} = k]$.

Lemma 6.16. Let k be a positive integer. Then

$$Pr[c_{0110} = k] = \sum_{\omega=1}^{m-1} \sum_{r=k}^{\omega} \sum_{s=t}^{r-k} C_r^{m-\omega} \cdot C_k^r \cdot C_s^{r-k} \cdot C_{r-k-s}^{\omega-2r+s} \cdot p^{\omega} \cdot q^{m-\omega},$$

where $t = 2r - \omega$.

Proof. Let r be the maximum number of 01 patterns. Using a similar reasoning to the proof of Lemma 6.13 we obtain the sequence

$$\underbrace{1\ldots1}_{y_0}\underbrace{0\ldots0}_{x_1}\underbrace{0}_{}\underbrace{1\ldots1}_{y_1}\underbrace{1\ldots}_{}\underbrace{0\ldots0}_{x_r}\underbrace{0}_{}\underbrace{1\ldots1}_{y_r}\underbrace{1}_{}\underbrace{0\ldots0}_{x_{r+1}}$$

with the restrictions presented in Equations (6.14) and (6.15). According to Lemma 6.3 the number of solutions that satisfy Equation (6.14) is $C_r^{m-\omega}$.

To ensure that there are exactly k 0110 patterns Equation (6.15) that have to satisfy the following condition: exactly k out of r $y_1, \ldots, y_r$ must be 1. We further assume that $y_1 = \ldots = y_k = 1$ and $y_{k+1}, \ldots, y_r \neq 1$. Note that the number of solutions obtained under this assumption must be multiplied with a factor of C_k^r.

Let s be the number of y_i, $i \in [k+1, r]$ that are 0. We assume that $y_{k+1} = \ldots = y_{k+s}$. Thus, $y_i \geqslant 2$ for $i \in [k+s+1, r]$. Note that the number of solutions obtained under this assumption must be multiplied with a factor of C_s^{r-k}.

Equation (6.15) now becomes

$$\begin{aligned} &y_0 + y_{k+s+1} + \ldots + y_r = \omega - r - k, \\ &y_0 \geqslant 0, y_i \geqslant 2, i \in [k+s+1, r] \end{aligned} \tag{6.20}$$

According to Lemma 6.3 the number of solutions for Equation (6.20) is $C_{r-k-s}^{\omega-2r+s}$. By adding everything together and using the law of total probability we obtain the desired result. □

Lemma 6.17. Let k a positive integer. Then

$$Pr[c_{1001} = k] = \sum_{\omega=1}^{m-1} \sum_{r=k}^{m-\omega} \sum_{s=t}^{r-k} C_r^{\omega} \cdot C_k^r \cdot C_s^{r-k} \cdot C_{r-k-s}^{m-\omega-2r+s} \cdot p^{\omega} \cdot q^{m-\omega},$$

where $t = 2r - m + \omega$.

Proof. As in Lemma 6.15, r is the maximum number of 10 patterns and we obtain the sequence

$$\underbrace{0\ldots0}_{x_0}\underbrace{1\ldots1}_{y_1}\underbrace{1}\underbrace{0\ldots0}_{x_1}\underbrace{0\ldots1}\underbrace{\ldots1}_{y_r}\underbrace{1}\underbrace{0\ldots0}_{x_r}\underbrace{0}\underbrace{1\ldots1}_{y_{r+1}}$$

with the restrictions presented in Equations (6.17) and (6.18). According to Lemma 6.3 the number of solutions that satisfy Equation (6.17) is C_r^{ω}.

To ensure that there are exactly k 1001 patterns Equation (6.17) that have to satisfy the following condition: exactly k out of r $x_1, \ldots, x_r$ must be 1. We further assume that $x_1 = \ldots = x_k = 1$ and $x_{k+1}, \ldots, x_r \neq 1$. Note that the number of solutions obtained under this assumption must be multiplied with a factor of C_k^r.

Let s be the number of x_i, $i \in [k+1, r]$ that are 0. We assume that $x_{k+1} = \ldots = x_{k+s}$. Thus, $x_i \geqslant 2$ for $i \in [k+s+1, r]$. Note that the number of solutions obtained under this assumption must be multiplied with a factor of C_s^{r-k}.

Equation (6.17) now becomes

$$\begin{aligned} &x_0 + x_{k+s+1} + \ldots + x_r = m - \omega - r - k, \\ &x_0 \geqslant 0, x_i \geqslant 2, i \in [k+s+1, r] \end{aligned} \tag{6.21}$$

According to Lemma 6.3 the number of solutions for Equation (6.21) is $C_{r-k-s}^{m-\omega-2r+s}$. By adding everything together and using the law of total probability we obtain the desired result. □

To compute the probability P_{pass} that a sequence of length m is marked *pass*, we further assume that the 6 statistical tests are independent. Note that this is a standard assumption [13, 258] and offers us an estimate for the real probability. To derive the estimates for the bias amplifiers we use the probabilities from Lemmas 6.5 and 6.10.

Lemma 6.18. For a greedy amplifier with an amplification factor $n = 2k + 1$ and a Bernoulli noise source $B(\tilde{p})$ we have that

$$P_{pass} \simeq \prod_{i=1}^{6} \left(\sum_{\ell=a_i}^{b_i} Pr[c_i = \ell] \right),$$

where a_i, b_i are the lower and upper limits for $c_i \in \{c_1, c_{01}, c_{010}, c_{101}, c_{0110}, c_{1001}\}$ and $p = \sum_{j=0}^{k} C_j^n \cdot \tilde{p}^{n-j}\tilde{q}^j$.

Lemma 6.19. For a Von Neumann amplifier with an amplification factor $n = 2k$ and a Bernoulli noise source $B(\tilde{p})$ we have that

$$P_{pass} \simeq \prod_{i=1}^{6} \left(\sum_{\ell=a_i}^{b_i} Pr[c_i = \ell] \right),$$

where a_i, b_i are the lower and upper limits for $c_i \in \{c_1, c_{01}, c_{010}, c_{101}, c_{0110}, c_{1001}\}$, $p = \sum_{j=1}^{x} C_j^n \tilde{p}^{n-j}\tilde{q}^j + y\tilde{p}^{n-x-1}\tilde{q}^{x+1}$, x is an integer such that $\sum_{j=1}^{x} C_j^n < C_k^n/2 < \sum_{j=1}^{x+1} C_j^n$ and $y = C_k^n/2 - \sum_{j=1}^{x} C_j^n$.

6.2.6.2 Results

To test our model we implemented Lemmas 6.18 and 6.19 using the GMP library [19]. The results are presented in Figure 6.19 and, respectively, Figure 6.20. We can easily remark that for $p \neq 0.1$ the theoretical estimates are close to the experimental results obtained in Section 6.2.5.

Let $\mathcal{P} = \{0.01, 0.02, \ldots, 0.99\}$. To measure the exact distance between the experimental $E_{n,\tilde{p}}$ and theoretical $T_{n,\tilde{p}}$ distributions, we computed the Kullback-Leibler divergence

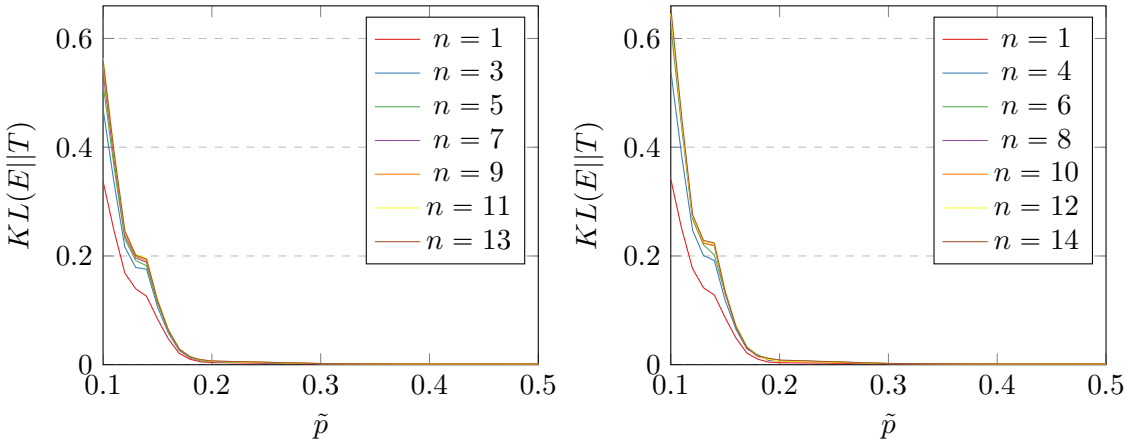
$$KL(E_{n,\tilde{p}}||T_{n,\tilde{p}}) = \sum_{\hat{p}\in\mathcal{P}} E_{n,\tilde{p}}(\hat{p}) \log(E_{n,\tilde{p}}(\hat{p})/T_{n,\hat{p}}(\hat{p}))$$

and the total variation distance

$$\delta(E_{n,\tilde{p}}, T_{n,\tilde{p}}) = \sum_{\hat{p}\in\mathcal{P}} |E_{n,\tilde{p}}(\hat{p}) - T_{n,\tilde{p}}(\hat{p})|/2.$$

Roughly speaking, $KL(E_{n,\tilde{p}}||T_{n,\tilde{p}})$ represents the amount of information lost when $T_{n,\tilde{p}}$ is used to approximate $E_{n,\tilde{p}}$ and $\delta(E_{n,\tilde{p}}, T_{n,\tilde{p}})$ represents the largest possible difference between the probabilities that the two probability distributions can assign to the same event [271]. The results for $\tilde{p} \in \{0.1, 0.11, \ldots, 0.2, 0.3, 0.4, 0.5\}$ are presented in Figures 6.12 and 6.13. We remark that for $\tilde{p} \geqslant 0.20$ we have $KL(E_{n,\tilde{p}}||T_{n,\tilde{p}}) \simeq 0.01$ and

$\delta(E_{n,\tilde{p}}, T_{n,\tilde{p}}) \simeq 0.02$. Thus, the theoretical model is a good estimate for the real probability when $\tilde{p} \geqslant 0.2$. Also, note that Remarks 6.1 to 6.5 remain true for the theoretical estimates.



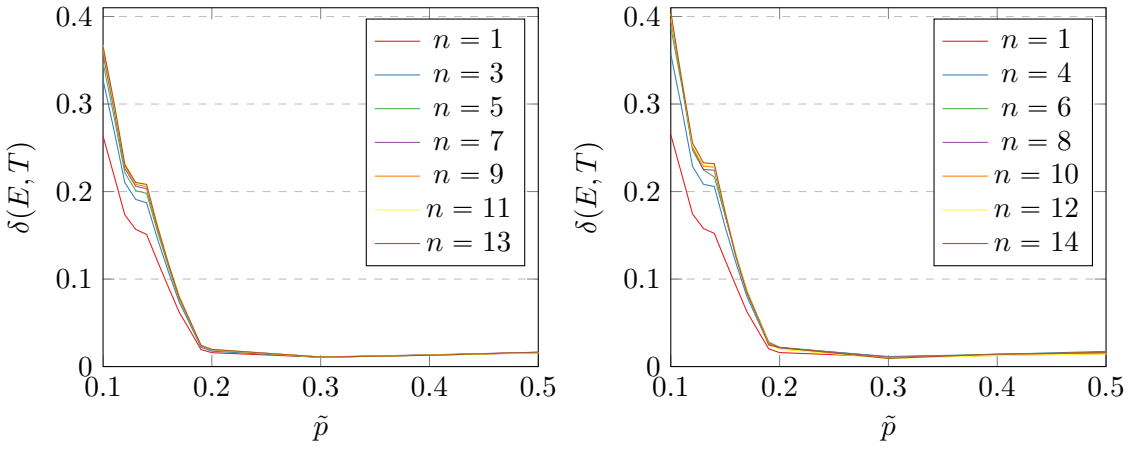
(A) Greedy amplifiers (B) Von Neumann amplifiers

FIGURE 6.12: Kullback-Leibler divergence

When $\tilde{p} < 0.2$ the model starts to distance himself from the real probability, due to the high correlations between the statistical tests. More precisely, the assumption made for Lemmas 6.18 and 6.19 starts to fail. To see how the tests are correlated, we computed the Pearson correlation coefficient

$$r_p(T_1, T_2) = \frac{\sum_{i=1}^{1000}(t_{1i} - \bar{t}_1)(t_{2i} - \bar{t}_2)}{\sqrt{\sum_{i=1}^{1000}(t_{1i} - \bar{t}_1)^2}\sqrt{\sum_{i=1}^{1000}(t_{2i} - \bar{t}_2)^2}},$$

where t_{1i} and t_{2i} represent the number of samples that pass test T_1 and, respectively, T_2 in experiment i, while $\bar{t}_1$ and $\bar{t}_2$ represent the associated expected values. The results for $p \in \mathcal{P}$ are presented in Figure 6.14. Note that in Figure 6.14 the correlation between



(A) Greedy Amplifiers (B) Von Neumann Amplifiers

FIGURE 6.13: Total variance distance

testing for the allowable number of occurrences per sample for 1 and 01 patterns is denoted by 01, for 1 and 010 patterns is denoted by 02 and so on.

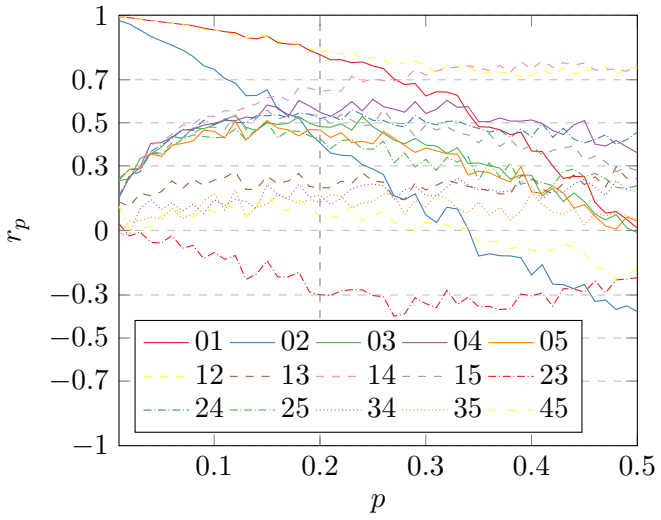
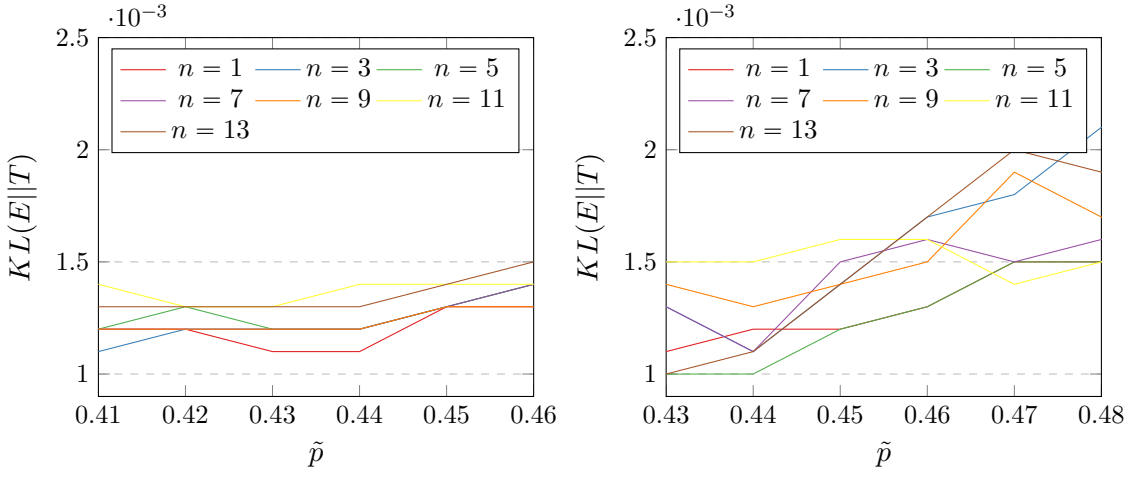


FIGURE 6.14: Tests correlation

6.2.7 Finer measurements

In this section we provide the reader with theoretical data for greedy amplifiers when $\tilde{p} \in [0.41, 0.46]$ and for Von Neumann amplifiers when $\tilde{p} \in [0.43, 0.48]$. According to the Kullback-Leibler divergence and total variance distance presented in Figures 6.15 and 6.16 and this suffices.

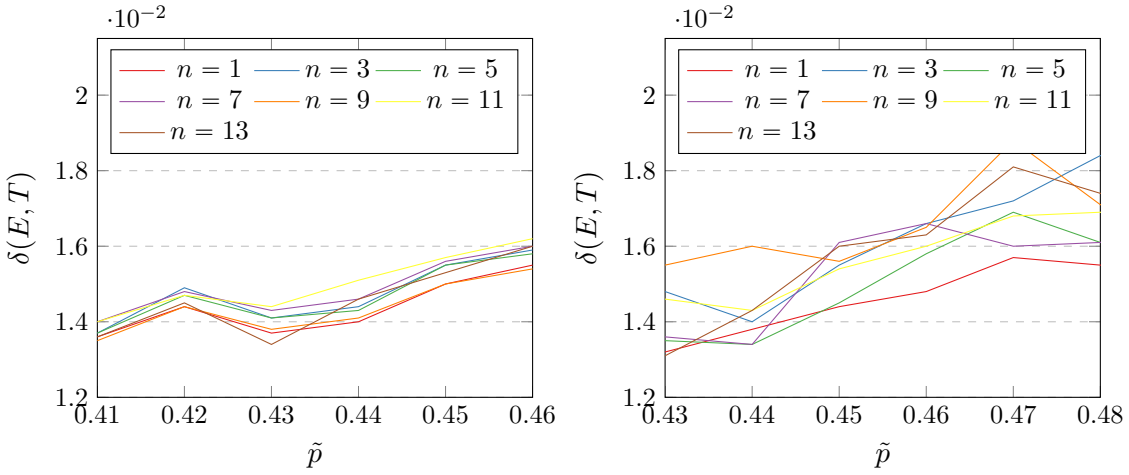
In the case of greedy amplifiers, for $\tilde{p} \geqslant 0.46$ we cannot reliably detect the drift from 0.5 ($P_{pass} > 0.99$). Let $n = 11, 13$. For $\tilde{p} \in [0.44, 0.45]$, according to Table 6.15a, we



(A) Greedy amplifiers

(B) Von Neumann amplifiers

FIGURE 6.15: Kullback-Leibler divergence



(A) Greedy Amplifiers (B) Von Neumann Amplifiers

FIGURE 6.16: Total variance distance

can detect the drift from 0.5 as long as the source is stable (*i.e.* $\hat{p} \leqslant 0.45$). When $\tilde{p} \in [0.41, 0.44)$, $\hat{p}$ can drift with 0.01 and we still have $P_{pass} \leqslant 0.97$. Thus, if we use $n = 11, 13$ $A_t(\tilde{p})$ enables us to have an early detection mechanism for catastrophic RNG failure (*i.e.* $\tilde{p} \leqslant 0.45$).

In the case of Von Neumann amplifiers, for $\tilde{p} = 0.49$ we cannot reliably detect the drift from 0.5 ($P_{pass} > 0.99$), while for $\tilde{p} \in [0.41, 0.42]$ and $n \geqslant 8$ we have $P_{pass} \simeq 0.00$. When $\tilde{p} = 0.48$ and $n > 8$, according to Table 6.15b, we can detect the drift from 0.5 as long as the source is stable. In the case $\tilde{p} = 0.47$ we can detect the drift from 0.5 when the source is stable and $n = 8, 10$, while for $n = 12, 14$ $\hat{p}$ can drift with 0.01 and we still have $P_{pass} \leqslant 0.97$. Let $n = 8, 10, 12, 14$. For $\tilde{p} \in [0.43, 0.46]$, $\hat{p}$ can drift with 0.01 and we still have $P_{pass} \leqslant 0.97$. Thus, if we use $n = 10, 12, 14$ $A_t(\tilde{p})$ enables us to have an early detection mechanism for catastrophic RNG failure (*i.e.* $\tilde{p} \leqslant 0.48$). Note that although Von Neumann amplifiers have a larger range for detecting deviations from 0.5, greedy amplifiers have faster testing times.

6.2.8 Future Work

A possible future direction would be to extended our results to other randomness extractors. Of particular interest, is finding a method to turn a block cipher or a hash function[23] into an amplifier.

Bias is not the only way for a RNG to go wrong. Another important feature that can deviate is correlation. Thus, an interesting question is the following: can bias amplifiers

[23] For a formal treatment of how one can use a block cipher or a hash function to extract randomness we refer the reader to [95].

Bit pattern	**Allowable number of occurrences per sample**			
	$\tilde{p} = 0.41$	$\tilde{p} = 0.42$	$\tilde{p} = 0.43$	$\tilde{p} = 0.44$
1	$70 - 141$	$72 - 144$	$74 - 147$	$76 - 149$
01	$43 - 82$	$43 - 81$	$44 - 81$	$44 - 82$
010	$13 - 68$	$13 - 66$	$13 - 66$	$13 - 62$
0110	$1 - 33$	$1 - 34$	$1 - 33$	$1 - 33$
101	$5 - 51$	$5 - 55$	$7 - 55$	$8 - 55$
1001	$1 - 34$	$1 - 34$	$1 - 36$	$1 - 36$

Bit pattern	**Allowable number of occurrences per sample**			
	$\tilde{p} = 0.45$	$\tilde{p} = 0.46$	$\tilde{p} = 0.47$	$\tilde{p} = 0.48$
1	$79 - 151$	$81 - 154$	$85 - 157$	$87 - 161$
01	$44 - 83$	$44 - 84$	$44 - 84$	$45 - 84$
010	$13 - 61$	$12 - 61$	$10 - 60$	$10 - 60$
0110	$1 - 33$	$1 - 33$	$1 - 33$	$1 - 34$
101	$8 - 55$	$8 - 56$	$9 - 57$	$7 - 60$
1001	$2 - 36$	$2 - 35$	$1 - 34$	$1 - 36$

TABLE 6.14: Health bounds for $H_i(\tilde{p})$.

	$\tilde{p} = 0.41$		$\tilde{p} = 0.42$		$\tilde{p} = 0.43$	
	$\hat{p} = 0.41$	$\hat{p} = 0.42$	$\hat{p} = 0.42$	$\hat{p} = 0.43$	$\hat{p} = 0.43$	$\hat{p} = 0.44$
$n = 11$	0.44	0.76	0.67	0.90	0.83	0.97
$n = 13$	0.21	0.56	0.45	0.78	0.69	0.92

	$\tilde{p} = 0.44$		$\tilde{p} = 0.45$	
	$\hat{p} = 0.44$	$\hat{p} = 0.45$	$\hat{p} = 0.45$	$\hat{p} = 0.46$
$n = 11$	0.94	0.99	0.98	0.99
$n = 13$	0.87	0.98	0.95	0.99

(A) Greedy amplifiers

	$\tilde{p} = 0.43$		$\tilde{p} = 0.44$		$\tilde{p} = 0.45$	
	$\hat{p} = 0.43$	$\hat{p} = 0.44$	$\hat{p} = 0.44$	$\hat{p} = 0.45$	$\hat{p} = 0.45$	$\hat{p} = 0.46$
$n = 8$	0.00	0.09	0.05	0.41	0.27	0.77
$n = 10$	0.00	0.00	0.00	0.14	0.07	0.52
$n = 12$	0.00	0.00	0.00	0.02	0.01	0.23
$n = 14$	0.00	0.00	0.00	0.00	0.00	0.05

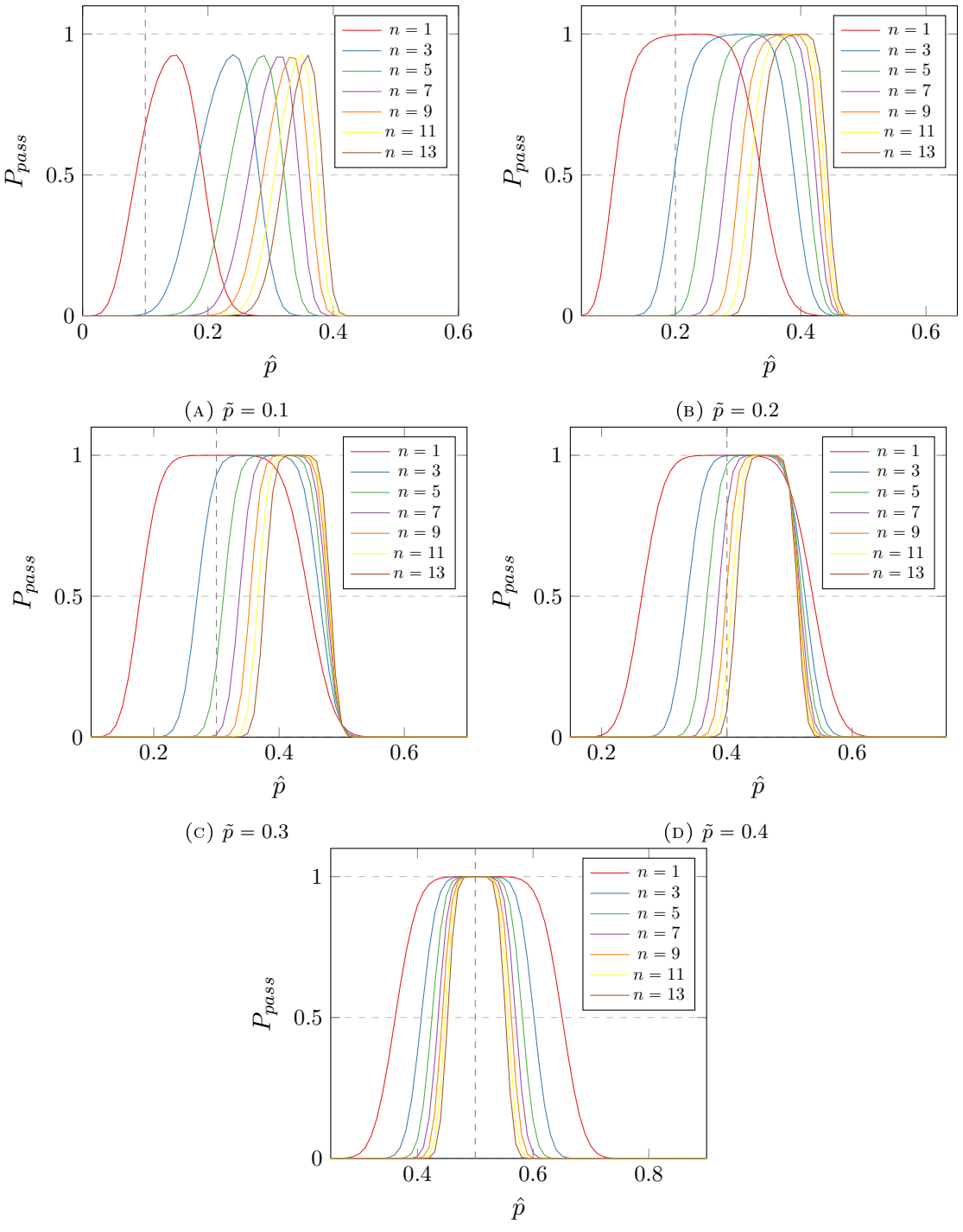
	$\tilde{p} = 0.46$		$\tilde{p} = 0.47$		$\tilde{p} = 0.48$
	$\hat{p} = 0.46$	$\hat{p} = 0.47$	$\hat{p} = 0.47$	$\hat{p} = 0.48$	$\hat{p} = 0.48$
$n = 8$	0.68	0.97	0.90	0.99	0.99
$n = 10$	0.41	0.90	0.78	0.99	0.98
$n = 12$	0.16	0.76	0.57	0.97	0.95
$n = 14$	0.03	0.52	0.32	0.93	0.89

(B) Von Neumann amplifiers.

TABLE 6.15: Approximate theoretical values for P_{pass}.

detect when random data becomes correlated or other classes of amplifiers need to be developed?

The theoretical model presented in this paper is devised only for Intel's health tests. But the architecture presented in Figure 6.9 can be applied to any health test. Thus, an important step into understanding the behavior of bias amplifiers would be to model the architecture's behavior when it is instantiated with other health tests and compare the results with our initial findings.



(A) $\tilde{p} = 0.1$

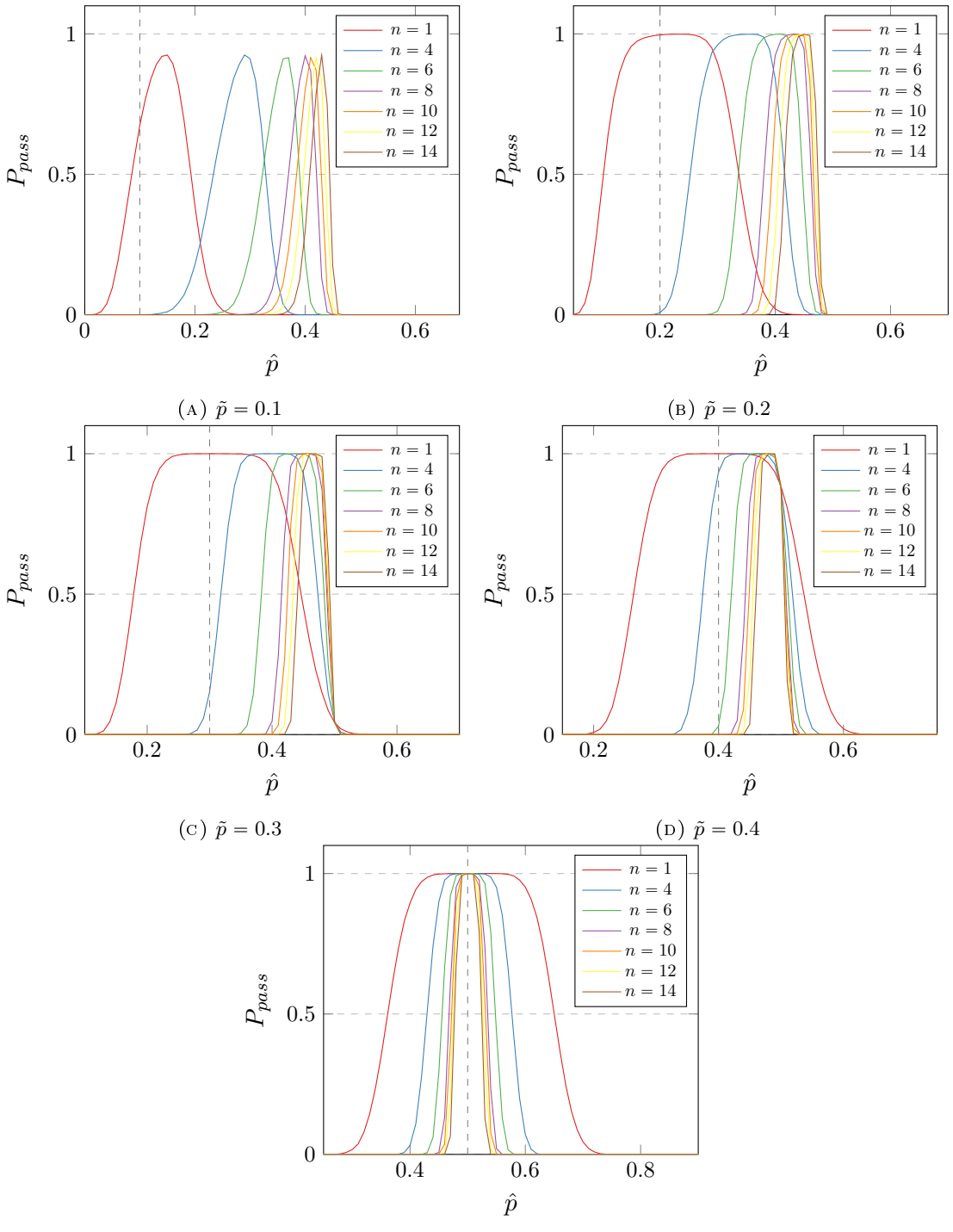
(B) $\tilde{p} = 0.2$

(C) $\tilde{p} = 0.3$

(D) $\tilde{p} = 0.4$

(E) $\tilde{p} = 0.5$

FIGURE 6.17: Experimental results for greedy amplifiers.



(A) $\tilde{p} = 0.1$

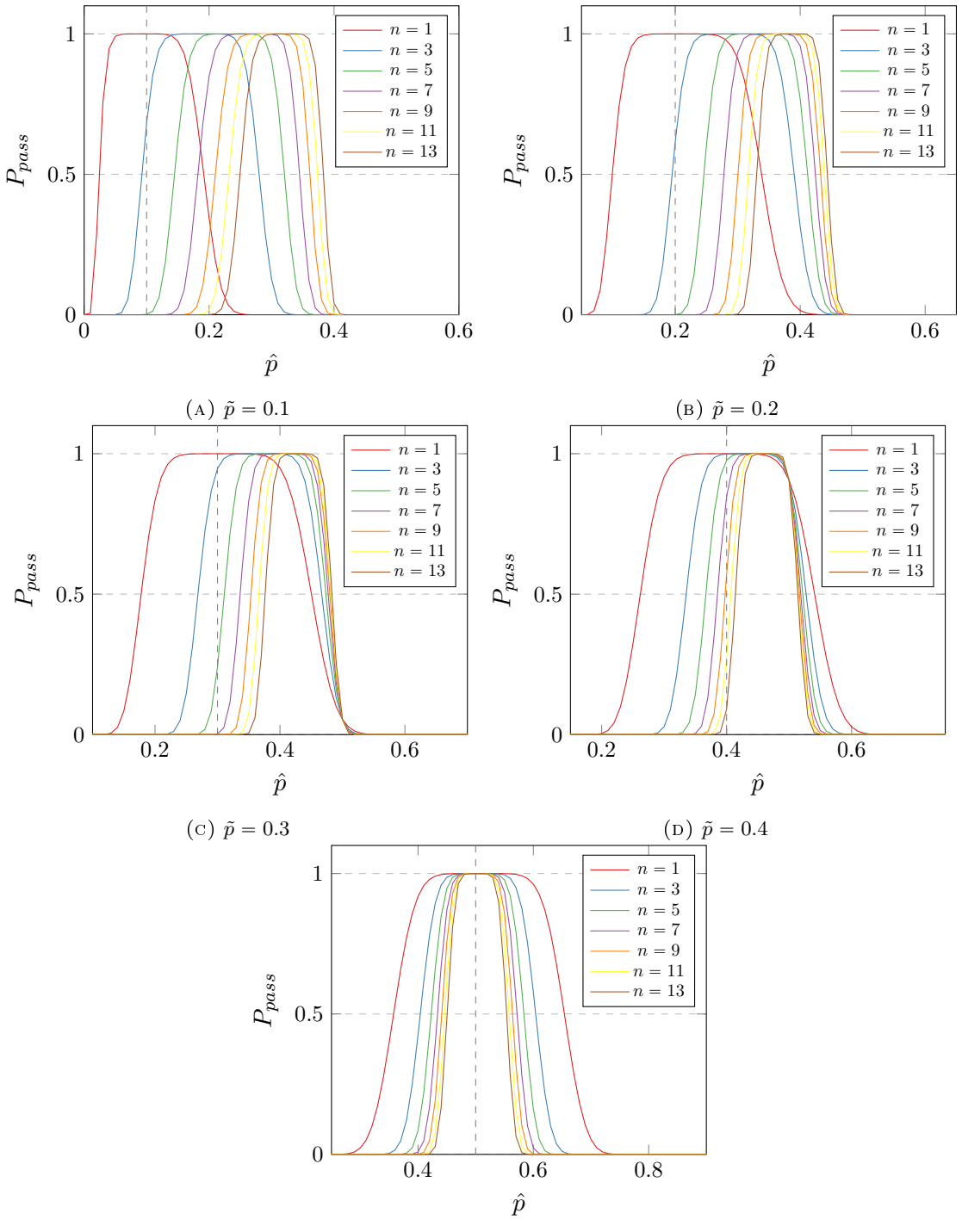
(B) $\tilde{p} = 0.2$

(C) $\tilde{p} = 0.3$

(D) $\tilde{p} = 0.4$

(E) $\tilde{p} = 0.5$

FIGURE 6.18: Experimental results for Von Neumann amplifiers.



(A) $\tilde{p} = 0.1$

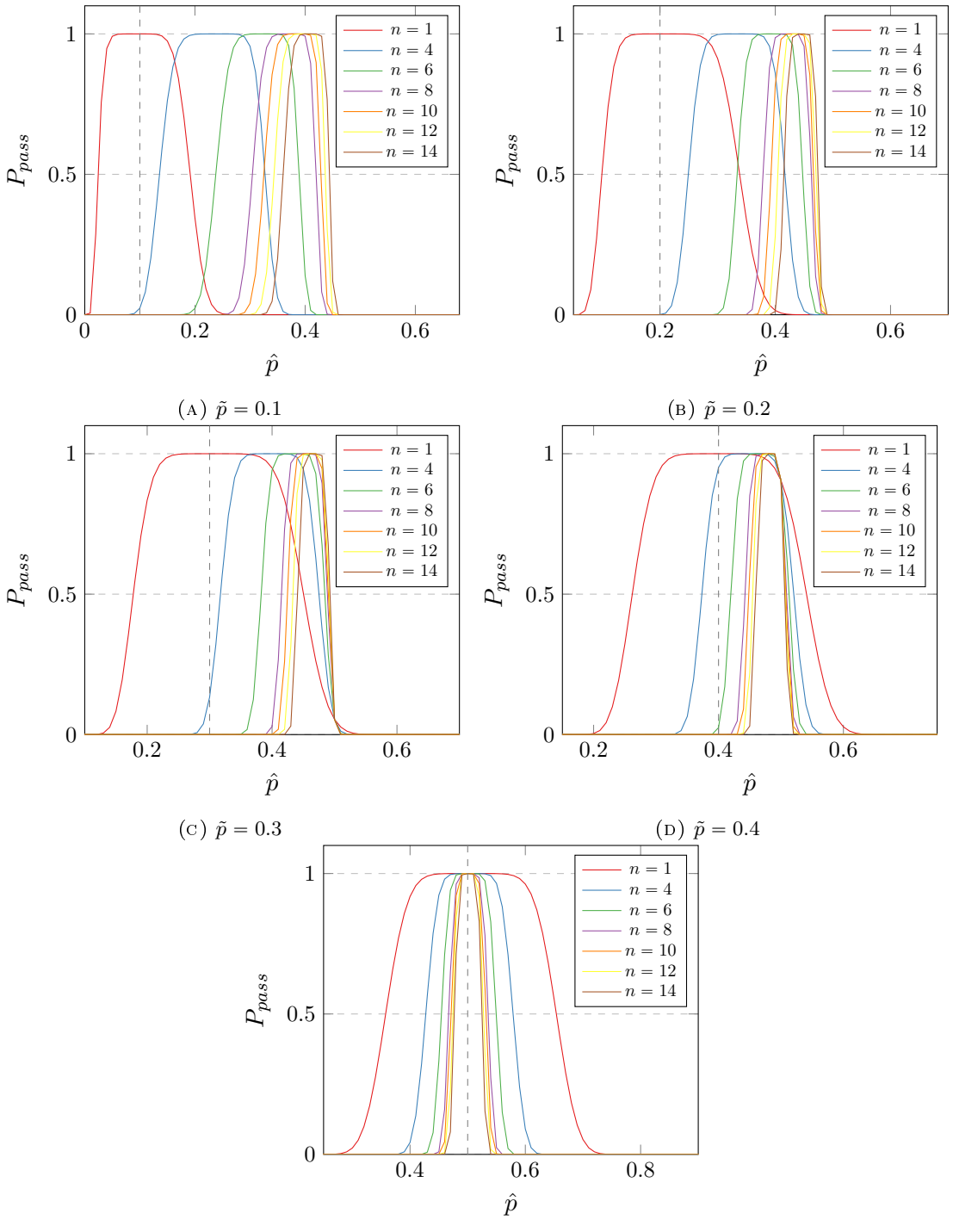
(B) $\tilde{p} = 0.2$

(C) $\tilde{p} = 0.3$

(D) $\tilde{p} = 0.4$

(E) $\tilde{p} = 0.5$

FIGURE 6.19: Theoretical estimates for greedy amplifiers.



(A) $\tilde{p} = 0.1$

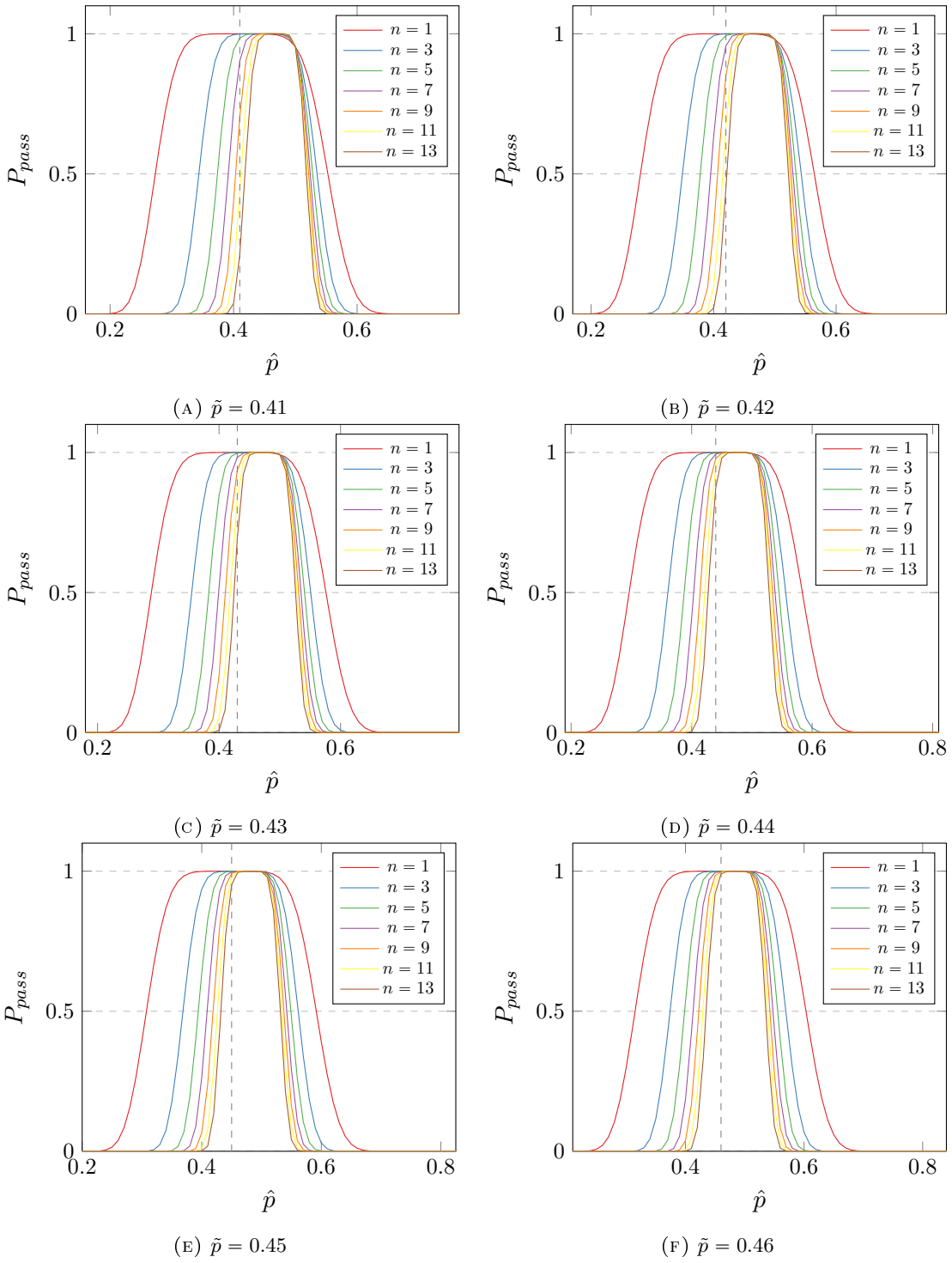
(B) $\tilde{p} = 0.2$

(C) $\tilde{p} = 0.3$

(D) $\tilde{p} = 0.4$

(E) $\tilde{p} = 0.5$

FIGURE 6.20: Theoretical estimates for Von Neumann amplifiers.

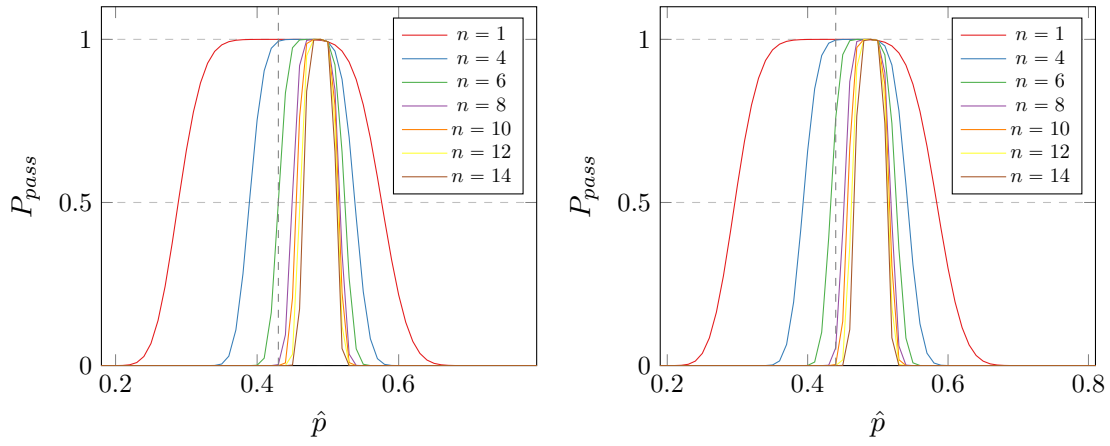


(A) $\tilde{p} = 0.41$ (B) $\tilde{p} = 0.42$

(C) $\tilde{p} = 0.43$ (D) $\tilde{p} = 0.44$

(E) $\tilde{p} = 0.45$ (F) $\tilde{p} = 0.46$

FIGURE 6.21: More theoretical estimates for greedy amplifiers.



(A) Theoretical estimates when $\tilde{p} = 0.43$

(B) Theoretical estimates when $\tilde{p} = 0.44$

(C) Theoretical estimates when $\tilde{p} = 0.45$

(D) Theoretical estimates when $\tilde{p} = 0.46$

(E) Theoretical estimates when $\tilde{p} = 0.47$

(F) Theoretical estimates when $\tilde{p} = 0.48$

FIGURE 6.22: More theoretical estimates for Von Neumann amplifiers.

Chapter 7

Physical Cryptography

In this chapter we present a security analysis to a series of problems that can be seen as abstract games. Our main motivation for studying such protocols is their teaching utility. Note that we are not aware of any real-world application of any sort, as these problems fall in the category of "recreational cryptography". Although recreational, these protocols can provide interesting insight and techniques that can be useful for understanding the concepts on which the underlying problems are based.

Physical cryptography [130, 44, 191, 218] makes use of physical properties of systems for encrypting and/or exchanging information (*i.e.* without using one-way functions). Although a very interesting teaching tool, it can be shown that some of the proposed methods are not safe in practice. Thus, our aim is to attack such physical protocols using methods similar to classical side channel techniques.

Besides the obvious cryptographic teaching utility of physical cryptography schemes, we believe that some of the schemes tackled in the current chapter may be successfully used for introducing concepts corresponding to other domains. We provide the reader with such examples in the following sections.

Although some authors acknowledge that their proposed protocols are only useful for playing with children or introducing new concepts to non-technical audiences, the authors of [129, 130, 128, 225] claim that their schemes can be securely implemented in real-life scenarios. In [81], Courtois attacks one of the protocols proposed in [129], but the authors contest his results in [130]. We independently conducted a simulation of the attack and our results acknowledge Courtois' claim.

Conventions. We denote by U and V the private spaces of Alice and, respectively, Bob. By "impenetrable" we further refer to an object that can not be broken or looked

into no matter the means employed by an adversary. Note that, in practice, "impenetrable" objects do not exist, but we use this concept for presenting the philosophical aspects of different cryptographic problems.

7.1 Yao's Millionaires' Problem

In [260] Yao introduced "Two Millionaires' Problem". The problem can be defined as follows. Alice has a private number a and Bob has a private number b. The goal of the two parties is solving the inequality $a \leqslant b$ without revealing the actual values. We further assume that $a, b \in [0, n]$ are integers.

In [129, 130, 225, 128] the authors present a number of solutions for the previously mentioned problem based on physical principles. In this section we focus on describing their proposed protocols together with our security analyses.

According to the original security model, during the following we consider Alice and Bob as being *honest but curious* users, *i.e.* they can observe, measure and compute whatever they like and try to get a hold on the other party's private numbers while following the protocol's steps.

7.1.1 "Elevator" Solution.

Description. To recall the scheme we follow the descriptions given in [130, 128]. We start by assuming that we have at our disposal a building with at least n floors. Moreover, we consider that the chosen building is equipped with an elevator. Alice positions herself on floor number a while Bob goes to floor number b. Then, Bob takes an elevator (from Bob's private space V) going down and stopping at every floor. Alice watches the elevator doors on her floor, making sure that Bob does not see her if the elevator doors open (here is Alice's private space U). If she sees the elevator doors open, she knows that Bob's number is larger. If not, then his number is smaller. Using such a protocol, Bob will not know the result of the comparison until Alice shares it with him.

Security Analysis. The only security considerations of [130, 128] are that Bob can lock the stairs and disable all elevators except one. This may prevent Alice from cheating by running between different floors to get a better estimate of Bob's number.

During our analysis we found other various attack scenarios. We consider the steps of the protocol as being sequential (*i.e.* first Alice gets to floor a and then Bob gets to floor b).

1. If Alice uses the same elevator as Bob she can simply conceal a small camera[1] while ascending to floor a. Thus, she can recover b as soon as Bob ascends to his designated floor. In order to mitigate such an attack, Bob must be ensured that Alice uses a different elevator or the stairs (*i.e.* making sure that Bob's elevator remains somewhat protected).

2. If the floor doors of Bob's elevator are not secured then Alice can open one of the doors and attach a motion sensor to the elevator. By analyzing the elevator's movement Alice can deduce b. Hence, Bob must be ensured that all the floor doors are secured against unauthorized access.

3. If Alice has access only to the stairs then she can install cameras on each of the n floors[2]. If Bob limits Alice's access to only one floor (a) for security reasons, then he can always check the access readers installed on each floor and find a. These attacks can also be mounted by Alice if Bob takes the stairs. As a result, the only viable solution would be for Alice and Bob to use separate elevators.

4. Once Alice reaches a then she can use a microphone to detect the sound made by the elevator's movement. By counting the number of times the elevator's engine starts or the doors open Alice can deduce b. Hence, to prevent such an attack, Bob can use a device for generating noise in order to mask the other relevant sounds. This attack can also be mounted by Bob for deducing a.

When Alice and Bob simultaneously ascend to their designated floors, the attack scenarios Items 3 and 4 are still feasible.

We do not claim that the protocol is feasible in practice (the doors must be "impenetrable" and the noise source must perfectly mask the sound of the elevator's movement). We only claim that the example can be practically used to introduce Yao's problem to non-specialized audiences and also to make people think of different methods of attacking the system.

7.1.2 "Race Track" Solution.

Description. For recalling the scheme we follow the description from [130]. Let us consider that Alice and Bob have at their disposal a race track of length n. Then, the two parties run toward each other from the opposite ends of the race track, maintaining the speeds of a m/s (Alice), respectively b m/s (Bob). The party which reaches first

[1] We can also consider all types of small devices which incorporate cameras.

[2] If the building already has security cameras, a simpler solution is bribing the security guard and watching the security footage to obtain b.

the midpoint of the track leaves a mark there and runs back, knowing that he/she was faster[3]. When the other party gets to the midpoint, he/she will know that he/she was slower[4]. In order to create their private spaces in this scenario, Alice and Bob have to construct an "impenetrable" fence across the track at the midpoint.

The authors of [130] state that the "race track" idea can be implemented on a computer if two different programs are allowed to work with the same file at the same time. Thus, consider that the shared file is a bit string of length n, with all bits initially equal to 1. Alice provides a program that goes over this bit string left to right, replacing the current 1 symbol by 0 at the speed of one symbol per a time units. Bob provides a similar program going over the same bit string right to left, at the speed of one symbol per b time units. When either of the two programs replaces $n/2$ symbols, it replaces the current symbol by X and stops. In such a way, the two parties will know that whose program stops first has the bigger number. Both programs will have to use the computer's internal clock.

Security Analysis. In [130] the authors mention that the "race track" solution only works if both parties are honest and provide the reader with an attack scenario otherwise. More precisely, the party who reaches the fence first does not run back but just waits to see when the other party arrives, thus figuring out the other party's speed.

During our analysis we found that another restriction must hold. If Alice and Bob run on a circular track when they are "close enough"[5] to the midpoint they will be able to see each other. Thus, even if the parties are honest, the previous attack is still valid. To avoid such a scenario, a possible solution would be to put an "impenetrable"[6] fence such that both private spaces are isolated one from the other and also from the outside world[7].

The digital variant of the "race track" idea on a computer is, unfortunately, flawed. In order for the protocol to be valid both users need read/write access to the file. This implies that any of the parties can choose two positions of the other parties' half of the file, continuously read the symbols corresponding to these positions and record the time needed for the symbols to change. This can be easily extended to monitoring multiple positions. Thus, each user can compute the other party's value.

[3] without knowing the actual speed of the other party
[4] again, without knowing the actual speed of the other party
[5] The precise difference between a and b depends on the race track's radius.
[6] from both a visual and acoustic point of view
[7] If, for example, we isolate the two areas using only a wall, one of the parties can use a drone for spying the other.

Teaching Utility. Although the digital variant is not secure, it can be used by teachers as an implementation task. Thus, students can implement two programs that race each other and also a third program that monitors the speed of either Alice and/or Bob.

7.1.3 "Communicating Vessels" Solution.

Description. To recall the scheme we follow the description from [130]. We start by assuming that Alice has a communicating vessel C_A in her private space U, while Bob has a communicating vessel C_B in his private space V. C_A and C_B are connected by a horizontal pipe attached to their bottoms and, thus, a working system is constructed. The shapes of the vessels are part of the parties' private keys. In the beginning the system is "almost" filled with water. Then, Alice starts pumping the water out of her vessel at the speed of a gallons[8] per second, while Bob starts pumping the water in his vessel at the speed of b gallons per second. The parties are simply watching whether the level of water is decreasing or increasing. If it is decreasing, then $a > b$; if it is increasing, then $a < b$.

Security Analysis. According to the authors of [225] the final level of water in the system depends not only on a and b, but also on the shapes of both vessels. Also, the relation between a and quantities that can be measured outside of Alice's vessel depends on the shape of Alice's vessel, which is unknown to anybody except Alice herself.

During our analysis we observed two main issues of the proposed protocol. First of all, if the participants pump water in and out of the system the shapes of their communicating vessels become irrelevant. In such a case, the authors might have thought about *pouring* water instead of pumping it while constructing their scheme. Secondly, the shapes of the vessels must be considered in such a way that the two parties can precisely measure fluctuations in their corresponding vessels. To explain this type of phenomena we can consider the following exaggerated example: the shapes of Alice and Bob's vessels correspond to those of two small artificial lakes and they pump water in and out with negligible speeds (*e.g.* a milliliter per hour). Then, they can not accurately detect which speed is greater than the other.

The scheme enhanced with our previous comments becomes equivalent with: Alice and Bob have two cylinder shaped vessels such that they can accurately measure fluctuations of the system. To detect Alice's value, Bob can use a graduated cylinder and measure the volume's fluctuation. Then, using his own speed value b he can compute a. Hence, the

[8] or whatever units

scheme is insecure for solving Yao's problem but it can be used as a public key encryption scheme (see Appendix M).

Teaching Utility. Communicating vessels are a common example in physics teaching (see for example [133]). More precisely, the scheme provides a good opportunity for a teacher to introduce students to the dynamics of (ideal) fluids.

7.1.4 "Rope" Solution.

Description. For recalling the scheme we follow the description given in [129]. Alice and Bob privately select $c < 0$ and, respectively, $d > 0$. We position Alice and Bob in a plane, Alice at point $A = (a, c)$ and Bob at point $B = (b, d)$. Also, we give them both long pieces of rope. We assume that the scaling is such that Alice and Bob cannot see each other's point.

First, Alice fixes one end of her rope at point A and selects as her private space U a neighborhood of point A that cannot be seen by Bob. Bob, too, selects V as a neighborhood of his point B. Then, Alice fixes the other end of her rope to a random point C in the plane, far enough so that her neighborhood U can not be seen from C. After fixing the rope, she positions the part of the rope inside U so that this part is not a straight line. She then communicates the coordinates of point C to Bob.

Bob walks to point C, ties one end of his rope to Alice's rope, then walks back to his point B, while unwinding (not pulling) his rope along the way. When Bob reaches his B, he starts pulling the rope until Alice tells him to stop, which is as soon as Alice sees that the part of the rope inside her neighborhood U is a straight line. To make sure that it is not by accident that the part of the rope inside her neighborhood U is a straight line, Alice asks Bob whether or not the part of the rope inside his neighborhood V is a straight line. If it is not, then Alice starts pulling her end of the rope toward her point A until Bob tells her to stop, which is as soon as Bob sees that the part of the rope inside his neighborhood V is a straight line.

When the parts of the rope inside both neighborhoods U and V are straight, Alice and Bob assume that their points A and B are connected by a straight rope, and they find the slope s of the corresponding straight line by selecting any two points on the parts of the line inside their private neighborhoods. Then, $a < b$ if and only if $s > 0$.

Security Analysis. Some parts of the scheme described in [129] may seem redundant according to the authors. As pointed out by them, if both parties are honest the protocol

can be simplified. To mitigate dishonest parties attacks, *e.g.* Alice must tell Bob to stop as soon as she sees that the part of the rope inside her neighborhood U is a straight line. Otherwise, Bob could triangulate Alice's point A by straightening the rope between A and two different points of his choice.

Since we do not consider the honest but curious attack model for this precise protocol, another simple attack can be mounted. Bob can walk along Alice's rope until he is able to determine the coordinates of point A. To prevent Alice from seeing Bob while he tries to find A, he can use, for example, either a small drone or a powerful telescopic sight. To avoid such a vulnerability of the protocol, the neighborhood U must be covered by an "impenetrable" material and, also, to contain a large number of points such that it is impossible for Bob to determine the exact position of A. When selecting the number of points in U we also need to take into account the following scenario. After determining the precise position of U in the plane Bob gets back to point C and follows the initial protocol for determining s. Then, Bob can narrow down the number of possibilities for A.

Teaching Utility. A variation of this protocol for key exchange may be the following. Ted, a trusted third party, takes an infinite rope and fixes one end of it at Alice's point A. Similarly, Ted fixes another rope at Bob's point B. After fixing the ropes, Ted walks to a random point T such that the distance to A and B is equal and then cuts the ropes at point T. In the last step of the protocol Ted returns the ropes to Alice and, respectively, Bob. The common key is the length of the two ropes.

Besides a good reason for a discussion about analytic geometry, this variations of the protocol can be the starting point for describing the secure key exchange protocol for the Internet of Things networks introduced in [195].

7.1.5 "Laboratory Scale" Solution.

Description. To recall the scheme we follow the description from [128]. We assume that Alice and Bob have access to a laboratory scale[9]. Each of the two parties manufacture a weight corresponding to their private number (*e.g.* in grams). We also assume that they have identical boxes[10] where each of them can put their corresponding weight. Alice enters the room where the scale is positioned and puts her box on one of the plates. Then, Bob enters and puts his box on the other plate. If his plate goes down, then his number is larger; otherwise, it is Alice's number that is larger.

[9] a simple mechanism with two plates that are in balance when no weight is placed on either of them
[10] which, in this case, are considered their private spaces

Security Analysis. The authors argue in [128] that Alice and Bob do not have to be in the same place at the same time to perform the comparison, but they still have to be in the same place at some point, which may be inconvenient. In fact, if, say, Alice is worried about Bob cheating (by putting different weights on his plate to zoom in on Alice's weight), then she would have to stay in the room and watch what Bob is doing.

Note that when we analyzed the solution we assume that the box is "impenetrable". Compared to the "rope" solution where Bob needs to cheat in order to detect the dimensions of U, here Bob knows the precise size of the covering box. This gives him an upper limit of the weight's volume. If he knows the material of the weight, then he has an upper limit of the value a. This could be easily mitigated by keeping the weight's material secret.

7.2 Comparing Information Without Revealing It

The initial problem from which the study in [102] started is the following. Charlie complains to one of his managers, Alice, about a sensitive matter and asks her to keep it secret. A few months later, another manager, Bob, tells Alice that someone complained to him, also with a confidentiality request, about the same matter. Alice and Bob need a way to determine if the same person complained to them without revealing the identity of the complainer. The authors of [102] describe a series of complex protocols that try to accomplish this task. But, the simplest solution was actually provided by the 13 year old son of the first author: "Why not just ask Charlie whether he complained to Bob?". This proves that sometimes experts try to find too complicated solutions for simple things.

We further present a few solutions that can still work when implemented using our current technology. A legacy example may be considered the "airline reservation" solution. While Bob is not in the same room Alice calls a specific airline and makes a particular reservation in the name of her complainer. Then, Bob tries to cancel the reservation in his complainer's name. Finally, Alice cancels or tries to cancel the reservation she made. It is obvious that nowadays such a version of the protocol can not be functional anymore, due to the fact that in order to cancel a reservation one needs to have extra pieces of information (*e.g.* the reservation code).

For uniformity, we consider, as in Section 7.1, that Alice and Bob are honest but curious.

7.2.1 Message for Bob

Description. We assume that Alice and Bob associate each candidate with a random telephone number. Alice dials the number[11] assigned to the person who complained to her (Charlie) and asks to leave a message for Bob. It is clear that the one answering the phone does not know who Bob is. A while after, Bob dials the number of the person who complained to him and asks if anyone has left him a message.

Security Analysis. The authors of [102] provide a short security analysis. More precisely: ① if Alice does not supervise Bob, then Bob might try several candidates and ② Dave might deny that a message was left for Bob.

The protocol was designed in a period of time in which telephones were only analog. But, nowadays, we also have digital and mobile phones. Thus, we further consider all the three cases when analyzing the security of the scheme. If Alice and Bob use the same phone to run the protocol, then, in the digital and mobile cases, Bob can check the call history of the phone to find out the identity of the complainer. Thus, to prevent such an attack, Alice must delete the call history. Even if she does this, there is a small probability that Dave will call back and, if Bob, is near the phone at that particular time, he can see the phone number and deduce the identity of the complainer. This problem can be easily rectified if Alice hides her number. Note that the previously mentioned problems do not happen in the analog case.

If Alice and Bob use different analog phones and Bob is nearby, he can redial the last number and ask Dave which is his phone number. Thus, in the analog case Alice needs to call another number afterwards[12]. In the digital case, Alice simply has to delete the call history to avoid the redialing attack. If the protocol is run using mobile phones, such an attack is even harder because Bob has to physically take Alice's phone. Even if he manages to snatch Alice's phone, the device might be locked.

We conclude that in the analog case either version is secure (*i.e.* with one or two phones) as long as Alice overwrites the call logs, while in the digital case it is better to use two phones. We believe that the protocol is secure as long as the initial scenario is valid[13] and our proposed countermeasures are taken into account.

[11] We denote the owner by Dave.
[12] to overwrite the call history
[13] A powerful enough Bob can always eavesdrop the landline or ask the operator for Alice's call history.

7.2.2 Password

Description. We assume that Alice chooses to change her password in accordance with Charlie's name. Next, Bob tries to log in as Alice. In order to do so, Bob uses the name of the person who complained to him as a password.

Security Analysis. As in Section 7.2.1, Bob might try several candidates [102]. Additional to the initial security analysis, there is always the possibility that Alice installs either a key logger on the computer or a video camera inside the room and directly finds out Bob's password. Thus, the protocol is insecure.

Teaching Utility. In one version of the protocol, the authors of [102] suggest using the "passwd" Linux command to run the scheme. This provides a good opportunity for a teacher to introduce students to the Linux terminal basics and also how passwords are stored in Linux.

7.2.3 Cups

Description. We start by assuming that we have a small number s of candidates. Alice and Bob get s identical containers (*e.g.* by acquiring disposable cups), line them up and label them[14]. Then, Alice puts a folded slip of paper saying "yes" in the cup of Charlie and a slip saying "no" in the other $s-1$ cups. Bob does the same. Next, Alice and Bob remove the labels and shuffle the cups. To complete the protocol, both the parties look inside the cups to see whether one of them contains two slips saying "yes".

Security Analysis. If Alice and Bob use the suggested containers, Bob can always check which cup contains the slip saying "yes". Thus, it is better to use secure containers, for example ballot boxes which are tamper-evident. Hence, even if Bob manages to break into all the secure containers, Alice can detect that Bob cheated.

Teaching Utility. The secure version of the protocol may be seen as a toy version of the voting process. Thus, it can be used as an introduction to elections and electoral fraud.

[14] one for each candidate

7.3 Public Key Encryption

Several public key cryptosystems based on different laws of physics[15] can be found in [130][16]. Although these solutions are hard to implement in the real world[17], they provide a very good teaching tool. More precisely, a teacher can interactively transition from these toy protocols to precise explanations of the underlying physical laws.

Given the attack possibilities we observed while analyzing the schemes in [130], we chose to only discuss the "capacitors" solution during the following.

7.3.1 "Capacitors" Solution.

Description. Assume that Alice wishes to send a secret positive number q_a to Bob. Let us consider that Alice has a capacitor C_1 of the capacitance c_A (denoting her *public key*) and charge q_A (denoting her *secret message*) in U. Similarly, Bob has a capacitor C_2 of the capacitance c_B (denoting his *long-term private key*) and a randomly chosen charge q_B (denoting his *session private key*) in V. Note that the private key is selected by Bob randomly before each transmission from Alice. The capacitors are connected in such a way that the plates holding the positive charges are connected by one wire, and the plates holding the negative charges are connected by another wire (see Figure 7.1). Alice has a switch that keeps the circuit disconnected until the actual transmission begins. Also, Alice has an ammeter to monitor the electric current in the circuit. Bob has a rheostat included in the circuit in V. This allows him to randomly change the resistance of the whole circuit, and therefore also to change parameters of the electric current during transmission.

According to the authors, Alice uses her switch to connect the circuit, starting the redistribution of the electric charges between the two capacitors. When this process is complete, she disconnects the circuit. After redistribution of charges, both Alice and Bob, have new charges: Q_A and Q_B. Now, all that Bob has to do in order to compute the secret of Alice is to apply the following mathematical expression: $q_A = Q_B \cdot (1 + \frac{c_A}{c_B}) - q_B$.

Security Analysis. To promote an idea which might be relevant in practice, some experimental results should be presented. In this case, the authors gave an example of a system used for information transmission based on physical properties of passive components. Although the authors are theoretically right, Courtois contested the strength

[15] We refer the reader to Appendix M.
[16] A similar solution for Yao's problem is described in [128].
[17] The authors assume that only Alice and Bob interferes with the system.

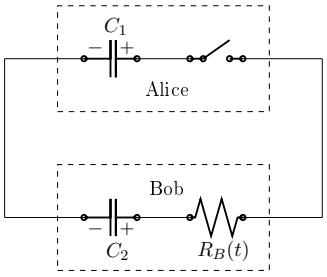


FIGURE 7.1: "Capacitors" solution

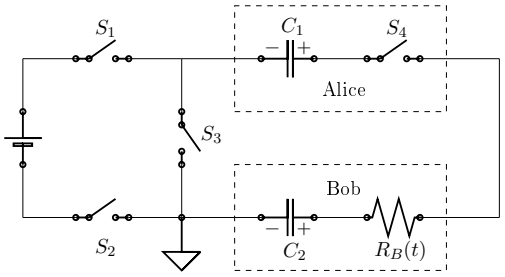


FIGURE 7.2: Proposed "Capacitors" solution

of their model in [81]. In our analysis, we propose a complete, yet simple way to demonstrate both theories. The proposed scheme is represented in Figure 7.2. In order to do so, we extended the electrical circuit proposed in [129] so that we could prove its functionality by simulating it. Based on the fact that the authors gave no technical specifications regarding the circuit, we analyzed several scenarios. The first one concerns the type of capacitors used in the circuit. We tested the scheme using polarized and non-polarized capacitors with specific given input values and concluded that, in simulation, the differences are not significant. Nevertheless, in practice, the type of capacitor used is very important in order to avoid damaging the circuit.

To ease description, in order to validate the functionality of the "capacitors" solution we randomly choose a set of parameters for the scheme. Our example can directly be used in class to experimentally show that the solution is a viable one.

For obtaining a functional "capacitors" solution, we propose adding a power supply and 3 more switches (see Figure 7.2). The voltage generated by the power supply is 1 V. We use a 10 μF capacitance for Alice's capacitor and a 1 μF capacitance for Bob's capacitor. The rheostat is set at $R_1 = 431\ \Omega$ and $R_1 = 569\ \Omega$. The simulation is done using the electronic circuit simulator hosted by [5]. The first step of the simulation consists of charging the capacitors, in order to obtain the initial values for the electric charges. For charging the capacitors, switches S_1, S_2 and S_4 must be connected. After this step, the power supply is disconnected and the circuit is closed, meaning that switches S_1 and S_2 must be disconnected and switch S_3 must be connected. Switch S_4 is Alice's switch. Based on the values that were set as input, we measured the voltage drop V_d on each capacitor and obtained the initial electric charges $q_A = 899.09\ nC$ ($V_{d_A} = 89.909\ mV$) and $q_B = 910.091\ nC$ ($V_{d_B} = 910.091\ mV$). After re-distributing charges (*i.e.* when Alice connects the circuit) the charges become $Q_A = 10\ nC$ ($V_{d_A} = 1\ mV$) and $Q_B = 1\ nC$ ($V_{d_B} = 1\ mV$). In the final step of the protocol, Bob computes Alice's electric

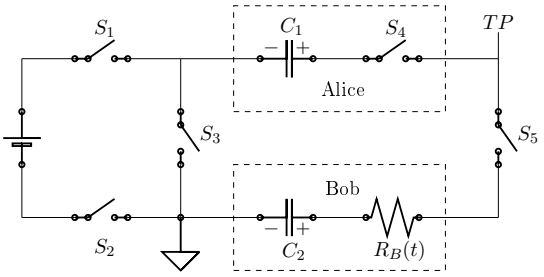


FIGURE 7.3: Attack scenario "Capacitors" solution

charge:

$$\begin{aligned} q_A &= Q_B \cdot (1 + \frac{c_A}{c_B}) - q_B \\ &= 10 \cdot 10^{-9} \cdot (1 + \frac{10 \cdot 10^{-6}}{10^{-6}}) - 910.091 \cdot 10^{-9} \\ &= -899.091 \cdot 10^{-9}\ C \end{aligned}$$

In [81], Courtois presents a rather intrusive attack in which Eve inserts a switch between Alice and Bob and measures the voltage (see Figure 7.3). In this case, switches S_1 and S_2 are disconnected. Switch S_3 is connected, Alice's switch is S_4 and Eve's switch is S_5. S_4 and S_5 are disconnected. Eve measures the voltage between Alice and Bob, right after Alice connects her switch. After the measurement, Eve connects her switch too. This is a very simple way to determine V_{d_A}. Since Alice's capacitance is a public parameter, Eve just computes:

$$\begin{aligned} q_A &= c_A \cdot V_{d_A} \\ &= 10 \cdot 10^{-6} \cdot 89.909 \cdot 10^{-3} \\ &= 899.09 \cdot 10^{-9}\ C \end{aligned}$$

After running the simulation, we observed that the attack scenario is a plausible one. Note that the detection of Eve's attack depends on the quality of the equipment that she possesses.

Initially, for protecting the circuit we thought of adding a plus of security by connecting each capacitor to a different power supply. It turned out this is not enough, since Eve can measure the circuit in any point which surrounds each Alice's and Bob's private space. Thus, we dropped the idea and choose the simpler version of the two.

Chapter 8

Appendices

A Letter Frequencies

To have uniform letter frequency tables, we added the probability of letters with diacritical marks to the probability of their base letter. For example, in Danish, the letter O has a 0.0464 occurrence probability and the letter Ø one of 0.0094. We added the two and we recorded O's probability as 0.0558. Note that the frequency tables we used for computing our tables are from [170].

A, Å, Æ	0.0809	H	0.0162	O, Ø	0.0558	V	0.0233
B	0.0200	I	0.0600	P	0.0176	W	0.0007
C	0.0056	J	0.0073	Q	0.0001	X	0.0003
D	0.0586	K	0.0339	R	0.0896	Y	0.0070
E	0.1545	L	0.0523	S	0.0581	Z	0.0003
F	0.0241	M	0.0324	T	0.0686		
G	0.0408	N	0.0724	U	0.0198		

Table A.1: Relative frequencies of Danish letters.

A	0.0855	H	0.0496	O	0.0747	V	0.0106
B	0.0160	I	0.0733	P	0.0207	W	0.0183
C	0.0316	J	0.0022	Q	0.0010	X	0.0019
D	0.0387	K	0.0081	R	0.0633	Y	0.0172
E	0.1210	L	0.0421	S	0.0673	Z	0.0011
F	0.0218	M	0.0253	T	0.0894		
G	0.0209	N	0.0717	U	0.0268		

Table A.2: Relative frequencies of English letters.

A, Ä	0.1580	H	0.0185	O, Ö	0.0605	V	0.0225
B	0.0028	I	0.1082	P	0.0184	W	0.0009
C	0.0028	J	0.0204	Q	0.0001	X	0.0003
D	0.0104	K	0.0497	R	0.0287	Y	0.0174
E	0.0797	L	0.0576	S	0.0786	Z	0.0005
F	0.0019	M	0.0320	T	0.0875		
G	0.0039	N	0.0883	U	0.0501		

TABLE A.3: Relative frequencies of Finnish letters.

A, À, Â	0.0808	H	0.0093	O, Ô, Œ	0.0546	V	0.0129
B	0.0096	I, Î, Ï	0.0726	P	0.0298	W	0.0008
C, Ç	0.0344	J	0.0030	Q	0.0085	X	0.0043
D	0.0408	K	0.0016	R	0.0686	Y	0.0034
E, È, É, Ê	0.1745	L	0.0586	S	0.0798	Z	0.0010
F	0.0112	M	0.0278	T	0.0711		
G	0.0118	N	0.0732	U, Ù, Û, Ü	0.0559		

TABLE A.4: Relative frequencies of French letters.

A, Ä	0.0688	H	0.0411	O, Ö	0.0299	V	0.0094
B	0.0221	I	0.0760	P	0.0106	W	0.0140
C	0.0271	J	0.0027	Q	0.0004	X	0.0007
D	0.0492	K	0.0150	R	0.0771	Y	0.0013
E	0.1599	L	0.0372	S, ß	0.0656	Z	0.0122
F	0.0180	M	0.0275	T	0.0643		
G	0.0302	N	0.0959	U, Ü	0.0376		

TABLE A.5: Relative frequencies of German letters.

A, Ą	0.0997	H	0.0125	O, Ó	0.0879	V	0.0000
B	0.0139	I	0.0809	P	0.0292	W	0.0478
C, Ć	0.0422	J	0.0226	Q	0.0000	X	0.0000
D	0.0323	K	0.0354	R	0.0506	Y	0.0370
E, Ę	0.0849	L, Ł	0.0418	S, Ś	0.0504	Z, Ź, Ż	0.0590
F	0.0041	M	0.0273	T	0.0394		
G	0.0154	N, Ń	0.0602	U	0.0259		

TABLE A.6: Relative frequencies of Polish letters.

A	0.1250	H	0.0081	O	0.0898	V	0.0098
B	0.0127	I	0.0691	P	0.0275	W	0.0003
C	0.0443	J	0.0045	Q	0.0083	X	0.0019
D	0.0514	K	0.0008	R	0.0662	Y	0.0079
E	0.1324	L	0.0584	S	0.0744	Z	0.0042
F	0.0079	M	0.0261	T	0.0442		
G	0.0117	N, Ñ	0.0731	U	0.0400		

TABLE A.7: Relative frequencies of Spanish letters.

A, Ä, Å	0.1252	H	0.0209	O, Ö	0.0579	V	0.0242
B	0.0154	I	0.0582	P	0.0184	W	0.0014
C	0.0149	J	0.0061	Q	0.0002	X	0.0016
D	0.0470	K	0.0314	R	0.0843	Y	0.0071
E	0.1015	L	0.0528	S	0.0659	Z	0.0007
F	0.0203	M	0.0347	T	0.0769		
G	0.0286	N	0.0854	U	0.0192		

TABLE A.8: Relative frequencies of Swedish letters.

B Vigenère Cryptanalysis

In [233], the author describes an algorithm for breaking the Vigenère cipher. Because of better results in practice, we changed the scoring function from [233] with a scoring function based on digraphs. The result is presented in Algorithm 51.

C Grain v1

In the case of Grain v1, $n = 80$ and $m = 64$. The padding value is $P =$ `0xffff`. The values IV and P are loaded in the LFSR using the function $LoadIV(IV, P) = IV \| P$. Given $S \in \{0,1\}^{80}$, we define $ExtractIV(S) = MSB_{64}(S)$.

We denote by $f_1(x)$ the primitive feedback of the LFSR:

$$f_1(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80}.$$

Algorithm 51. The algorithm for breaking the Vigenère cipher with key length *matrix_size*.

Input: The ciphertext *enc*.
Output: The best possible message *best_msg* and its associated score *best_score*.

```
1  Function decrypt_vigenere(enc, key):
2      fragment[matrix_size] ← {""};
3      for i ∈ [0, enc.size()/matrix_size) do
4          for j ∈ [0, matrix_size) do
5              fragment[j] +=
                (enc[i · matrix_size + j] − "a" + key) mod alphabet_size + "a";
6          end
7      end
8      return fragment;
9  Function compute_score(best_fragment_score):
10     best_score ← 0;
11     for i ∈ [0, matrix_size) do
12         best_score += best_fragment_score[i];
13     end
14     return best_score;
15 Function recompose_msg(best_fragment):
16     best_msg ← "";
17     for i ∈ [0, enc_size / matrix_size) do
18         for j ∈ [0, matrix_size) do
19             best_msg += best_fragment[j][i];
20         end
21     end
22     return best_msg;
23 Function break_vigenere(enc):
24     best_fragment_score[matrix_size] ← {−∞};
25     for key ∈ [0, alphabet_size) do
26         fragment ← decrypt_vigenere(enc, key);
27         for i ∈ [0, matrix_size) do
28             fragment_score[i] ←
                score_function(fragment[i], digraph_freq, digraph_default, 2);
29             if fragment_score[i] > best_fragment_score[i] then
30                 best_fragment_score[i] ← fragment_score[i];
31                 best_fragment[i] ← fragment[i];
32             end
33         end
34         best_score ← compute_score(best_fragment_score);
35         best_msg ← recompose_msg(best_fragment);
36     end
37     return (best_score, best_msg);
```

We denote by $g_1(x)$ the nonlinear feedback polynomial of the NFSR:

$$\begin{aligned} g_1(x) = 1 &+ x^{18} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52} + x^{59} + x^{66} + x^{71} + x^{80} \\ &+ x^{17}x^{20} + x^{43}x^{47} + x^{65}x^{71} + x^{20}x^{28}x^{35} + x^{47}x^{52}x^{59} + x^{17}x^{35}x^{52}x^{71} \\ &+ x^{20}x^{28}x^{43}x^{47} + x^{17}x^{20}x^{59}x^{65} + x^{17}x^{20}x^{28}x^{35}x^{43} + x^{47}x^{52}x^{59}x^{65}x^{71} \\ &+ x^{28}x^{35}x^{43}x^{47}x^{52}x^{59}. \end{aligned}$$

The boolean filter function $h_1(x_0, \ldots, x_4)$ is

$$\begin{aligned} h_1(x_0, \ldots, x_4) &= x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 + x_0x_2x_3 + x_0x_2x_4 \\ &+ x_1x_2x_4 + x_2x_3x_4. \end{aligned}$$

The output function is

$$z_i^1 = \sum_{j \in \mathcal{A}_1} x_{i+j} + h_1(y_{i+3}, y_{i+25}, y_{i+46}, y_{i+64}, x_{i+63}),$$

where $\mathcal{A}_1 = \{1, 2, 4, 10, 31, 43, 56\}$.

Algorithm 52. KSA^{-1} routine for Grain v1.

Input: State $S_i = (x_0, \ldots, x_{79}, y_0, \ldots, y_{79})$
Output: The preceding state $S_{i-1} = (x_0, \ldots, x_{79}, y_0, \ldots, y_{79})$

$v = y_{79}$ and $w = x_{79}$
for $t = 79$ to 1 **do**
  $y_t = y_{t-1}$ and $x_t = x_{t-1}$
end
$z = \sum_{j \in \mathcal{A}_1} x_j + h_1(y_3, y_{25}, y_{46}, y_{64}, x_{63})$
$y_0 = z + v + y_{13} + y_{23} + y_{38} + y_{51} + y_{62}$
$x_0 = z+w+y_0+x_9+x_{14}+x_{21}+x_{28}+x_{33}+x_{37}+x_{45}+x_{52}+x_{60}+x_{62}+x_{63}x_{60}+x_{37}x_{33}+ x_{15}x_9 + x_{60}x_{52}x_{45} + x_{33}x_{28}x_{21} + x_{63}x_{45}x_{28}x_9 + x_{60}x_{52}x_{37}x_{33} + x_{63}x_{60}x_{21}x_{15} + x_{63}x_{60}x_{52}x_{45}x_{37} + x_{33}x_{28}x_{21}x_{15}x_9 + x_{52}x_{45}x_{37}x_{33}x_{28}x_{21}$

D Grain-128

In the case of Grain-128, $n = 128$ and $m = 96$.The padding value is $P =$ `0xffffffff`. The values IV and P are loaded in the LFSR using the function $LoadIV(IV, P) = IV \| P$. Given $S \in \{0,1\}^{128}$, we define $ExtractIV(S) = MSB_{96}(S)$.

We denote by $f_{128}(x)$ the primitive feedback of the LFSR:

$$f_{128}(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128}.$$

We denote by $g_{128}(x)$ the nonlinear feedback polynomial of the NFSR:

$$\begin{aligned} g_{128}(x) = 1 &+ x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{44}x^{60} + x^{61}x^{125} \\ &+ x^{63}x^{67} + x^{69}x^{101} + x^{80}x^{88} + x^{110}x^{111} + x^{115}x^{117}. \end{aligned}$$

The boolean filter function $h_{128}(x_0, \ldots, x_8)$ is

$$h_{128}(x_0, \ldots, x_8) \quad = \quad x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8.$$

The output function is

$$z_i^{128} = \sum_{j \in \mathcal{A}_{128}} x_{i+j} + y_{i+93} + h_{128}(x_{i+12}, y_{i+8}, y_{i+13}, y_{i+20}, x_{i+95}, y_{i+42}, y_{i+60}, y_{i+79}, y_{i+95}),$$

where $\mathcal{A}_{128} = \{2, 15, 36, 45, 64, 73, 89\}$.

Algorithm 53. KSA^{-1} routine for Grain-128.

Input: State $S_i = (x_0, \ldots, x_{127}, y_0, \ldots, y_{127})$
Output: The preceding state $S_{i-1} = (x_0, \ldots, x_{127}, y_0, \ldots, y_{127})$

$v = y_{127}$ and $w = x_{127}$
for $t = 127$ to 1 **do**
  $y_t = y_{t-1}$ and $x_t = x_{t-1}$
end
$z = \sum_{j \in \mathcal{A}_{128}} x_{i+j} + y_{93} + h_{128}(x_{12}, y_8, y_{13}, y_{20}, x_{95}, y_{42}, y_{60}, y_{79}, y_{95}),$
$y_0 = z + v + y_7 + y_{38} + y_{70} + y_{81} + y_{96}$
$x_0 = z + w + y_0 + x_{26} + x_{56} + x_{91} + x_{96} + x_{84}x_{68} + x_{65}x_{61} + x_{48}x_{40} + x_{59}x_{27} + x_{18}x_{17} + x_{13}x_{11} + x_{67}x_3$

E Grain-128a

In the case of Grain-128a, $n = 128$ and $m = 96$. The padding value is $P = \texttt{0xfffffffe}$. The values IV and P are loaded in the LFSR using the function $LoadIV(IV, P) = IV \| P$. Given $S \in \{0,1\}^{128}$, we define $ExtractIV(S) = MSB_{96}(S)$.

We denote by $f_{128a}(x)$ the primitive feedback of the LFSR:

$$f_{128a}(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128}.$$

We denote by $g_{128a}(x)$ the nonlinear feedback polynomial of the NFSR:

$$\begin{aligned} g_{128a}(x) = 1 &+ x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{44}x^{60} + x^{61}x^{125} + x^{63}x^{67} + x^{69}x^{101} \\ &+ x^{80}x^{88} + x^{110}x^{111} + x^{115}x^{117} + x^{46}x^{50}x^{58} + x^{103}x^{104}x^{106} + x^{33}x^{35}x^{36}x^{40}. \end{aligned}$$

The boolean filter function $h_{128a}(x_0, \ldots, x_8)$ is

$$h_{128a}(x_0, \ldots, x_8) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8.$$

The output function is

$$z_i^{128a} = \sum_{j \in \mathcal{A}_{128a}} x_{i+j} + y_{i+93} + h_{128a}(x_{i+12}, y_{i+8}, y_{i+13}, y_{i+20}, x_{i+95}, y_{i+42}, y_{i+60}, y_{i+79}, y_{i+94}),$$

where $\mathcal{A}_{128a} = \{2, 15, 36, 45, 64, 73, 89\}$.

Algorithm 54. KSA^{-1} routine for Grain-128a.

Input: State $S_i = (x_0, \ldots, x_{127}, y_0, \ldots, y_{127})$
Output: The preceding state $S_{i-1} = (x_0, \ldots, x_{127}, y_0, \ldots, y_{127})$

$v = y_{127}$ and $w = x_{127}$
for $t = 127$ to 1 **do**
  $y_t = y_{t-1}$ and $x_t = x_{t-1}$
end
$z = \sum_{j \in A_{128a}} x_j + y_{93} + h_{128a}(x_{12}, y_8, y_{13}, y_{20}, x_{95}, y_{42}, y_{60}, y_{79}, y_{94})$
$y_0 = z + v + y_7 + y_{38} + y_{70} + y_{81} + y_{96}$
$x_0 = z + w + y_0 + x_{26} + x_{56} + x_{91} + x_{96} + x_3x_{67} + x_{11}x_{13} + x_{17}x_{18} + x_{27}x_{59} + x_{40}x_{48} + x_{61}x_{65} + x_{68}x_{84} + x_{88}x_{92}x_{93}x_{95} + x_{22}x_{24}x_{25} + x_{70}x_{78}x_{82}$

F Propagation of Single Bit Differentials

Parameters. In Theorem 2.4, let $q_2 = 96$ for Grain v1[1] and $q_2 = 160$ for Grain-128 and Grain-128a[2].

[1] as in Theorem 2.1
[2] as in Theorem 2.2, respectively Theorem 2.3

TABLE F.1: Propagation of a single bit differential in the case of Grain v1's LFSR.

Flipped Bit Position	Number of Identical Keystream Bits	Positions of Identical Keystream Bits
15	50	0-11, 13-17, 19-30, 33-35, 37, 38, 40-46, 48, 51, 53, 55, 58, 61-63, 71
31	59	0-5, 7-23, 25-27, 29-33, 35-41, 43-46, 49-51, 54, 56-59, 61, 62, 64, 67, 69, 74, 77, 79, 87
47	63	0, 2-21, 23, 24, 26-39, 41, 42, 45-49, 51-53, 55-57, 59, 60, 62, 65, 66, 70, 73-75, 77, 78, 80, 95
63	63	0-16, 18-27, 29-34, 36, 37, 39, 40, 42-45, 47-52, 54, 55, 58, 61-63, 65, 68, 69, 72, 73, 76, 81, 90, 91, 94
79	74	0-14, 16-32, 34-43, 45-50, 52, 53, 55, 56, 58-61, 63-68, 70, 71, 74, 77-79, 81, 84, 85, 88, 89, 92

TABLE F.2: Propagation of a single bit differential in the case of Grain v1's NFSR.

Flipped Bit Position	Number of Identical Keystream Bits	Positions of Identical Keystream Bits
15	23	0-4, 6-10, 12, 15, 16, 19, 20-22, 26, 27, 28, 29, 31, 33
31	32	1-19, 22-26, 28, 31, 32, 35, 36, 42, 43, 49
47	32	0-15, 17, 18, 20-25, 28, 29, 30, 32, 33, 35, 40, 41, 42
63	25	1-6, 8-16, 19, 21-23, 26, 29-31, 33, 39
79	41	0-15, 17-22, 24-32, 35, 37-39, 42, 45-47, 49, 55

TABLE F.3: Propagation of a single bit differential in the case of Grain 128's LFSR.

Flipped Bit Position	Number of Identical Keystream Bits	Positions of Identical Keystream Bits
31	92	0-10, 12-17, 19-22, 24-56, 58, 60-63, 65, 67-69, 71, 72, 74-79, 81-85, 87, 88, 90, 93, 94, 97, 100, 103, 109, 116, 119, 126, 129, 135, 141, 148
55	97	0-12, 14-34, 36-41, 43-46, 48, 49, 51, 53-65, 67-80, 86, 87, 89, 91-93, 95, 96, 100-102, 105-107, 109, 111, 112, 118, 121, 127, 133, 153, 159
79	101	1-18, 20-36, 38-41, 43, 45-57, 60-65, 67-70, 72, 73, 75, 78-88, 92-94, 96-99, 101, 103, 104, 110, 111, 113, 115, 119, 120, 125, 126, 130, 131, 133, 145, 151, 157
103	86	0-7, 9, 11-23, 25-39, 41, 44-54, 58-60, 62-65, 67, 69, 70, 73, 76-81, 84-86, 91, 92, 94, 96, 97, 99, 105, 109, 110-112, 116, 117, 123, 128, 143, 144

127	108	0-31, 33, 35-47, 49-63, 65, 68-78, 82-84, 86-89, 91, 93, 94, 97, 100-105, 108-110, 115, 116, 118, 120, 121, 123, 129, 133-136, 140, 141, 147, 152

TABLE F.4: Propagation of a single bit differential in the case of Grain 128's NFSR.

Flipped Bit Position	Number of Identical Keystream Bits	Positions of Identical Keystream Bits
31	52	0-15, 17, 18, 20-28, 30-36, 39-42, 45, 48-50, 54-56, 58, 62, 63, 65, 66, 71, 72
55	65	0-9, 11-18, 20-39, 41, 42, 44, 45, 47, 49-52, 55-60, 63-66, 69, 73, 74, 82, 87, 89, 95, 96
79	55	0-5, 7-14, 16-33, 35-42, 46, 48, 49, 52, 54, 55, 58, 60, 61, 63, 65, 68, 71, 74, 80
103	63	0-7, 9-13, 15-29, 31-38, 41-44, 47-50, 53-57, 59-61, 63-66, 70, 73, 79, 85, 87, 92, 98
127	87	0-31, 33-37, 39-53, 55-62, 65-68, 71-74, 77-81, 83-85, 87-90, 94, 97, 103, 109, 111, 116, 122

TABLE F.5: Propagation of a single bit differential in the case of Grain 128a's LFSR.

Flipped Bit Position	Number of Identical Keystream Bits	Positions of Identical Keystream Bits
31	83	0-10, 12-17, 19-22, 24-57, 60-63, 67-69, 71, 72, 74-79, 81-85, 87-89, 93, 94, 109, 111, 115
55	94	0-12, 14-34, 36-41, 43-46, 48-50, 53-65, 67-81, 86, 87, 91-93, 95, 96, 100-102, 105-108, 111, 112, 118, 133, 139
79	100	1-18, 20-36, 38-42, 45-57, 60-65, 67-70, 72-74, 78-89, 92-94, 96-100, 103, 104, 110, 111, 115, 119, 120, 125, 126, 130-132, 136, 157
103	93	0-8, 11-23, 25-40, 44-55, 58-60, 62-66, 69, 70, 72, 76-81, 84-87, 91, 92, 94, 96-98, 102, 109, 110-113, 116, 117, 123, 124, 128, 134, 143, 144, 149, 156
127	113	0-32, 35-47, 49-64, 68-79, 82-84, 86-90, 93, 94, 96, 100-105, 108-111, 115, 116, 118, 120-122, 126, 133-137, 140, 141, 147, 148, 152, 158

TABLE F.6: Propagation of a single bit differential in the case of Grain 128a's NFSR.

Flipped Bit Position	Number of Identical Keystream Bits	Positions of Identical Keystream Bits
31	44	0-15, 17, 18, 20-28, 30-36, 41, 49, 50, 54-56, 58, 63, 65, 66
55	55	0-9, 11-18, 20-39, 41, 42, 44, 45, 47, 49-52, 55-60, 65, 74

79	48	0-5, 7-14, 16-33, 35-39, 41, 46, 49, 52, 54, 55, 58, 60, 61, 63, 68
103	43	0-7, 9-13, 15-29, 31-38, 42, 53, 55-57, 59, 61
127	67	0-31, 33-37, 39-53, 55-62, 66, 77, 79-81, 83, 85

G Slide Attacks Examples

Within Tables G.1 to G.3, the padding is written in blue, while the red text denotes additional data necessary to mount the proposed attacks. Test vectors presented in this section are expressed as hexadecimal strings. For simplicity, we omit the `0x` prefix.

TABLE G.1: Examples of generic attacks (Algorithm 9).

Cipher	Key	LFSR Loading
Grain v1	a8af910f2755c064d713	1c60b94e09512adbffff
Grain 128	525c3676953ecec2bc5388f1474cdc61	b78d3637b6442501 5fa3ef63ffffffff
Grain 128a	a04f944e6ca1e1406537a0ef215689a3	aaaebb010224478f 48567997fffffffe

TABLE G.2: Examples of compact padding attacks (index $i = 1$).

	Cipher	Key	LFSR Loading	Keystream
Theorem 2.5 Condition 1 (Algorithm 13)	Grain v1	7e72b6f960cf9165 b891	1007bc3594e07f7f 7fa5	004e2da99a273923 83696e9e7120370a
		72b6f960cf9165b8 9145	07bc3594e07f7f7f a580	4e2da99a27392383 696e9e7120370a48
	Grain 128	00166499157d39c9 5a723b601eccfffb	4a9a37ef1e3dfc13 7fff7fff7fffeb05	000076755ac4cd53 028caa577964929e
		6499157d39c95a72 3b601eccfffb2fd1	37ef1e3dfc137fff 7fff7fffeb05d636	76755ac4cd53028c aa577964929ef1c0
	Grain 128a	b9e20a7619a8d622 5152cfa83eb73361	ef53aafa3c6c47ca 7fff7fff7ffff5cd	0000bac1203a11b5 54d69fd7f9f27b7f
		0a7619a8d6225152 cfa83eb7336175a5	aafa3c6c47ca7fff 7fff7ffff5cd98ba	bac1203a11b554d6 9fd7f9f27b7fd545
Theorem 2.5 Condition 3 (Algorithm 16)	Grain v1	455b5df993b367e3 7b60	07f7f7fe9b4a3044 efd1	0095e584ea234610 f7ec250a948a8267
		5b5df993b367e37b 604d	f7f7fe9b4a3044ef d139	95e584ea234610f7 2ec250a948a8267c
	Grain 128	9302f6b9d7136599 ac1caee130c596bb	8d7fff7fff7fff10 d59595e5568beb11	00007ca563c6831b 63868259f547cdff
		f6b9d7136599ac1c aee130c596bb0dc8	ff7fff7fff10d595 95e5568beb11628c	7ca563c6831b6386 8259f547cdff695b
	Grain 128a	0f478aa147938251 5e0a94d3357764f4	cd7fff7fff7fffed bb0e00ddcb18d1eb	000059362a172d87 48185e0850be7cb8
		8aa1479382515e0a 94d3357764f4b8bb	ff7fff7fffedbb0e 00ddcb18d1eb0416	59362a172d874818 5e0850be7cb824a0

Theorem 2.6 Condition 1	Grain v1	4febc079167f99bd b1db	bd4710804f9eff0f f0fa	000575b77251f394 6864d1bdc2510212
		bc079167f99bdb1d b338	710804f9eff0ff0f a272	575b77251f394686 4d1bdc251021229b
	Grain 128	5a0d4b3907f65ce5 f036b3671614244b	0bbd00872ecb0732 ffff00ffff00fffe	0000006b2014ecde e8d499646ba08a9f
		3907f65ce5f036b3 671614244be57112	872ecb0732ffff00 ffff00fffeaf68a2	6b2014ecdee8d499 646ba08a9fd93085
	Grain 128a	6472c21093cd2225 4118e1a69230e0ac	2c9c47771ed4f648 ffff00ffff00ffde	0000009e196e7e86 6193867ea31b1df0
		1093cd22254118e1 a69230e0ac668222	771ed4f648ffff00 ffff00ffdeb9f179	9e196e7e86619386 7ea31b1df09f306a
Theorem 2.6 Condition 2	Grain v1	701aa599737c957a 0b5e	07ff0ff0fdedd9bd 4d1b	000f9b9045f817c5 51a7c56c18e4ec02
		aa599737c957a0b5 eb77	f0ff0fdedd9bd4d1 b1bf	f9b9045f817c51a7 c56c18e4ec025d85
	Grain 128	30bfe11f3b7080be 47396a37f889b57c	aafdffff00ffff00 ff38ff5b14da5371	0000008a735f3adf 71728258dcaf47fd
		1f3b7080be47396a 37f889b57cac5367	ff00ffff00ff38ff 5b14da53715a4291	8a735f3adf717282 58dcaf47fd6edad1
	Grain 128a	c4b8607e854abc5f 7a74eba33d563ad1	950bffff00ffff00 ff7182c277b77e8f	000000681060aa4b f10c0181bd7e4d95
		7e854abc5f7a74eb a33d563ad125aaff	ff00ffff00ff7182 c277b77e8f5db61f	681060aa4bf10c01 81bd7e4d957b5f2e

TABLE G.3: Examples of fragmented padding attacks (index $i = 1$).

	Cipher	Key	LFSR Loading	Keystream
Theorem 2.7 Condition 1 (Algorithm 20)	Grain v1	cc0d50254f72d88d 3c71	3a86d17377777777 7b2c	04c79ebb4db7bc67 5644b3d0bf2a59a4
		c0d50254f72d88d3 c714	a86d173777777777 b2cf	4c79ebb4db7bc675 644b3d0bf2a59a47
	Grain 128	c506d0ca5bff72e1 6ea07fd8f98d7ba3	63ba70cf067f7f7f 7f7f7f7f7f879f9b	004e2c99a48677b4 c217f9e14e620d48
		06d0ca5bff72e16e a07fd8f98d7ba368	ba70cf067f7f7f7f 7f7f7f7f879f9be1	4e2c99a48677b4c2 17f9e14e620d4884
	Grain 128a	0948bd1a0a5d275c 54744db3dc27cec8	895ba804147f7f7f 7f7f7f7f7f2f9892	003a5f1e38d9c446 70b0dc017377e698
		48bd1a0a5d275c54 744db3dc27cec82b	5ba804147f7f7f7f 7f7f7f7f2f9892f1	3a5f1e38d9c44670 b0dc017377e698d7
Theorem 2.7 Condition 3 (Algorithm 21)	Grain v1	77a73157cabfa603 49dc	77777777318f59ac 6aff	0c61bfa06e1c2201 1dcefe673765acb7
		7a73157cabfa6034 9dc3	7777777318f59ac6 affd	c61bfa06e1c22011 dcefe673765acb7f
	Grain 128	9aca3bd2cf312080 769338bec86f9da6	7f7f7f7f7f7f7f7f b6f7e83b3793f746	004624d2271d3420 104b2fd1058675fd
		ca3bd2cf31208076 9338bec86f9da63f	7f7f7f7f7f7f7fb6 f7e83b3793f746ff	4624d2271d342010 4b2fd1058675fd45
	Grain 128a	0e9eb1a896077e93 5b21de8700f3ef44	7f7f7f7f7f7f7f7f 29b03ff3e82cda8b	007f06d63e3545f6 b7c4b50d255b6663
		9eb1a896077e935b 21de8700f3ef4462	7f7f7f7f7f7f7f29 b03ff3e82cda8bfc	7f06d63e3545f6b7 c4b50d255b6663ea

Theorem 2.8 Condition 1 (Algorithm 24)	Grain 128	d3ea84c99a8b1354	ed52bf1b25ff0ff0	0001590b803ff3c9
		71d8c320b870e109	fff0ff0f4ed8f575	972d96481a6e8ad4
		a84c99a8b135471d	2bf1b25ff0ff0fff	1590b803ff3c9972
		8c320b870e109120	0ff0f4ed8f575dac	d96481a6e8ad48ee
	Grain 128a	9ee02802ccf920e6	ab24f8ab82ff0ff0	00082e1cbbb25fa3
		868a8aa46113a406	fff0ff0fd32dc4e9	25518665a17f2efc
		02802ccf920e6868	4f8ab82ff0ff0fff	82e1cbbb25fa3255
		a8aa46113a40681d	0ff0fd32dc4e9473	18665a17f2efc2eb
Theorem 2.8 Condition 2	Grain 128	8d89931ae1e13215	f18ccfbf3cff0ff0	000e612c620ae176
		77bba20640c193a1	ff0ff0fde5af2b58	5ded57a835b713ac
		9931ae1e1321577b	ccfbf3cff0ff0ff0	e612c620ae1765de
		ba20640c193a13b8	ff0fde5af2b58811	d57a835b713ace4a
	Grain 128a	626262808f0ca24c	c4ca6f9535ff0ff0	0003f5a6d1b7f615
		cc517bb93fb5c3cb	ff0ff0fdfe92e568	dfb32e34cea7cc4a
		262808f0ca24ccc5	a6f9535ff0ff0ff0	3f5a6d1b7f615dfb
		17bb93fb5c3cb22f	ff0fdfe92e568a4f	32e34cea7cc4a106
Theorem 2.8 Condition 3	Grain 128	416ddd14b4c096cb	80ff0ff0fff0ff0f	00076a8e9def620d
		0181ae8830ada69d	d7ef096c7a8700a3	fe704b264988da02
		ddd14b4c096cb018	f0ff0fff0ff0fd7e	76a8e9def620dfe7
		1ae8830ada69d3b6	f096c7a8700a318f	04b264988da02cc0
	Grain 128a	724d58601b44396d	84ff0ff0fff0ff0f	0008ab9f20d8a418
		60e83723a65bfa7b	6c25a1d79af2a85c	932150d3ba97400e
		d58601b44396d60e	f0ff0fff0ff0f6c2	8ab9f20d8a418932
		83723a65bfa7b973	5a1d79af2a85c626	150d3ba97400ebd5
Theorem 2.8 Condition 4	Grain 128	97516dced374a089	3aff0ff0ff0ff0f1	000a8e820bedfb8c
		88ce86acaa2ff1a4	12b72427d44b92f1	d9d651d8221f3b34
		16dced374a08988c	f0ff0ff0ff0f112b	a8e820bedfb8cd9d
		e86acaa2ff1a4399	72427d44b92f1bba	651d8221f3b34846
	Grain 128a	a29ae6fb8b23f747	4bff0ff0ff0ff0fc	000cd469723847db
		f3723e59df0d3a8e	92ace3a64691e733	72f6f856e51f9d96
		ae6fb8b23f747f37	f0ff0ff0ff0fc92a	cd469723847db72f
		23e59df0d3a8eabb	ce3a64691e733a54	6f856e51f9d96b38
Theorem 2.9 Condition 1 (Algorithm 25)	Grain 128	930cb0086c93293e	f767352c26395e8a	0000000a44dcae9a
		9722a710e28a1375	ffffb0ffff80fffb	68c7b66389e440eb
		086c93293e9722a7	2c26395e8affffb0	0a44dcae9a68c7b6
		10e28a1375ec5696	ffff80fffbb6fcf2	6389e440ebbdf198
	Grain 128a	270f72277e7540cf	c7df3ee9c792f5d5	000000fd8bbdb3d3
		9a58fa4426e28aae	ffffd0ffff00fff1	a8c885704f43a022
		277e7540cf9a58fa	e9c792f5d5ffffd0	fd8bbdb3d3a8c885
		4426e28aaebc06e1	ffff00fff13204c5	704f43a022557a89
Theorem 2.9 Condition 2	Grain 128	895bea372ffe4e76	a8147ffff80fffffe	0000004b5394f9ba
		e84113dd18afa6b9	0fff2cd80e83e74	f0f6a6ff3d921542
		372ffe4e76e84113	fff80fffff0fff2c	4b5394f9baf0f6a6
		dd18afa6b9fb5cef	d80e83e74e3d134e	ff3d9215422cbdbb
	Grain 128a	70a2fecddbc94115	9e132ffff50ffffd	0000002839a6bec7
		017b571df0854817	0fff5cf89b04484d	7a007d3d12b4d597
		cddbc94115017b57	fff50ffffd0fff5c	2839a6bec77a007d
		1df08548178142d5	f89b04484d01fb4b	3d12b4d597c9041b

H Optimized Decryption Algorithms

In [150], the authors provide the reader with different versions of the decryption algorithm corresponding to the Joye-Libert cryptosystem. We present slightly modified versions of [150, Algorithm 3 and 4] in Algorithms 55 and 56. The authors also propose two other optimizations [150, Algorithm 5 and 6], but their complexity is similar with Algorithm 3 and 4's complexity. Note that these optimizations contain a typo: in line 5, Algorithm 5 and line 6, Algorithm 6 we should have $A^{k-j} \neq C[k-j] \bmod p$ instead of $A \neq C[k-j] \bmod p$.

For these algorithms to work we need to enhance the *KeyGen* algorithm of our proposed cryptosystem. More precisely, we generate the $\gamma + 1$ prime numbers p_i with the supplementary restriction $p_i \not\equiv 1 \bmod 2^{k+1}$. For $0 \leqslant i < \gamma$, let $p'_i = (p_i - 1)/2^k$. We precompute $D_i = y_i^{-p'_i}$ for Algorithm 55 and $D_i[j] = D_i^{2^{j-1}} \bmod p_i$, $1 \leqslant j \leqslant k-1$, for Algorithm 56 and augment the private key with these values. Remark that Algorithm 56 requires more memory than Algorithm 55.

Algorithm 55. Fast decryption algorithm Version 1

Input: The secret values (p_i, p'_i, D_i), the value y_i and the ciphertext c
Output: The message block m_i

1 $m_i \leftarrow 0$, $B \leftarrow 1$
2 $C \leftarrow c^{p'_i} \bmod p_i$
3 **foreach** $j \in [1, k-1]$ **do**
4 $\quad z \leftarrow C^{2^{k-j}} \bmod p_i$
5 $\quad$ **if** $z \neq 1$ **then**
6 $\quad\quad m_i \leftarrow m_i + B$
7 $\quad\quad C \leftarrow C \cdot D_i \bmod p_i$
8 $\quad$ **end**
9 $\quad B \leftarrow 2B$, $D \leftarrow D^2 \bmod p_i$
10 **end**
11 **if** $C \neq 1$ **then**
12 $\quad m_i \leftarrow m_i + B$
13 **end**
14 **return** m_i

Correctness. Let $m_i = \sum_{w=0}^{k-1} b_w 2^w$ be the binary expansion of block m_i. We define $\alpha_i[s] = 2^{k-s} p'_i$. Note that

$$\begin{aligned} c^{\alpha_i[s]} &\equiv (x^{2^k} \cdot \prod_{v=1}^{\gamma} y_v^{m_v})^{\alpha_i[s]} \\ &\equiv y_i^{\alpha_i[s] \sum_{w=0}^{s-1} b_w 2^w} \\ &\equiv y_i^{b_{s-1} 2^{k-1} p'_i} y^{\alpha_i[s] \sum_{w=0}^{s-2} b_w 2^w} \\ &\equiv (-1)^{b_{s-1}} y^{\alpha_i[s] \sum_{w=0}^{s-2} b_w 2^w} \bmod p_i \end{aligned}$$

since

1. $(x^{2^k})^{\alpha_i[s]} = x^{2^{k-s}(p_i-1)} = 1$
2. $\left(\frac{y_j}{p_i}\right)_{2^k} = 1$, where $j \neq i$
3. $\sum_{w=0}^{k-1} b_w 2^w = \left(\sum_{w=0}^{s-1} b_w 2^w\right) + 2^s \cdot \left(\sum_{w=s}^{k-1} b_w 2^{w-s}\right)$
4. $\left(\frac{y_i}{p_i}\right) = -1$

As a result, the message block m_i can be recovered bit by bit using the values p_i, p'_i and the vector D_i.

Algorithm 56. Fast decryption algorithm Version 2

Input: The secret values $(p_i, p'_i, D_i[1], \ldots D_i[k-1])$, the value y_i and the ciphertext c

Output: The message block m_i

```
1  m_i ← 0, B ← 1
2  C ← c^{p'_i} mod p_i
3  foreach j ∈ [1, k − 1] do
4  |   z ← C^{2^{k−j}} mod p_i
5  |   if z ≠ 1 then
6  |   |   m_i ← m_i + B
7  |   |   C ← C · D_i[j] mod p_i
8  |   end
9  |   B ← 2B
10 end
11 if C ≠ 1 then
12 |   m_i ← m_i + B
13 end
14 return m_i
```

H.0.1 Implementation Details

The complexities of Algorithms 55 and 56 are $\mathcal{O}(\gamma(\lambda+\frac{k^2}{2}+\frac{3k}{2})M(\lambda)\lceil\frac{\eta}{\gamma k}\rceil)$ and $\mathcal{O}(\gamma(\lambda+\frac{k^2}{2}+\frac{k}{2})M(\lambda)\lceil\frac{\eta}{\gamma k}\rceil)$.

We further provide the reader with benchmarks for the optimized versions of our PKE scheme.

TABLE H.1: Average running times for Algorithm 55.

Algorithm	$\gamma=1$				
	$k=1$	$k=2$	$k=4$	$k=8$	$k=16$
KeyGen	0.458942	0.456482	0.441694	0.453773	0.480818
Encrypt	0.006976	0.004521	0.003295	0.002397	0.001896
Decrypt	0.343437	0.171095	0.086448	0.043583	0.023451

Algorithm	$\gamma=2$			
	$k=1$	$k=2$	$k=4$	$k=8$
KeyGen	0.680506	0.668120	0.651916	0.772909
Encrypt	0.006693	0.005263	0.004054	0.003350
Decrypt	0.340996	0.170928	0.085498	0.043677

Algorithm	$\gamma=4$			$\gamma=8$	
	$k=1$	$k=2$	$k=4$	$k=1$	$k=2$
KeyGen	1.135950	1.174290	1.200390	2.041340	2.023290
Encrypt	0.008339	0.008142	0.007131	0.012590	0.015871
Decrypt	0.339094	0.170217	0.085524	0.336030	0.168950

TABLE H.2: Average running times for Algorithm 56.

Algorithm	$\gamma=1$				
	$k=1$	$k=2$	$k=4$	$k=8$	$k=16$
KeyGen	0.512341	0.470040	0.435809	0.522212	0.482328
Encrypt	0.006914	0.004496	0.003091	0.002375	0.001878
Decrypt	0.338572	0.170079	0.084772	0.042978	0.023005

Algorithm	$\gamma=2$			
	$k=1$	$k=2$	$k=4$	$k=8$
KeyGen	0.664869	0.740169	0.676827	0.675473
Encrypt	0.006617	0.005105	0.004002	0.003323
Decrypt	0.337033	0.168480	0.084523	0.043039

Algorithm	$\gamma=4$			$\gamma=8$	
	$k=1$	$k=2$	$k=4$	$k=1$	$k=2$
KeyGen	1.077020	1.062580	1.098260	1.957820	2.129930
Encrypt	0.008273	0.008070	0.007033	0.012446	0.015671
Decrypt	0.335889	0.168382	0.084759	0.331968	0.165645

I Two-Party Malicious Signing

In [169], the author introduces a two-party protocol for signing with ECDSA. Based on this idea, we sketch a protocol that extends $S - GEGS$. Using this extension, two malicious parties M_1 and M_2 can impersonate V without explicitly computing sk_V.

As the Paillier cryptosystem [202] is later used in the protocol, we shortly describe its homomorphic properties. We denote the public and private key pair of M_1 by (pk_p, sk_p), Paillier encryption by $Enc(pk_p, \cdot)$ and Paillier decryption by $Dec(sk_p, \cdot)$. Let n be a large composite number in the Paillier scheme sense, $c_1 \leftarrow Enc(pk_p, m_1)$ and $c_2 \leftarrow Enc(pk_p, m_2)$, where messages $m_1, m_2 \in \mathbb{Z}_n$. The upcoming properties hold

- the *addition* of m_1 and m_2 modulo n (represented by $c_1 \oplus c_2$ in the current section):

  $Dec(sk_p, c_1 c_2 \bmod n^2) = m_1 + m_2 \bmod n$;

- the *multiplication* of m_1 by a constant t modulo n (represented by $t \odot c_1$ in the current section):

  $Dec(sk_p, c_1^t \bmod n^2) = t m_1 \bmod n$.

Before the malicious signing protocol can start, the two parties must agree on the protocol's parameters. In Figure I.1 we present the parameters agreement protocol. The protocol uses an ideal commitment scheme and an ideal non-interactive zero-knowledge proof. For concrete instantiation of the two, we refer the reader to [169].

M_1		M_2
α_1		α_2
$Q \leftarrow g^{\alpha_1}$		
Compute DL proof π_1.		
Generate a Paillier key-pair (pk_p, sk_p).		
Generate a proof π_2 that pk_p is generated correctly.		
$c_{key} \leftarrow Enc(pk_p, \alpha_1)$		
Generate a proof π_3 of a DL in c_{key}.		
	$\xrightarrow{Q, \pi_1, \pi_2, \pi_3}$	
		Verify proofs π_1, π_2 and π_3.
		If `false` abort, else store c_{key}.

FIGURE I.1: Parameters generation.

Let m_3 be the message that M_1 wants to sign. By combining equation (5.3) with the $\mathcal{GEGS}$ signing operation we obtain the following equation for malicious signing m_3

$$s_3 \leftarrow k_3^{-1}\xi_1^{-1}[\xi_1 \cdot h(m_3) - \xi_2 \cdot h(r_3)] \bmod q, \tag{1}$$

where

$$\xi_1 \leftarrow \left(\sum_{i=1}^{2} \alpha_i \cdot h(r_i) \cdot \Delta_i - s_0^{-1} \cdot h(r_0)\right) \text{ and } \xi_2 \leftarrow \left(\sum_{i=1}^{2} \alpha_i \cdot h(m_i) \cdot \Delta_i - s_0^{-1} \cdot h(m_0)\right).$$

In Figure I.2 we describe in detail the two-party protocol for signing m_3. To simplify the protocol, instead of $h(m)$ and $h(r)$ we simply write m and r. As in Figure I.1, we use a commitment scheme and a zero knowledge protocol.

We can observe that, by using c_{key} and the homomorphic properties of the Paillier cryptosystem, M_2 can encrypt $u_1 \leftarrow k_{33}\xi_1$ and $u_2 \leftarrow k_{34}^{-1}[\xi_1 \cdot h(m_3) - \xi_2 \cdot h(r_3)]$. After M_1 receives the ciphertexts, it decrypts them and computes $k_{31}u_1$ and $k_{32}u_3$. Now, M_1 can compute m_3's signature (r_3, s_3), where $k_3 \leftarrow k_{31}k_{32}k_{33}k_{34}$.

J An ℓ out of ℓ Threshold Attack on the Generalized ElGamal Signature

In this section, we introduce an ℓ out of ℓ threshold version of the Young-Yung SETUP mechanism. In this particular case, the proposed scheme is more efficient than the one proposed in Section 5.2.2.

J.1 Description

To implement their attack, the ℓ malicious parties work in almost the same environment as in Section 5.2.2. Thus, we only mention the differences between the environments. We denote by $PK_M = \{pk_i\}_{1 \leqslant i \leqslant \ell}$ and present these changes below.

Signing Sessions: The possible signing sessions performed by D are described below. Let $i \geqslant 1$.

Session$_0(m_0, sk_V)$: To sign message $m_0 \in \mathbb{G}$, D does the following

$$k_0 \xleftarrow{\$} \mathbb{Z}_q^*, r_0 \leftarrow g^{k_0}, s_0 \leftarrow k_0^{-1}[h(m_0) - a \cdot h(r_0)].$$

M_1		M_2
m_3, α_1		m_3, α_2, c_{key}
$k_{31}, k_{32} \xleftarrow{\$} \mathbb{Z}_q^*$		
$R_1 \leftarrow g^{k_{31}k_{32}}$		
Compute DL proof π_1.		
Compute commit τ to R_1, π_1.		
	$\xrightarrow{\tau}$	
		$k_{33}, k_{34} \xleftarrow{\$} \mathbb{Z}_q^*$
		$R_2 \leftarrow g^{k_{33}k_{34}}$
		Compute DL proof π_2.
	$\xleftarrow{R_2, \pi_2}$	
Verify proof π_2.		
If `false` abort.		
	$\xrightarrow{R_1, \pi_1}$	
		Verify commit τ and proof π_1.
		If `false` abort.
		$r_3 \leftarrow R_1^{k_{33}k_{34}}$
		$\rho_1, \rho_2 \xleftarrow{\$} \mathbb{Z}_{q^2}^*$
		$v_1 \leftarrow \alpha_2 r_2 \Delta_2 - s_0^{-1} r_0 \bmod q$
		$c_1 \leftarrow Enc(pk_p, (k_{33} v_1 \bmod q) + \rho_1 q)$
		$v_2 \leftarrow k_{33} r_1 \Delta_1 \bmod q$
		$c_2 \leftarrow v_2 \odot c_{key}$ and $c_3 \leftarrow c_1 \oplus c_2$
		$v_3 \leftarrow r_3(\alpha_2 m_2 \Delta_2 - s_0^{-1} m_0) \bmod q$
		$v_4 \leftarrow k_{34}^{-1}(m_3 v_1 - v_3) \bmod q$
		$c_4 \leftarrow Enc(pk_p, v_4 + \rho_2 q)$
		$v_5 \leftarrow k_{34}^{-1}(m_3 r_1 \Delta_1 - r_3 m_1 \Delta_1) \bmod q$
		$c_5 \leftarrow v_5 \odot c_{key}$ and $c_6 \leftarrow c_4 \oplus c_5$
	$\xleftarrow{c_3, c_6}$	
$r_3 \leftarrow R_2^{k_{31}k_{32}}$		
$u_1 \leftarrow Dec(sk_p, c_3)$		
$u_2 \leftarrow k_{31} u_1 \bmod q$		
$u_3 \leftarrow Dec(sk_p, c_6)$		
$s_3 \leftarrow k_{32}^{-1} u_1^{-1} u_3 \bmod q$		
Verify signature (r_3, s_3).		
If `false` abort, else output (r_3, s_3).		

FIGURE I.2: Two-party malicious signing.

The value k_0 is stored in D's volatile memory until the end of $Session_1$. Output the signature (r_0, s_0).

$Session_i(m_i, sk_V, PK_M)$: To sign message $m_i \in \mathbb{G}$, D does the following

$$z_i \leftarrow (y_1 \cdot \ldots \cdot y_\ell)^{k_{i-1}}, k_i \leftarrow H(z_i), r_i \leftarrow g^{k_i}, s_i \leftarrow k_i^{-1}[h(m_i) - a \cdot h(r_i)].$$

The value k_i is stored in D's volatile memory until the end of $Session_{i+1}$. Output the signature (r_i, s_i).

Recovering$(m_i, r_{i-1}, r_i, s_i, SK_M)$: Compute $\alpha_i \leftarrow r_{i-1}^{x_i}$ and $k_i \leftarrow H(\alpha_1 \cdot \ldots \cdot \alpha_\ell)$. Recover a by computing

$$a \leftarrow h(r_i)^{-1}[h(m_i) - k_i \cdot s_i)].$$

Remark J.1. Let Q be an honest generator for the values r used by the Generalized ElGamal signature scheme and let σ_i denote the i-th internal state and $\rho_i = g^{\sigma_i}$ the i-th output of Q. The mechanism described above can be seen as a malicious PRNG $\tilde{Q}$ based on the honest PRNG Q. We define the internal states and outputs of $\tilde{Q}$ by

- $\tilde{\sigma}_0 = \sigma_0$, $\tilde{\rho}_0 = \rho_0$;
- $\tilde{\sigma}_i = H(z_i)$, $\tilde{\rho}_i = g^{\tilde{\sigma}_i}$, where $z_i \leftarrow (y_1 \cdot \ldots \cdot y_\ell)^{\tilde{\sigma}_{i-1}}$, $i \geqslant 1$.

Unlike $\tilde{P}$ from Remark 5.13, $\tilde{Q}$ can be harmful by itself[3]. A coalition of ℓ malicious parties that know an output $\tilde{\rho}_{i-1}$ can compute the next internal state $\tilde{\sigma}_i$. $\tilde{Q}$ is a threshold variant of the generator described in Remark 5.5.

J.2 Security Analysis

In this subsection we show that the scheme described above, denoted $F-GEGS$, cannot be distinguished from $GEGS$ if adversary A corrupted at most $\ell - 1$ malicious parties M_i.

Theorem J.1. If the number of signatures is polynomial and HDH is hard in $\mathbb{G}$ then $GEGS$ and $F - GEGS$ are IND-SETUP in the standard model as long as at most $\ell - 1$ malicious parties are corrupted by A. Formally, let A be an efficient PPT IND-SETUP adversary. There exists an efficient algorithm B such that

$$ADV^{\text{IND-SETUP}}_{MEG,GEGS,F-GEGS}(A) \leqslant 4\Gamma ADV^{\text{HDH}}_{\mathbb{G},g,H}(B),$$

where Γ is the number of infected signatures.

Proof. Let A be an IND-SETUP adversary that is trying to distinguish between $GEGS$ and $F - GEGS$. A has access to "random coins" sampled uniformly from a set R. Without loss of generality, we further assume that A has corrupted the first $\ell - 1$ malicious participants.

Algorithm 57 describes the IND-SETUP game. The first and second rows set up the public keys. Then the $GEGS$ and $F - GEGS$ oracles are described. The challenger then flips a bit b and reveals oracle C_b. A then computes its guess b' for b. A wins if $b = b'$.

[3] *i.e* not only when used with ElGamal based signatures

Algorithm 57. The IND-SETUP game.

Function `init()`**:**
  Choose the secret keys $a, x_1, \ldots, x_\ell \xleftarrow{\$} \mathbb{Z}_q^*$
  Compute the public keys $y \leftarrow g^a$, $y_1 \leftarrow g^{x_1}, \ldots, y_\ell \leftarrow g^{x_\ell}$
  Set $\mathcal{L}_1 \leftarrow \left(\cup_{i=1}^{\ell-1}\{x_i\}\right) \cup \left(\cup_{i=1}^{\ell}\{y_i\}\right)$ and $i \leftarrow 1$
Function $C_0(a,\ m)$**:**
  Choose $k \xleftarrow{\$} \mathbb{Z}_q^*$
  Compute $r \leftarrow g^k$ and $s \leftarrow k^{-1}[h(m) - a \cdot h(r)]$
  return (r, s)
Function $C_1(a,\ m)$**:**
  if $i = 0$ **then**
    Choose $k_0 \xleftarrow{\$} \mathbb{Z}_q^*$
  else
    Compute $z_i \leftarrow (y_1 \cdot \ldots \cdot y_\ell)^{k_{i-1}}$ and $k_i \leftarrow H(z_i)$
  end
  Compute $r \leftarrow g^{k_i}$, $s \leftarrow k_i^{-1}[h(m) - a \cdot h(r)]$ and $i \leftarrow i + 1$
  return (r, s)
`init()`
Choose $b \xleftarrow{\$} \{0, 1\}$ and $\rho \xleftarrow{\$} R$
return $A^{C_b(a,\cdot)}(\rho, y, \mathcal{L}_1)$

Algorithm 58. The init() and C_1 functions for the new IND-SETUP game.

Function `init()`**:**
  Choose the secret keys $a, x_1, \ldots, x_\ell \xleftarrow{\$} \mathbb{Z}_q^*$
  Compute the public keys $y \leftarrow g^a$, $y_1 \leftarrow g^{x_1}, \ldots, y_\ell \leftarrow g^{x_\ell}$
  Select $x_t \xleftarrow{\$} \mathbb{Z}_q^*$ and let $y_t \leftarrow g^{x_t}$
  Set $\mathcal{L}_1 \leftarrow \left(\cup_{i=1}^{\ell-1}\{x_i\}\right) \cup \left(\cup_{i=1}^{\ell}\{y_i\}\right)$ and $i \leftarrow 1$
Function $C_1(a,\ m)$**:**
  if $i = 0$ **then**
    Choose $k_0 \xleftarrow{\$} \mathbb{Z}_q^*$
  else
    Compute $z_i \leftarrow y_t^{k_{i-1}}$ and $k_i \leftarrow H(z_i)$
  end
  Compute $r \leftarrow g^{k_i}$, $s \leftarrow k_i^{-1}[h(m) - a \cdot h(r)]$ and $i \leftarrow i + 1$
  return (r, s)

We proceed by changing the initial IND-SETUP game (described in Algorithm 57) into a new IND-SETUP game (described in Algorithm 58). In addition to the original set up, in the new version, we choose an extra secret internal state y_t. Another change is the way we compute the k_i values from oracle C_1. In the original game we multiply the element $y_1 \cdot \ldots \cdot y_{\ell-1}$ from $\mathbb{G}$ with an uniformly random element y_ℓ of the same set and we obtain an uniformly random element. In the new game we directly use a random value y_t for

computing the k_i values, thus the change is statically indistinguishable. Since these are the only changes, an adversary will not notice any difference between the IND-SETUP games.

Since MEG is IND\$ an adversary cannot distinguish between C_0 and C_1. Note that the number of k values that A has to distinguish is n. Thus, we obtain the security margin. □

Remark J.2. Similarly to Theorem J.1, we obtain that if Q is a secure PRNG, then $\tilde{Q}$ is a secure PRNG in the standard model.

Remark J.3. As in the case of Dual-EC, it is easy to see that if in the $F - GEGS$ scheme, we replace y_i with $y_i' \xleftarrow{\$} \mathbb{G}$, $1 \leqslant i \leqslant n$, the SETUP mechanism becomes benign. The security margin of the SETUP-free system remains the same as the one stated in Theorem J.1.

K Additional Algorithms

In [253] the algorithm used to invert g is not presented in full. Based on the descriptions found in [253, 1] we present the full algorithm in Algorithm 59. Note that the algorithm works for any generic polynomial g, not only for the one used in the Flash Player PRNG. Note that $\&S$ means that we pass S by reference.

The only algorithm we found for reversing the Flash Player PRNG is described in [1]. We improve their attack in Algorithm 60. To reverse the bit manipulation function f and the polynomial g we use the abstract functions *Reverse_bit_manipulation* and *Reverse_polynomial*, respectively. Remark that Algorithm 60 works for any generic polynomial g and any generic function $h(x) = p \cdot x \bmod 2^n$ with p odd. In the Flash Player case we have $p^{-1} \equiv 3811027319 \bmod 2^{32}$.

L Recreational Cryptographic Problems

The interest of the cryptographic community regarding various recreational cryptography problems has grown in time. We further recall a series of physical cryptographic solutions which appeared in the literature. Note that our list of recreational cryptographic problems is, by no means, extensive.

Algorithm 59. Backtracking algorithm for reversing g.

Input: The value to reverse v
Output: The set of possible solutions S

Function $Verify\_ith\_bit(v, i, sol)$**:**
  $bit_1 \leftarrow g(sol)$ & $(1 \ll i)$;
  $bit_2 \leftarrow v$ & $(1 \ll i)$;
  return $bit_1 == bit_2$;
Function $Add\_ith\_bit(v, i, sol, \&S, b)$**:**
  $sol \leftarrow sol \mid (b \ll i)$;
  if $Verify\_ith\_bit(v, i, sol) ==$ `true` **then**
    $i \leftarrow i + 1$;
    $Reverse\_bit(v, i, sol, S)$;
  end
Function $Reverse\_bit(v, i, sol, \&S)$**:**
  if $i == n$ **then**
    $S \leftarrow S \cup sol$;
    return;
  end
  $add\_ith\_bit(v, i, sol, S, 0)$;
  $add\_ith\_bit(v, i, sol, S, 1)$;
Function $Reverse\_polynomial(v)$**:**
  $S \leftarrow \varnothing$; //the set of possible solutions
  $i \leftarrow 0$; //the target bit
  $sol \leftarrow 0$; //the current solution
  $reverse\_bit(v, i, sol, S)$;
  return S;

Algorithm 60. The algorithm for reversing the PRNG.

Input: The value to reverse v
Output: The set of possible solutions S

$v' \leftarrow v \mid (1 \ll (n-1))$;
$S_{bit} \leftarrow Reverse\_bit\_manipulation(v) \cup Reverse\_bit\_manipulation(v')$;
$S_{pol}, S_{hash}, S \leftarrow \varnothing$;
for $s_{bit} \in S_{bit}$ **do**
  $S_{pol} \leftarrow S_{pol} \cup Reverse\_polynomial(s_{bit})$;
end
for $s_{pol} \in S_{pol}$ **do**
  $S_{hash} \leftarrow S_{hash} \cup Reverse\_bit\_manipulation(s_{pol})$;
end
for $s_{hash} \in S_{hash}$ **do**
  $s \leftarrow s_{hash} \cdot p^{-1} \bmod 2^n$;
  $S \leftarrow S \cup s$;
end
return S;

"Finding Waldo" Solution. The authors of [191] provide an insight on how to convince people about knowing Waldo's location without revealing it. We initially assume

that Alice and Bob have a large piece of cardboard[4]. As a first step, Alice cuts a Waldo shaped hole in the middle of the cardboard. To prove that she knows where Waldo is, Alice puts the shape precisely on top of Waldo while Bob is not looking and then calls Bob to check. Given the previous steps of the protocol, Bob learns nothing about the location of Waldo. Next, Alice must prove that she has the correct Waldo picture. Therefore, she must pull the book beneath the cardboard in front of Bob's eyes without revealing information about the place from which she is pulling the book[5].

"Ali Baba Cave" Solution. A well known story for explaining the intuition behind zero knowledge protocols is presented in [210]. The story is about a magical cave shaped like a ring with an entrance on one side as well as a magical door blocking the opposite side. We assume that Alice discovers the secret magical word that opens the door and wants to prove to Bob that she knows the secret without revealing it. Thus, they agree to label the left and right paths from the entrance `head` and `tail`. The protocol proceeds as follows. Bob waits outside the cave as Alice goes in. Then, Alice flips a coin to determine the path she follows. Note that Bob is not allowed to see which path she takes. Bob enters the cave, flips a coin and shouts the outcome. If Alice knows the magical word she opens the door, if necessary, and returns along the path chosen by Bob. If she lied about knowing it, then she has a 50% chance of returning through the correct path (*i.e.* by guessing Bob's outcome). If they repeat this protocol multiple times, the chance of Alice tricking Bob decreases. Thus, if Alice always exits through the right path, Bob can conclude that Alice really knows the secret word.

"Locked Boxes" Solution. A classical method for explaining symmetric encryption is through the use of "impenetrable" locked boxes (see [44, 62]). More precisely, Alice and Bob both have a copy of the key that opens a chest. To exchange messages, Alice simply puts her letter in the box, locks it and sends it to Bob. Since Bob has an identical copy of the key, he opens the chest and reads the letter. Another protocol that can be explained using locked boxes is Shamir's three-pass protocol [180]. First, Alice puts her message in a box, locks it with her private padlock and sends it to Bob. Then, Bob places his private padlock on the box and sends it back to Alice. Once she receives the box, she removes her padlock and sends the box to Bob. Finally, Bob removes his padlock and reads Alice's message. In order to popularize cryptography to non-specialized audiences, the authors of [44] used a toolbox or a loose chain to implement the previous physical example of Shamir's protocol. The authors point out it is easy to prove[6] to audiences that a persistent code-breaker could always dismantle a padlock, or X-ray it, and hence

[4] at least twice as large as the picture in each dimension
[5] At least the hole should be covered while the book is pulled out.
[6] *e.g.* by showing a sawn up padlock

crack the code (*i.e.* knowing the inside of the lock is isomorphic to knowing the key). Thus, we have to employ other techniques than the secrecy of the encryption method.

By relaxing the security requirements from an "impenetrable" box to a tamper-evident box (*i.e.* the receiver can detect if someone managed to open the box) the authors of [186, 187] devise a series of secure protocols.

Ciphers Based on a Deck of Cards. Schneier designed the "Solitaire" cipher [218] for the book "Cryptonomicon" [232][7]. Solitaire was intended to be the first truly secure "pen and paper" cipher. It requires only a pack of cards both for encryption and decryption. A similar example is the "Mirdek" cipher [84].

"PEZ Dispenser" Solution. In [37] the authors present a solution for voting using a PEZ dispenser. Consider a group of kids wishing to vote between two candidates without revealing anything except the final outcome. Assume that they have a PEZ dispenser, which may be previously loaded with some publicly known sequence of red and yellow candies. The kids take turns. Each one decides how many candies to pop out of the dispenser according to his vote. Note that no other kid can see the number or the colors of these candies. Also, it is forbidden for the participants to weight the dispenser and, thus, deduce the number of remaining candies. When this process ends, the color of the candy on top has to correspond to the correct majority vote. The voting process is completed when one of the kids pops an additional candy and announces its color.

"Phonebook" Solution. Khovanova recalls on her blog [156] that, for explaining one-way functions, Micali used the following example of encryption. We start by assuming that Alice and Bob obtain the same edition of the white pages book for a particular town. For each letter Alice wants to encrypt, she finds a person in the book whose last name starts with this letter and uses his/her phone number as the encrypted version of that letter. To decrypt the message Bob has to read through the whole book to find all the numbers. The decryption will take a lot more time than the encryption. Unfortunately, the technology changes and the example is not up to date anymore: reverse look-up is always possible in a digital world. Furthermore, regarding the security of the scheme, an 8^{th} grader said: "If I were Bob, I would just call all the phone numbers and ask their last names." A similar example may be found in [44]. Such examples are very good for teaching one-way functions to non-mathematicians.

[7] entitled "Pontifex" in the book

"Colors" Solution. The Diffie-Hellman protocol can be depicted using colors as further presented. An illustration using common paint may be found in [18]. The idea, first proposed by Simon Singh [229], relies on two properties of colors: ① it is easy to mix two colors and ② given a color that was obtained by mixing two other colors, it is difficult to reverse the process[8]. As a specific example, we may assume that yellow ▪ is a public color. Let us further consider that Alice's secret color is blue ▪ and that Bob's secret color is red ▪. The parties wish to agree on a new shared secret color. In the first step, Alice sends green ▪ to Bob (*i.e.* the result of yellow ▪ mixed with blue ▪). Then, Bob sends orange ▪ to Alice (*i.e.* the result of yellow ▪ mixed with red ▪). By mixing the received color with the secret color, each party obtains the common secret brown ▪ (*i.e.* Alice mixes orange ▪ with her blue ▪ and Bob mixes green ▪ with red ▪).

Although insecure[9], the digital version of the above protocol is a good teaching tool *e.g.* when trying to explain beginners how to use colors in the case of programming languages used in web development.

M Physical Public Key Encryption

We further present a generic protocol based on the protocols described in [130]. Alice and Bob have access to a physical medium characterized by a parameter $p(t)$, such that $p(t)$ has two components $p = p_a(t) \circ p_b(t)$, where $\circ$ is a group law and $p_a(t)$, $p_b(t)$ can randomly be changed by varying t. In her private spaces U and V, Alice and Bob secretly vary $p_a(t)$ and, respectively, $p_b(t)$. Note that Eve only has access to $p(t)$. First Alice and Bob randomly vary $p_a(t)$ and $p_b(t)$. When they agree to synchronize[10], Alice and Bob stabilize their parameters $p_a(t') = a$ and $p_b(t') = b$. Bob can measure $p(t') = a \circ b$ and deduce Alice's value a. Similarly, Alice can compute b.

Example M.1. We consider the setup from Section 7.1.3. Thus, the components that Alice and Bob vary are their corresponding speeds values a and b. Once the system is stabilized Bob can deduce a using the attack we described in Section 7.1.3, but Eve can only deduce $b - a$.

[8] and obtain the initial colors

[9] When mixing two colors which can be described in the RGB (Red-Green-Blue) color model one can revert the process due to the uniqueness of each color. Note that such a phenomenon does not happen when working with paint.

[10] through the use of an authenticated channel

Bibliography

[1] A Full Exploit of CVE-2017-3000 on Flash Player Constant Blinding PRNG. https://github.com/dangokyo/CVE-2017-3000/blob/master/Exploiter.as.

[2] Bitcoin: Average Confirmation Time. https://www.blockchain.com/charts/avg-confirmation-time.

[3] C++ Random Library. www.cplusplus.com/reference/random/.

[4] eSTREAM: the ECRYPT Stream Cipher Project. http://www.ecrypt.eu.org/stream/.

[5] Falstad Electronic Circuit. https://www.falstad.com.

[6] Frequently Asked Questions About Netflix Billing. https://help.netflix.com/en/node/41049?ui_action=kb-article-popular-categories.

[7] How to Manage Your Prime Video Channel Subscriptions. https://www.amazon.com/gp/help/customer/display.html?nodeId=201975160.

[8] How to Order HBO: Subscriptios & Pricing Options. https://www.hbo.com/ways-to-get.

[9] Kryptos. https://en.wikipedia.org/wiki/Kryptos.

[10] Left Shift and Right Shift Operators. https://docs.microsoft.com/en-us/cpp/cpp/left-shift-and-right-shift-operators-input-and-output?view=vs-2017.

[11] mbed TLS. https://tls.mbed.org.

[12] Mining Hardware Comparison. https://en.bitcoin.it/wiki/Mining_hardware_comparison.

[13] NIST SP 800-22: Download Documentation and Software. https://csrc.nist.gov/Projects/Random-Bit-Generation/Documentation-and-Software.

[14] Non-Specialized Hardware Comparison. https://en.bitcoin.it/wiki/Non-specialized_hardware_comparison.

[15] OpenMP. https://www.openmp.org/.

[16] Safe Prime Database. https://2ton.com.au/safeprimes/.

[17] Source Code for the Actionscript Virtual Machine. https://github.com/adobe-flash/avmplus/tree/master/core/MathUtils.cpp.

[18] The Diffie-Hellman Key Exchange Using Paint. https://www.youtube.com/watch?v=3QnD2c4Xovk.

[19] The GNU Multiple Precision Arithmetic Library. https://gmplib.org/.

[20] Using the GNU Compiler Collection. https://gcc.gnu.org/onlinedocs/gcc/Integers-implementation.html.

[21] Vulnerability Details: CVE-2017-3000. https://www.cvedetails.com/cve/CVE-2017-3000/.

[22] World Map of Encryption Laws and Policies. https://www.gp-digital.org/world-map-of-encryption/.

[23] FIPS PUB 186-4: Digital Signature Standard (DSS). Technical report, NIST, 2013.

[24] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. DHAES: An Encryption Scheme Based on the Diffie-Hellman Problem. *IACR Cryptology ePrint Archive*, 1999/7, 1999.

[25] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The Oracle Diffie-Hellman Assumptions and an Analysis of DHIES. In *CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2001.

[26] Carlisle Adams, Pat Cain, Denis Pinkas, and Robert Zuccherato. RFC 3161: Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). Technical report, Internet Engineering Task Force, 2001.

[27] Gorjan Alagic and Alexander Russell. Quantum-Secure Symmetric-Key Cryptography Based on Hidden Shifts. In *EUROCRYPT 2018*, volume 10212 of *Lecture Notes in Computer Science*, pages 65–93. Springer, 2017.

[28] Ange Albertini, Jean-Philippe Aumasson, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Malicious Hashing: Eve's Variant of SHA-1. In *SAC 2014*, volume 8781 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2014.

[29] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic Protocols for Fair Exchange. In *CCS 1997*, pages 7–17. ACM, 1997.

[30] American Bankers Association et al. Working Draft: American National Standard X9. 62-1998 Public Key Cryptography for the Financial Services Industry. Technical report, 1998.

[31] Giuseppe Ateniese and Paolo Gasti. Universally Anonymous IBE Based on the Quadratic Residuosity Assumption. In *CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 32–47. Springer, 2009.

[32] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-Resilient Signature Schemes. In *CCS 2015*, pages 364–375. ACM, 2015.

[33] Michalis Athanasakis, Elias Athanasopoulos, Michalis Polychronakis, Georgios Portokalidis, and Sotiris Ioannidis. The Devil is in the Constants: Bypassing Defences in Browser JIT Engines. In *NDSS 2015*. The Internet Society, 2015.

[34] Jean-Philippe Aumasson, Itai Dinur, Luca Henzen, Willi Meier, and Adi Shamir. Efficient FPGA Implementations of High-Dimensional Cube Testers on the Stream Cipher Grain-128. *IACR Cryptology ePrint Archive*, 2009/218, 2009.

[35] Shahram Bakhtiari, Reihaneh Safavi-Naini, and Josef Pieprzyk. A Message Authentication Code Based on Latin Squares. In *ACISP 1997*, volume 1270 of *Lecture Notes in Computer Science*, pages 194–203. Springer, 1997.

[36] James Ball, Julian Borger, and Glenn Greenwald. Revealed: How US and UK Spy Agencies Defeat Internet Privacy and Security. *The Guardian*, 6, 2013.

[37] József Balogh, János A Csirik, Yuval Ishai, and Eyal Kushilevitz. Private Computation Using a PEZ Dispenser. *Theoretical Computer Science*, 306(1-3):69–84, 2003.

[38] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. Some Results on Related Key-IV Pairs of Grain. In *SPACE 2012*, volume 7644 of *Lecture Notes in Computer Science*, pages 94–110. Springer, 2012.

[39] Subhadeep Banik, Subhamoy Maitra, Santanu Sarkar, and Turan Meltem Sönmez. A Chosen IV Related Key Attack on Grain-128a. In *ACISP 2013*, volume 7959 of *Lecture Notes in Computer Science*, pages 13–26. Springer, 2013.

[40] Manuel Barbosa, Thierry Brouard, Stéphane Cauchie, and Simao Melo De Sousa. Secure Biometric Authentication with Improved Accuracy. In *ACISP 2008*, volume 5107 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2008.

[41] Craig Bauer, Gregory Link, and Dante Molle. James Sanborn's Kryptos and the Matrix Encryption Conjecture. *Cryptologia*, 40(6):541–552, 2016.

[42] Craig Bauer and Katherine Millward. Cracking Matrix Encryption Row by Row. *Cryptologia*, 31(1):76–83, 2007.

[43] Friedrich Ludwig Bauer. *Decrypted Secrets: Methods and Maxims of Cryptology*. Springer, 2002.

[44] Tim Bell, Harold Thimbleby, Mike Fellows, Ian Witten, Neil Koblitz, and Matthew Powell. Explaining Cryptographic Systems. *Computers & Education*, 40(3):199–215, 2003.

[45] Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-Surveillance without the State: Strongly Undetectable Algorithm-Substitution Attacks. In *CCS 2015*, pages 1431–1440. ACM, 2015.

[46] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Security Proofs for Identity-Based Identification and Signature Schemes. *Journal of Cryptology*, 22(1):1–61, 2009.

[47] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of Symmetric Encryption Against Mass Surveillance. In *CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2014.

[48] Mihir Bellare and Phillip Rogaway. Minimizing the Use of Random Oracles in Authenticated Encryption Schemes. In *ICICS 1997*, volume 1334 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 1997.

[49] Mihir Bellare and Phillip Rogaway. Introduction to Modern Cryptography. https://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf, 2005.

[50] Luciano Bello. DSA-1571-1 OpenSSL—Predictable Random Number Generator. https://www.debian.org/security/2008/dsa-1571, 2008.

[51] Fabrice Benhamouda, Javier Herranz, Marc Joye, and Benoît Libert. Efficient Cryptosystems from 2^k-th Power Residue Symbols. *Journal of Cryptology*, 30(2):519–549, 2017.

[52] Côme Berbain, Henri Gilbert, and Alexander Maximov. Cryptanalysis of Grain. In *FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2006.

[53] Sebastian Berndt and Maciej Liśkiewicz. Algorithm Substitution Attacks from a Steganographic Perspective. In *CCS 2017*, pages 1649–1660. ACM, 2017.

[54] Daniel J. Bernstein, Tanja Lange, and Ruben Niederhagen. Dual EC: A Standardized Back Door. In *The New Codebreakers*, volume 9100 of *Lecture Notes in Computer Science*, pages 256–281. Springer, 2016.

[55] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *CRYPTO 1990*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1991.

[56] Dionysus Blazakis. Interpreter Exploitation. In *WOOT 2010*. USENIX Association, 2010.

[57] Jens-Matthias Bohli, Maria Isabel Gonzalez Vasco, and Rainer Steinwandt. A subliminal-free variant of ECDSA. In *IH 2006*, volume 4437 of *Lecture Notes in Computer Science*, pages 375–387. Springer, 2006.

[58] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public Key Encryption with Keyword Search. In *EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[59] Dan Boneh and Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[60] Dan Boneh, Craig Gentry, and Michael Hamburg. Space-efficient Identity Based Encryption Without Pairings. In *FOCS 2007*, pages 647–657. IEEE Computer Society, 2007.

[61] Julien Bringer, Hervé Chabanne, Malika Izabachéne, David Pointcheval, Qiang Tang, and Sébastien Zimmer. An Application of the Goldwasser-Micali Cryptosystem to Biometric Authentication. In *ACISP 2007*, pages 96–106. Springer, 2007.

[62] Xavier Bultel, Jannik Dreier, Pascal Lafourcade, and Malika More. How to explain modern security concepts to your children. *Cryptologia*, 41(5):422–447, 2017.

[63] Christian Cachin and Jan Camenisch. Optimistic Fair Secure Computation. In *CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 93–111. Springer, 2000.

[64] Christophe Cannière, Özgül Küçük, and Bart Preneel. Analysis of Grain's Initialization Algorithm. In *AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 276–289. Springer, 2008.

[65] Anne Canteaut, Pascale Charpin, and Hans Dobbertin. Weight Divisibility of Cyclic Codes, Highly Nonlinear Functions on F_{2^m}, and Crosscorrelation of Maximum-Length Sequences. *SIAM J. Discrete Math.*, 13(1):105–138, 2000.

[66] Héctor Martín Cantero, Sven Peter, and Segher Bushing. Console Hacking 2010–PS3 Epic Fail. In *27th Chaos Communication Congress*, 2010.

[67] Charalambos A Charalambides. *Enumerative Combinatorics*. Chapman and Hall/CRC, 2002.

[68] David Chaum, Jan-Hendrik Evertse, and Jeroen Van De Graaf. An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations. In *EUROCRYPT 1987*, volume 304 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 1987.

[69] Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. A Systematic Analysis of the Juniper Dual EC Incident. In *CCS 2016*, pages 468–479. ACM, 2016.

[70] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the Practical Exploitability of Dual EC in TLS Implementations. In *USENIX Security Symposium*, pages 319–335. USENIX Association, 2014.

[71] Liqun Chen, Caroline Kudla, and Kenneth G. Paterson. Concurrent Signatures. In *EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 287–305. Springer, 2004.

[72] Benoît Chevallier-Mames. An Efficient CDH-Based Signature Scheme with a Tight Security Reduction. In *CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 511–526. Springer, 2005.

[73] Jong Youl Choi, Philippe Golle, and Markus Jakobsson. Tamper-Evident Digital Signature Protecting Certification Authorities Against Malware. In *DASC 2006*, pages 37–44. IEEE, 2006.

[74] Jae Cha Choon and Jung Hee Cheon. An Identity-Based Signature from Gap Diffie-Hellman Groups. In *PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 18–30. Springer, 2003.

[75] Nicolas Christin. Traveling the Silk Road: A Measurement Analysis of a Large Anonymous Online Marketplace. In *WWW 2013*, pages 213–224. ACM, 2013.

[76] Michael Clear, Hitesh Tewari, and Ciarán McGoldrick. Anonymous IBE from Quadratic Residuosity with Improved Performance. In *AFRICACRYPT 2014*, volume 8469 of *Lecture Notes in Computer Science*, pages 377–397. Springer, 2014.

[77] Clifford Cocks. An Identity Based Encryption Scheme Based on Quadratic Residues. In *IMACC 2001*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer, 2001.

[78] Simon Cogliani, Bao Feng, Houda Ferradi, Rémi Géraud, Diana Maimuţ, David Naccache, Rodrigo Portella do Canto, and Guilin Wang. Public Key-Based Lightweight Swarm Authentication. In *Cyber-Physical Systems Security*, pages 255–267. Springer, 2018.

[79] Josh Cohen and Michael Fischer. A Robust and Verifiable Cryptographically Secure Ellection Scheme (extended abstract). In *FOCS 1985*, pages 372–382. IEEE Computer Society Press, 1985.

[80] Mariana Costiuc, Diana Maimuţ, and George Teşeleanu. Physical Cryptography. In *SECITC 2019*, volume 12001 of *Lecture Notes in Computer Science*, pages 156–171. Springer, 2019.

[81] Nicolas T. Courtois. Cryptanalysis of Grigoriev-Shpilrain Physical Asymmetric Scheme With Capacitors. *IACR Cryptology ePrint Archive*, 2013/302, 2013.

[82] Richard Crandall and Carl Pomerance. *Prime Numbers: A Computational Perspective*. Number Theory and Discrete Mathematics. Springer, 2005.

[83] Claude Crépeau and Alain Slakmon. Simple Backdoors for RSA Key Generation. In *CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 403–416. Springer, 2003.

[84] Paul Crowley. Mirdek: A Card Cipher Inspired by "Solitaire". http://www.ciphergoth.org/crypto/mirdek/.

[85] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Science & Business Media, 2013.

[86] Harold Davenport. On the Distribution of Quadratic Residues (mod p). *Journal of the London Mathematical Society*, s1-6(1):49–54, 1931.

[87] Harold Davenport. On the Distribution of Quadratic Residues (mod p). *Journal of the London Mathematical Society*, s1-8(1):46–52, 1933.

[88] Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A More Cautious Approach to Security Against Mass Surveillance. In *FSE 2015*, volume 9054 of *Lecture Notes in Computer Science*, pages 579–598. Springer, 2015.

[89] Jean Paul Degabriele, Kenneth G. Paterson, Jacob CN Schuldt, and Joanne Woodage. Backdoors in Pseudorandom Number Generators: Possibility and Impossibility Results. In *CRYPTO 2016*, volume 9814 of *Lecture Notes in Computer Science*, pages 403–432. Springer, 2016.

[90] József Dénes and A Donald Keedwell. A New Authentication Scheme Based on Latin Squares. *Discrete Mathematics*, 106:157–161, 1992.

[91] Itai Dinur, Tim Güneysu, Christof Paar, Adi Shamir, and Ralf Zimmermann. An Experimentally Verified Attack on Full Grain-128 Using Dedicated Reconfigurable Hardware. In *ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 327–343. Springer, 2011.

[92] Itai Dinur and Adi Shamir. Breaking Grain-128 with Dynamic Cube Attacks. In *FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 167–187. Springer, 2011.

[93] Hans Dobbertin. One-to-One Highly Nonlinear Power Functions on GF(2^n). *Appl. Algebra Eng. Commun. Comput.*, 9(2):139–152, 1998.

[94] Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A Formal Treatment of Backdoored Pseudorandom Generators. In *EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 101–126. Springer, 2015.

[95] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes. In *CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2004.

[96] Yevgeniy Dodis, Ilya Mironov, and Noah Stephens-Davidowitz. Message Transmission with Reverse Firewalls—Secure Communication on Corrupted Machines. In *CRYPTO 2016*, volume 9814 of *Lecture Notes in Computer Science*, pages 341–372. Springer, 2016.

[97] Vasily Dolmatov and Alexey Degtyarev. GOST R 34.10-2012: Digital Signature Algorithm. Technical report, Internet Engineering Task Force, 2013.

[98] Morris Dworkin. Recommendation for Block Cipher Modes of Operation. Methods and Techniques. Technical report, NIST, 2001.

[99] Ibrahim Elashry, Yi Mu, and Willy Susilo. Jhanwar-Barua's Identity-Based Encryption Revisited. In *NSS 2014*, volume 8792 of *Lecture Notes in Computer Science*, pages 271–284. Springer, 2014.

[100] Ibrahim Elashry, Yi Mu, and Willy Susilo. An Efficient Variant of Boneh-Gentry-Hamburg's Identity-Based Encryption Without Pairing. In *WISA 2014*, volume 8909 of *Lecture Notes in Computer Science*, pages 257–268. Springer, 2015.

[101] Taher ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[102] Ronald Fagin, Moni Naor, and Peter Winkler. Comparing Information Without Leaking It. *Communications of the ACM*, 39(5):77–85, 1996.

[103] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-Knowledge Proofs of Identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[104] Houda Ferradi, Rémi Géraud, Diana Maimuţ, David Naccache, and David Pointcheval. Legally Fair Contract Signing Without Keystones. In *ACNS 2016*, volume 9696 of *Lecture Notes in Computer Science*, pages 175–190. Springer, 2016.

[105] Houda Ferradi, Rémi Géraud, Diana Maimuţ, David Naccache, and Amaury de Wargny. Regulating the Pace of von Neumann Correctors. *Journal of Cryptographic Engineering*, pages 1–7, 2017.

[106] Amos Fiat. Batch RSA. In *CRYPTO 1989*, volume 435 of *Lecture Notes in Computer Science*, pages 175–185. Springer, 1989.

[107] Amos Fiat. Batch RSA. *J. Cryptology*, 10(2):75–88, 1997.

[108] Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO 1986*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[109] Marc Fischlin, Christian Janson, and Sogol Mazaheri. Backdoored Hash Functions: Immunizing HMAC and HKDF. *IACR Cryptology ePrint Archive*, 2018/362, 2018.

[110] Joshua Fried, Pierrick Gaudry, Nadia Heninger, and Emmanuel Thomé. A Kilobit Hidden SNFS Discrete Logarithm Computation. In *EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 202–231. Springer, 2017.

[111] Juan Garay, Philip MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource Fairness and Composability of Cryptographic Protocols. In *TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 404–428. Springer, 2006.

[112] Rosario Gennaro, Hugo Krawczyk, and Tal Rabin. Secure Hashed Diffie-Hellman over Non-DDH Groups. In *EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 361–381. Springer, 2004.

[113] Marc Girault. An Identity-based Identification Scheme Based on Discrete Logarithms Modulo a Composite Number. In *EUROCRYPT 1990*, volume 473 of *Lecture Notes in Computer Science*, pages 481–486. Springer, 1990.

[114] Marc Girault, Guillaume Poupard, and Jacques Stern. On the Fly Authentication and Signature Schemes Based on Groups of Unknown Order. *Journal of Cryptology*, 19(4):463–487, 2006.

[115] Marc Girault and Jacques Stern. On the Length of Cryptographic Hash-Values Used in Identification Schemes. In *CRYPTO 1994*, volume 839 of *Lecture Notes in Computer Science*, pages 202–215. Springer, 1994.

[116] Danilo Gligoroski, Smile Markovski, and Svein Johan Knapskog. The Stream Cipher Edon80. In *New Stream Cipher Designs*, volume 4986 of *Lecture Notes in Computer Science*, pages 152–169. Springer, 2008.

[117] Danilo Gligoroski, Smile Markovski, and Ljupco Kocarev. Edon-R, An Infinite Family of Cryptographic Hash Functions. *I.J. Network Security*, 8(3):293–300, 2009.

[118] Eu-Jin Goh and Stanisław Jarecki. A Signature Scheme as Secure as the Diffie-Hellman Problem. In *EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 401–415. Springer, 2003.

[119] Dirk Goldhahn, Thomas Eckart, and Uwe Quasthoff. Building Large Monolingual Dictionaries at the Leipzig Corpora Collection: From 100 to 200 Languages. In *LREC 2012*, volume 29, pages 31–43. European Language Resources Association (ELRA), 2012.

[120] Oded Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 2007.

[121] Shafi Goldwasser. Cocks' IBE Scheme. Bilinear Maps. MIT Lecture Notes: "6876: Advanced Cryptography", 2004.

[122] Shafi Goldwasser, Leonid Levin, and Scott A. Vanstone. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPT0 1990*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 1991.

[123] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information. In *STOC 1982*, pages 365–377. ACM, 1982.

[124] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[125] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[126] Daniel Gordon. Designing and Detecting Trapdoors for Discrete Log Cryptosystems. In *CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 66–75. Springer, 1993.

[127] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete Fairness in Secure Two-Party Computation. *Jornal of the ACM*, 58(6):1–37, December 2011.

[128] Dima Grigoriev, Laszlo B. Kish, and Vladimir Shpilrain. Yao's Millionaires' Problem and Public-Key Encryption Without Computational Assumptions. *Int. J. Found. Comput. Sci.*, 28(4):379–390, 2017.

[129] Dima Grigoriev and Vladimir Shpilrain. Secure Information Transmission Based on Physical Principles. In *UCNC 2013*, volume 7956 of *Lecture Notes in Computer Science*, pages 113–124. Springer, 2013.

[130] Dima Grigoriev and Vladimir Shpilrain. Yao's Millionaires' Problem and Decoy-Based Public Key Encryption by Classical Physics. *Int. J. Found. Comput. Sci.*, 25(4):409–418, 2014.

[131] Louis C. Guillou and Jean-Jacques Quisquater. A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory. In *EUROCRYPT 1988*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128. Springer, 1988.

[132] Stuart Haber and W Scott Stornetta. How to Time-Stamp a Digital Document. In *CRYPTO 1990*, volume 537 of *Lecture Notes in Computer Science*, pages 437–455. Springer, 1990.

[133] D. Halliday, R. Resnick, and J. Walker. *Fundamentals of Physics*. John Wiley & Sons, 2010.

[134] Mike Hamburg, Paul Kocher, and Mark E Marson. Analysis of Intel's Ivy Bridge Digital Random Number Generator. Technical report, Rambus, 2012.

[135] Lucjan Hanzlik, Kamil Kluczniak, and Mirosław Kutyłowski. Controlled Randomness - A Defense against Backdoors in Cryptographic Devices. In *MyCrypt 2016*, volume 10311 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2016.

[136] Dan Harkins and Dave Carrel. RFC 2409: The Internet Key Exchange (IKE). Technical report, Internet Engineering Task Force, 1998.

[137] Sam Hasinoff. Solving Substitution Ciphers. https://people.csail.mit.edu/hasinoff/pubs/hasinoff-quipster-2003.pdf.

[138] Philip Hawkes and Luke O'Connor. XOR and Non-XOR Differential Probabilities. In *EUROCRYPT 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 272–285. Springer, 1999.

[139] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A Stream Cipher Proposal: Grain-128. In *ISIT 2006*, pages 1614–1618. IEEE, 2006.

[140] Martin Hell, Thomas Johansson, and Willi Meier. Grain - A Stream Cipher for Constrained Environments. Technical Report 010, ECRYPT Stream Cipher Project Report, 2005.

[141] Martin Hell, Thomas Johansson, and Willi Meier. Grain: A Stream Cipher for Constrained Environments. *International Journal of Wireless and Mobile Computing*, 2(1):86–93, May 2007.

[142] Florian Hess. Efficient Identity Based Signature Schemes Based On Pairings. In *SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 310–324. Springer, 2002.

[143] Howard M Heys. A Tutorial on Linear and Differential Cryptanalysis. *Cryptologia*, 26(3):189–221, 2002.

[144] Lester S Hill. Cryptography in an Algebraic Alphabet. *The American Mathematical Monthly*, 36(6):306–312, 1929.

[145] Lester S Hill. Concerning Certain Linear Transformation Apparatus of Cryptography. *The American Mathematical Monthly*, 38(3):135–154, 1931.

[146] Susan M Howitt and Anna N Wilson. Revisiting "Is the Scientific Paper a Fraud?". *EMBO Reports*, 15(5):481–484, 2014.

[147] Mahabir Prasad Jhanwar and Rana Barua. A Variant of Boneh-Gentry-Hamburg's Pairing-Free Identity Based Encryption Scheme. In *INSCRYPT 2008*, volume 5487 of *Lecture Notes in Computer Science*, pages 314–331. Springer, 2009.

[148] Marc Joye. Identity-Based Cryptosystems and Quadratic Residuosity. In *PKC 2016*, volume 9614 of *Lecture Notes in Computer Science*, pages 225–254. Springer, 2016.

[149] Marc Joye and Benoît Libert. Efficient Cryptosystems from 2^k-th Power Residue Symbols. In *EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 76–92. Springer, 2013.

[150] Marc Joye and Benoît Libert. Efficient Cryptosystems from 2^k-th Power Residue Symbols. *IACR Cryptology ePrint Archive*, 2013/435, 2014.

[151] Benjamin Justus. The Distribution of Quadratic Residues and Non-Residues in the Goldwasser-Micali Type of Cryptosystem. *Journal of Mathematical Cryptology*, 8(8):115–140, 2014.

[152] Jonathan Katz and Nan Wang. Efficiency Improvements for Signature Schemes With Tight Security Reductions. In *CCS 2003*, pages 155–164. ACM, 2003.

[153] Charlie Kaufman, Paul Hoffman, Yoav Nir, Parsi Eronen, and Tero Kivinen. RFC7296: Internet Key Exchange Protocol Version 2 (IKEv2). Technical report, Internet Engineering Task Force, 2014.

[154] Shahram Khazaei and Siavash Ahmadi. Ciphertext-Only Attack on $d \times d$ Hill in $O(d13^d)$. *Information Processing Letters*, 118:25–29, 2017.

[155] Shahram Khazaei, Mehdi Hassanzadeh, and Mohammad Kiaei. Distinguishing Attack on Grain. Technical Report 071, ECRYPT Stream Cipher Project Report, 2005.

[156] Tanya Khovanova. One-Way Functions. https://blog.tanyakhovanova.com/2010/11/one-way-functions/.

[157] William A Kiele. A Tensor-Theoretic Enhancement to the Hill Cipher System. *Cryptologia*, 14(3):225–233, 1990.

[158] Wolfgang Killmann and Werner Schindler. A Proposal for: Functionality Classes for Random Number Generators, version 2.0. Technical report, BSI, 2011.

[159] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional Differential cryptanalysis of NLFSR-Based Cryptosystems. In *ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2010.

[160] Czesław Kościelny. A Method of Constructing Quasigroup-Based Stream-Ciphers. *Applied Mathematics and Computer Science*, 6:109–122, 1996.

[161] Daniel Kucner and Mirosław Kutyłowski. Stochastic kleptography detection. In *Public-Key Cryptography and Computational Number Theory*, pages 137–149, 2001.

[162] Özgül Küçük. Slide Resynchronization Attack on the Initialization of Grain 1.0. http:www.ecrypt.eu.org/stream, 2006.

[163] Robin Kwant, Tanja Lange, and Kimberley Thissen. Lattice Klepto - Turning Post-Quantum Crypto Against Itself. In *SAC 2017*, volume 10719 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2017.

[164] Xuejia Lai and James L Massey. A Proposal for a New Block Encryption Standard. In *EUROCRYPT 1990*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 1991.

[165] Xuejia Lai, James L Massey, and Sean Murphy. Markov Ciphers and Differential Cryptanalysis. In *EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, 1991.

[166] Butler W Lampson. A Note on the Confinement Problem. *Communications of the ACM*, 16(10):613–615, 1973.

[167] Tom Leap, Tim McDevitt, Kayla Novak, and Nicolette Siermine. Further Improvements to the Bauer-Millward Attack on the Hill Cipher. *Cryptologia*, 40(5):452–468, 2016.

[168] Chae Hoon Lim and Pil Joong Lee. A Study on the Proposed Korean Digital Signature Algorithm. In *ASIACRYPT 1998*, volume 1514 of *Lecture Notes in Computer Science*, pages 175–186. Springer, 1998.

[169] Yehuda Lindell. Fast Secure Two-Party ECDSA Signing. In *CRYPTO 2017*, volume 10402 of *Lecture Notes in Computer Science*, pages 613–644. Springer, 2017.

[170] James Lyons. Practical Cryptography, http://practicalcryptography.com/.

[171] Diana Maimuţ and George Teşeleanu. A Unified Security Perspective on Legally Fair Contract Signing Protocols. In *SECITC 2018*, volume 11359 of *Lecture Notes in Computer Science*, pages 477–491. Springer, 2018.

[172] Diana Maimuţ and George Teşeleanu. New Configurations of Grain Ciphers: Security Against Slide Attacks. In *BalkanCrypt 2018*, Communications in Computer and Information Science. Springer, 2018.

[173] Diana Maimuţ and George Teşeleanu. A Generic View on the Unified Zero-Knowledge Protocol and its Applications. In *WISTP 2019*, volume 12024 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2019.

[174] Diana Maimuţ and George Teşeleanu. A New Generalisation of the Goldwasser-Micali Cryptosystem Based on the Gap 2^k-Residuosity Assumption. In *SECITC 2020*, Lecture Notes in Computer Science. Springer, 2020.

[175] John Malone-Lee and Nigel P. Smart. Modifications of ECDSA. In *SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2002.

[176] Ueli Maurer. Unifying Zero-Knowledge Proofs of Knowledge. In *AFRICACRYPT 2009*, volume 5580 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2009.

[177] Kevin McCurley. A Key distribution System Equivalent to Factoring. *Journal of cryptology*, 1(2):95–105, 1988.

[178] Tim McDevitt, Jessica Lehr, and Ting Gu. A Parallel Time-memory Tradeoff Attack on the Hill Cipher. *Cryptologia*, 42(5):1–19, 2018.

[179] Peter Medawar. Is the Scientific Paper a Fraud? *The Listener*, 70(12):377–378, 1963.

[180] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC press, 1996.

[181] Silvio Micali. Simple and Fast Optimistic Protocols for Fair Electronic Exchange. In *PODC 2003*, pages 12–19. ACM, 2003.

[182] Markus Michels, David Naccache, and Holger Petersen. GOST 34.10-A Brief Overview of Russia's DSA. *Computers & Security*, 15(8):725–732, 1996.

[183] Microprocessor, MS Committee, et al. IEEE Standard Specifications for Public-Key Cryptography. *IEEE Computer Society*, 2000.

[184] Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic Reverse Firewalls. In *ASIACRYPT 2015*, volume 9057 of *Lecture Notes in Computer Science*, pages 657–686. Springer, 2015.

[185] Arjan J. Mooij, Nicolae Goga, and Jan Willem Wesselink. *A Distributed Spanning Tree Algorithm for Topology-Aware Networks*. Technische Universiteit Eindhoven, Department of Mathematics and Computer Science, 2003.

[186] Tal Moran and Moni Naor. Polling with Physical Envelopes: A rigorous Analysis of a Human-Centric Protocol. In *EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 88–108. Springer, 2006.

[187] Tal Moran and Moni Naor. Basing Cryptographic Protocols on Tamper-Evident Seals. *Theoretical Computer Science*, 411(10):1283–1310, 2010.

[188] Nicky Mouha. On Proving Security against Differential Cryptanalysis. In *CFAIL 2019*, 2019.

[189] David M'Raïhi, David Naccache, David Pointcheval, and Serge Vaudenay. Computational Alternatives to Random Number Generators. In *SAC 1998*, volume 1556 of *Lecture Notes in Computer Science*, pages 72–80. Springer, 1998.

[190] David Naccache and Jacques Stern. A New Public Key Cryptosytem Based on Higher Residues. In *CCS 1998*, pages 59–66. ACM, 1998.

[191] Moni Naor, Yael Naor, and Omer Reingold. Applied Kid Cryptography or How to Convince Your Children You Are Not Cheating. http://www.wisdom.weizmann.ac.il/~naor/PAPERS/waldo.pdf.

[192] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS 1997*, pages 458–467. IEEE Computer Society, 1997.

[193] Moni Naor and Omer Reingold. Number-Theoretic Constructions of Efficient Pseudo-Random Functions. *Journal of the ACM*, 51(2):231–262, 2004.

[194] Melvyn B. Nathanson. *Elementary Methods in Number Theory*. Graduate Texts in Mathematics. Springer, 2000.

[195] Koki Nishigami and Keiichi Iwamura. Geometric pairwise key-sharing scheme. In *SECITC 2018*, volume 11359 of *Lecture Notes in Computer Science*, pages 518–528. Springer, 2018.

[196] Kaisa Nyberg. Perfect Nonlinear S-boxes. In *EUROCRYPT 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 378–386. Springer, 1991.

[197] Kaisa Nyberg and Rainer A. Rueppel. A New Signature Scheme Based on the DSA Giving Message Recovery. In *CCS 1993*, pages 58–61. ACM, 1993.

[198] Luke O'Connor. On the Distribution of Characteristics in Bijective Mappings. In *EUROCRYPT 1993*, volume 765 of *Lecture Notes in Computer Science*, pages 360–370. Springer, 1994.

[199] Luke O'Connor. On the Distribution of Characteristics in Bijective Mappings. *Journal of Cryptology*, 8(2):67–86, 1995.

[200] Tatsuaki Okamoto. Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes. In *CRYPTO 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53. Springer, 1992.

[201] Jeffrey Overbey, William Traves, and Jerzy Wojdylo. On the Keyspace of the Hill Cipher. *Cryptologia*, 29(1):59–72, 2005.

[202] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Eurocrypt 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.

[203] Kenneth G. Paterson. ID-Based Signatures from Pairings on Elliptic Curves. *Electronics Letters*, 38(18):1025–1026, 2002.

[204] René Peralta. On the Distribution of Quadratic Residues and Nonresidues Modulo a Prime Number. *Mathematics of Computation*, 58(197):433–440, 1992.

[205] Nicole Perlroth, Jeff Larson, and Scott Shane. NSA Able to Foil Basic Safeguards of Privacy on Web. *The New York Times*, 5, 2013.

[206] Oskar Perron. Bemerkungen über die Verteilung der quadratischen Reste. *Mathematische Zeitschrift*, 56(2):122–130, 1952.

[207] Benny Pinkas. Fair Secure Two-Party Computation. In *EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 87–105. Springer, 2003.

[208] David Pointcheval and Jacques Stern. Security Proofs For Signature Schemes. In *EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer, 1996.

[209] David Pointcheval and Jacques Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

[210] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaël Quisquater, Louis Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, and Soazig Guillou. How to Explain Zero-Knowledge Protocols to Your Children. In *CRYPTO 1989*, volume 435 of *Lecture Notes in Computer Science*, pages 628–631. Springer, 1990.

[211] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: A New Version of Grain-128 with Optional Authentication. *International Journal of Wireless and Mobile Computing*, 5(1):48–59, December 2011.

[212] Elena Reshetova, Filippo Bonazzi, and N. Asokan. Randomization Can't Stop BPF JIT spray. In *NSS 2017*, volume 10394 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2017.

[213] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock Puzzles and Timed-release Crypto. Technical report, MIT, 1996.

[214] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. In *ASIACRYPT 2016*, volume 10032 of *Lecture Notes in Computer Science*, pages 34–64. Springer, 2016.

[215] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Destroying Steganography via Amalgamation: Kleptographically CPA Secure Public Key Encryption. *IACR Cryptology ePrint Archive*, 2016/530, 2016.

[216] Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. Cryptosystems Based on Pairings. In *SCIS 2000*, 2000.

[217] Conrad Sanderson and Ryan Curtin. Armadillo: A Template-Based C++ Library for Linear Algebra. *Journal of Open Source Software*, 1(2):26, 2016.

[218] Bruce Schneier. The Solitaire Encryption Algorithm. https://www.schneier.com/academic/solitaire/.

[219] Claus-Peter Schnorr. Efficient Identification and Signatures For Smart Cards. In *CRYPTO 1989*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.

[220] Martin A Schwartz. The Importance of Stupidity in Scientific Research. *Journal of Cell Science*, 121(11):1771–1771, 2008.

[221] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[222] Adi Shamir. Identity-Based Cryptosystems and Signature Schemes. In *CRYPTO 1984*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1985.

[223] Victor Shoup. Sequences of Games: A Tool for Taming Complexity in Security Proofs. *IACR Cryptology ePrint Archive*, 2004/332, 2004.

[224] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2008.

[225] Vladimir Shpilrain. Decoy-Based Information Security. *Groups Complexity Cryptology*, 6(2):149–155, 2014.

[226] Gustavus J. Simmons. The Subliminal Channel and Digital Signatures. In *EUROCRYPT 1984*, volume 209 of *Lecture Notes in Computer Science*, pages 364–378. Springer, 1984.

[227] Gustavus J. Simmons. Subliminal Communication is Easy Using the DSA. In *EUROCRYPT 1993*, volume 765 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 1993.

[228] Gustavus J Simmons. Subliminal Channels; Past and Present. *European Transactions on Telecommunications*, 5(4):459–474, 1994.

[229] Simon Singh. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Anchor, 2000.

[230] Jonathan DH Smith. Four Lectures on Quasigroup Representations. *Quasigroups Related Systems*, 15:109–140, 2007.

[231] Paul Stankovski. Greedy Distinguishers and Nonrandomness Detectors. In *INDOCRYPT 2010*, volume 6498 of *Lecture Notes in Computer Science*, pages 210–226. Springer, 2010.

[232] Neal Stephenson. *Cryptonomicon*. Arrow, 2000.

[233] Douglas R Stinson. *Cryptography: Theory and Practice*. CRC press, 2005.

[234] Fatih Sulak. New Statistical Randomness Tests: 4-bit Template Matching Tests. *Turkish Journal of Mathematics*, 41(1):80–95, 2017.

[235] Terence Tao. Ask Yourself Dumb Questions - and Answer Them! https://terrytao.wordpress.com/career-advice/ask-yourself-dumb-questions-and-answer-them/.

[236] Terence Tao. Use The Wastebasket. https://terrytao.wordpress.com/career-advice/use-the-wastebasket/.

[237] George Teşeleanu. Threshold Kleptographic Attacks on Discrete Logarithm Based Signatures. In *LatinCrypt 2017*, volume 11368 of *Lecture Notes in Computer Science*, pages 401–414. Springer, 2017.

[238] George Teşeleanu. Random Number Generators Can Be Fooled to Behave Badly. In *ICICS 2018*, volume 11149 of *Lecture Notes in Computer Science*, pages 124–141. Springer, 2018.

[239] George Teşeleanu. Unifying Kleptographic Attacks. In *NordSec 2018*, volume 11252 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 2018.

[240] George Teşeleanu. Managing Your Kleptographic Subscription Plan. In *C2SI 2019*, volume 11445 of *Lecture Notes in Computer Science*, pages 452–461. Springer, 2019.

[241] George Teşeleanu. Reinterpreting and Improving the Cryptanalysis of the Flash Player PRNG. In *C2SI 2019*, volume 11445 of *Lecture Notes in Computer Science*, pages 92–104. Springer, 2019.

[242] George Teşeleanu. Subliminal Hash Channels. In *A2C 2019*, volume 1133 of *Communications in Computer and Information Science*, pages 149–165. Springer, 2019.

[243] George Teşeleanu. A Love Affair Between Bias Amplifiers and Broken Noise Sources. In *ICICS 2020*, Lecture Notes in Computer Science. Springer, 2020.

[244] George Teşeleanu. Cracking Matrix Modes of Operation with Goodness-of-Fit Statistics. In *HistoCrypt 2020*, Linköping Electronic Conference Proceedings. Linköping University Electronic Press, 2020.

[245] George Teşeleanu. Quasigroups and Substitution Permutation Networks: A Failed Experiment. *Cryptologia*, 2020.

[246] Ferucio Laurenţiu Ţiplea, Sorin Iftene, George Teşeleanu, and Anca-Maria Nica. Security of Identity-Based Encryption Schemes from Quadratic Residues. In *SECITC 2016*, volume 10006 of *Lecture Notes in Computer Science*, pages 63–77, 2016.

[247] Ferucio Laurentiu Tiplea, Sorin Iftene, George Teseleanu, and Anca-Maria Nica. On the Distribution of Quadratic Residues and Non-residues Modulo Composite Integers and Applications to Cryptography. *Appl. Math. Comput.*, 372, 2020.

[248] Peter Truran. *Practical Applications of the Philosophy of Science: Thinking About Research*. Springer Science & Business Media, 2013.

[249] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry McKay, Mary Baish, and Mike Boyle. NIST DRAFT Special Publication 800-90B: Recommendation for the Entropy Sources Used for Random Bit Generation. Technical report, NIST, 2012.

[250] Umesh V. Vazirani and Vijay V. Vazirani. Trapdoor Pseudo-random Number Generators, with Applications to Protocol Design. In *FOCS 1983*, pages 23–30. IEEE, 1983.

[251] Milan Vojvoda, Marek Sỳs, and Matú Jókay. A Note on Algebraic Properties of Quasigroups in Edon80. Technical report, eSTREAM report 2007/005, 2007.

[252] John Von Neumann. Various Techniques Used in Connection with Random Digits. *Applied Math Series*, 12:36–38, 1951.

[253] Chenyu Wang, Tao Huang, and Hongjun Wu. On the Weakness of Constant Blinding PRNG in Flash Player. In *ICICS 2018*, volume 11149 of *Lecture Notes in Computer Science*, pages 107–123. Springer, 2018.

[254] Greg Ward. A Recursive Implementation of the Perlin Noise Function. In *Graphics Gems II*, pages 396–401. Elsevier, 1991.

[255] Donald R Weidman. Emotional Perils of Mathematics. *Science*, 149(3688):1048–1048, 1965.

[256] Chuan-Kun Wu. Hash channels. *Computers & Security*, 24(8):653–661, 2005.

[257] Mark Wutka. The Crypto Forum, http://s13.zetaboards.com/Crypto/topic/123721/1/.

[258] Akihiro Yamaguchi, Takaaki Seo, and Keisuke Yoshikawa. On the Pass Rate of NIST Statistical Test Suite for Randomness. *JSIAM Letters*, 2:123–126, 2010.

[259] Song Y. Yan. *Number Theory for Computing.* Theoretical Computer Science. Springer, 2002.

[260] Andrew C. Yao. Protocols for Secure Computations. In *SFCS 1982*, pages 160–164. IEEE Computer Society, 1982.

[261] Adam Young and Moti Yung. The Dark Side of "Black-Box" Cryptography or: Should We Trust Capstone? In *CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 89–103. Springer, 1996.

[262] Adam Young and Moti Yung. Kleptography: Using Cryptography Against Cryptography. In *EUROCRYPT 1997*, volume 1233 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1997.

[263] Adam Young and Moti Yung. The Prevalence of Kleptographic Attacks on Discrete-Log Based Cryptosystems. In *CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 264–276. Springer, 1997.

[264] Adam Young and Moti Yung. *Malicious Cryptography: Exposing Cryptovirology.* John Wiley & Sons, 2004.

[265] Adam Young and Moti Yung. Malicious Cryptography: Kleptographic Aspects. In *CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 7–18. Springer, 2005.

[266] Dae Hyun Yum and Pil Joong Lee. Cracking Hill Ciphers with Goodness-of-Fit Statistics. *Cryptologia*, 33(4):335–342, 2009.

[267] Haina Zhang and Xiaoyun Wang. Cryptanalysis of Stream Cipher Grain Family. *IACR Cryptology ePrint Archive*, 2009/109, 2009.

[268] Yuliang Zheng. Digital Signcryption or How to Achieve Cost (Signature & Encryption) $\ll$ Cost (Signature)+ Cost (Encryption). In *CRYPTO 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 165–179. Springer, 1997.

[269] Yuliang Zheng and Hideki Imai. How to Construct Efficient Signcryption Schemes on Elliptic Curves. *Information Processing Letters*, 68(5):227–233, 1998.

[270] Yuliang Zheng and Jennifer Seberry. Immunizing Public Key Cryptosystems Against Chosen Ciphertext Attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):715–724, 1993.