HAL open science

Post-Quantum Signatures from Secure Multiparty Computation

Thibauld Feneuil

▶ To cite this version:

Thibauld Feneuil. Post-Quantum Signatures from Secure Multiparty Computation. Cryptography and Security [cs.CR]. Sorbonne Université, 2023. English. NNT : 2023SORUS324 . tel-04316885v2

HAL Id: tel-04316885
https://hal.science/tel-04316885v2

Submitted on 1 Dec 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

École doctorale Sciences Mathématiques de Paris Centre

Thèse de doctorat

Post-Quantum Signatures from Secure Multiparty Computation

Spécialité : Informatique

présentée et soutenue publiquement par

Thibauld Feneuil

le 23 octobre 2023

devant le jury composé de

Jean Claude Bajard	Directeur de thèse
Alain Couvreur	Rapporteur
Geoffroy Couteau	Examinateur
Antoine Joux	Co-directeur de thèse
Tanja Lange	Examinatrice
Emmanuela Orsini	Rapporteure
Matthieu Rivain	Co-encadrant de thèse
Nicolas Sendrier	Président du jury
Greg Zaverucha	Invité

Abstract

The ongoing effort to build a quantum computer urges the cryptography community to develop new secure cryptosystems based on quantum-hard cryptographic problems. In this thesis, we focus on the design of signature schemes built from zero-knowledge proofs of knowledge. More precisely, we focus on the MPC-in-the-Head paradigm which provides a generic way to build zero-knowledge proofs using techniques from secure multiparty computation.

We propose several new signature schemes using the MPC-in-the-Head framework. Most of these schemes are competitive with the existing schemes in the post-quantum literature. They have signature sizes between 5 KB and 20 KB for 128-bit security, and have very small public keys (less than 200 B). Their security relies on a large scope of hard problems. Some of them rely on code-based assumptions, such as the hardness of solving the syndrome decoding problem for random linear codes. Others rely on the multivariate quadratic problem, the subset sum problem, and the MinRank problem.

We also develop two new MPC-in-the-Head techniques. The first one aims to efficiently address a context of small secret values over large modulus. The second one consists of a new way of transforming an MPC protocol into a zero-knowledge proof. This new transformation provides new trade-offs in terms of communication costs vs running times. In particular, it enables us to achieve small verification times.

Several submissions in the NIST call for additional post-quantum signatures rely (sometimes partially) on ideas developed in this thesis.

Keywords: zero-knowledge proofs, post-quantum signatures, MPC-in-the-Head.

Résumé

Le développement actuel des ordinateurs quantiques pousse la communauté cryptographique à mettre au point de nouveaux cryptosystèmes dont la sécurité se fonde sur la difficulté à résoudre des problèmes cryptographiques résistant au calcul quantique. Dans le cadre de cette thèse, nous nous sommes focalisés sur la conception de schémas de signatures électroniques construits à partir de preuves à divulgation nulle de connaissance (*zero-knowledge proofs of knowledge* en anglais). Plus précisément, nous nous sommes intéressés au paradigme "MPC-in-the-Head" (littéralement, "calcul-multipartite-dans-la-tête" en français) qui fournit une méthode générique de construire de telles preuves en utilisant des techniques de calcul multipartite sécurisé.

Nous proposons plusieurs nouveaux schémas de signatures utilisant le paradigme "MPC-in-the-Head". La plupart d'entre eux sont compétitifs avec les schémas existants dans l'état de l'art post-quantique. Ils produisent des signatures ayant des tailles entre 5 et 20 kilo-octets (pour un niveau de sécurité de 128 bits) et possèdent de très petites clés (de moins de 200 octets). Les problèmes difficiles sur lesquels la sécurité de ces schémas se fonde sont très variés. Certains schémas s'appuient sur des hypothèses de sécurité issues de la théorie des codes correcteurs d'erreurs, telle que celle sur la difficulté à résoudre le problème de décodage par syndrome pour des codes linéaires aléatoires. Les autres schémas s'appuient sur la difficultés à résoudre un système d'équations quadratiques, le problème de la somme de sous-ensembles ou le problème MinRank.

Nous avons également mis au point deux nouvelles techniques de MPC-in-the-Head. La première vise à gérer efficacement les situations où le secret est de petite taille avec un grand modulus. La seconde consiste en une nouvelle méthode pour transformer un protocole de calcul multipartite en preuve de divulgation nulle de connaissance. Cette nouvelle transformation offre des nouveaux compromis entre coût de communication et temps de calcul. En particulier, elle permet de produire des algorithmes de vérification très rapides.

Plusieurs soumissions à l'appel du NIST pour des schémas de signatures post-quantiques supplémentaires s'appuient (parfois partiellement) sur des idées développées dans le cadre de cette thèse.

Mots-clés: preuves à divulgation nulle de connaissance, signatures post-quantiques, MPC-in-the-Head.

Remerciements – Acknowledgments

Ce travail a été réalisé dans le cadre d'une thèse CIFRE à CryptoExperts. En plus de mon activité de recherche, j'ai eu l'occasion de travailler sur des missions clients, ce qui m'a permis d'élargir le spectre de mes rencontres.

En tout premier lieu, je veux présenter mes remerciements les plus sincères à Jean Claude Bajard, Antoine Joux et Matthieu Rivain pour m'avoir offert l'opportunité de faire cette thèse en cryptographie. Je tiens à remercier tout particulièrement Matthieu qui m'a permis d'avoir un cadre idéal pour mon doctorat et a été un interlocuteur si attentionné. Je lui en serai indéfiniment reconnaissant. Merci à Antoine pour nos nombreuses discussions et ses conseils durant ces trois années. Merci enfin à Jean Claude pour son aide et sa réactivité, notamment pour les aspects administratifs de la thèse.

Je remercie de tout coeur mes collègues passés et présents de CryptoExperts pour ces trois superbes années: Sonia Belaïd, Ryad Benadjila, Nicolas Bon, Louis Goubin, Darius Mercadier, Viet-Sang Nguyen, Pascal Paillier, Matthieu Rivain, Abdul Rahman Taleb, Aleksei Udovenko et Junwei Wang. J'ai beaucoup apprécié de travailler avec eux... et j'ai adoré tous les moments de partage comme les Crytobeers et les team buildings.

I would like to strongly thank Geoffroy Couteau, Alain Couvreur, Tanja Lange, Emmanuela Orsini, Nicolas Sendrier, and Greg Zaverucha for doing me the honor of constituting my thesis jury. I also thank Alain and Emmanuela a lot for the time they took to review my manuscript and for their feedback.

Je remercie les membres de l'équipe ALMASTY[1] du LIP6 (Sorbonne Université) de m'avoir accueilli avec bienveillance, de m'avoir fait découvrir le milieu de la recherche académique et d'avoir partagé certains de leurs moments de convivialité avec moi. Merci donc à Ahmed Khulaif Alharbi, Samuel Bouaziz–Ermann, Charles Bouillaguet, Andersson Calle Viera, Orel Cosseron, Ambroise Fleury, Mickaël Hamdad, Jules Maire, Florette Martinez, Julia Sauvage, Abdul Rahman Taleb et Damien Vergnaud.

Je tiens à remercier toutes les personnes que j'ai rencontrées au cours des différents séminaires et conférences, et avec lesquelles j'ai eu beaucoup de plaisir à discuter. Je pense notamment à Maxime Bros, Thomas Debris-Alazard, Charles Meyer-Hilfiger et Pierre Briaud, mais cette liste n'est pas exhaustive et mes remerciements s'adressent bien sûr à tous ceux avec qui j'ai pu échanger au cours de ces trois années de thèse.

Je remercie Jean-Pierre Tillich pour l'organisation des GT Code-Based Cryptography à l'INRIA Paris. Ce groupe de travail m'a permis de rencontrer de nombreuses personnes de la communauté française de cryptographie avec lesquelles je n'aurais pas eu l'occasion de discuter autrement.

[1] L'équipe ALMASTY et ses "amis"

Je souhaite aussi remercier les membres du projet CRY.ME : Ryad Benadjila, Jérémy Jean, Louiza Khati, Ange Martinelli et Chrysanthi Mavromati, pour ne citer que ceux extérieurs[2] à CryptoExperts. La conception du challenge et son déroulement ont constitué une très belle aventure, à la fois formatrice et passionnante.

Je remercie également les membres des équipes de soumissions pour l'appel du NIST pour les signatures post-quantiques. Prendre part au processus de création de toutes ces soumissions a été très intéressant et formateur.

Pour finir, je remercie les membres de ma famille de m'avoir soutenu durant ces années, s'inquiétant parfois pour moi lors des périodes de travail particulièrement intenses mais respectant mes choix. Je remercie particulièrement mon petit frère Fergal pour les moments qu'il m'a offerts pour me sortir un peu du cadre professionnel. Merci aussi à Lumi d'avoir été un compagnon si solidaire.

[2]Pendant le projet

Contents

Abstract iii

Résumé v

Remerciements – Acknowledgments vii

1. Introduction 1
- 1.1. Cryptology . . . 1
- 1.2. Post-quantum cryptography . . . 2
- 1.3. Signature Schemes . . . 3
- 1.4. Contributions of this Thesis . . . 4

2. Technical Background 7
- 2.1. Notations . . . 8
- 2.2. Cryptographic Building Blocks . . . 8
  - 2.2.1. Hash Functions . . . 8
  - 2.2.2. Pseudo-Random Generator . . . 9
  - 2.2.3. Commitment Schemes . . . 10
- 2.3. Zero-Knowledge Proofs . . . 12
  - 2.3.1. Interactive Protocols . . . 12
  - 2.3.2. Proofs of Knowledge . . . 12
  - 2.3.3. Schwartz-Zippel Lemma and Variants . . . 14
- 2.4. Secure Multiparty Computation . . . 15
  - 2.4.1. Secret Sharing Schemes . . . 15
  - 2.4.2. Multiparty computation . . . 17
- 2.5. Signature Schemes . . . 18

3. The MPC-in-the-Head Paradigm 19
- 3.1. Generic Transformations . . . 20
  - 3.1.1. General Setting . . . 20
  - 3.1.2. Broadcast-based Multiparty Computation . . . 22
  - 3.1.3. Linear Broadcast-based Multiparty Computation . . . 25
- 3.2. Paradigm Usage . . . 27
- 3.3. Useful techniques . . . 27
  - 3.3.1. Protocols with Helper . . . 27
  - 3.3.2. Statistical checkings . . . 28
  - 3.3.3. MPCitH with Rejection . . . 29
- 3.4. State of the art . . . 29
  - 3.4.1. Zero-knowledge proofs of knowledge for arbitrary circuits . . . 30
  - 3.4.2. Zero-knowledge proofs of knowledge for dedicated circuits . . . 32

4. Shared Permutation for Syndrome Decoding **35**
4.1. Introduction . . . 36
4.2. A Zero-Knowledge Protocol for Syndrome Decoding . . . 37
4.2.1. General Idea . . . 37
4.2.2. Description of the Protocol . . . 39
4.2.3. Security Proofs . . . 39
4.2.4. Producing the Trusted Vector . . . 40
4.3. The Five-Round Zero-Knowledge Protocol . . . 43
4.3.1. Presentation of the Optimizations . . . 43
4.3.2. Description of the Protocol . . . 46
4.3.3. Security Proofs . . . 46
4.4. The Signature Scheme . . . 48
4.4.1. Transformation into a Non-Interactive Scheme . . . 48
4.4.2. Description of the Signature Scheme . . . 49
4.4.3. Security Proof . . . 51
4.5. Performance . . . 51
4.5.1. Communication Cost and Signature Size . . . 51
4.5.2. Choice of the SD Parameters . . . 52
4.5.3. Impact of the Other Parameters . . . 53
4.6. Comparison . . . 54
4.6.1. Comparison with Other Code-Based Signature Schemes . . . 54
4.6.2. Comparison with other Post-Quantum Signature Schemes . . . 56
4.7. Conclusion . . . 57

5. Syndrome Decoding in the Head **59**
5.1. Introduction . . . 60
5.1.1. The d-split Syndrome Decoding Problem . . . 60
5.2. A Zero-Knowledge Protocol for Syndrome Decoding . . . 62
5.2.1. Standard Case ($d = 1$) . . . 63
5.2.2. General case (any d) . . . 66
5.2.3. Description of the Protocol . . . 66
5.2.4. Security Proofs . . . 67
5.2.5. Performance . . . 67
5.2.6. Comparison . . . 69
5.3. The Signature Scheme . . . 71
5.3.1. Transformation into a Non-Interactive Scheme . . . 71
5.3.2. Description of the Signature Scheme . . . 71
5.3.3. Signature Properties . . . 72
5.3.4. Parameters . . . 72
5.3.5. Implementation and Performance . . . 75
5.4. Comparison . . . 77
5.4.1. Comparison with Other Code-Based Signature Schemes . . . 77
5.4.2. Comparison with other Post-Quantum Signature Schemes . . . 78
5.5. Conclusion . . . 79

6. MPC-in-the-Head with Rejection **81**
6.1. Introduction . . . 82
6.1.1. Prior Work . . . 82
6.1.2. Contributions . . . 83
6.1.3. Preliminaries . . . 84
6.2. General Idea . . . 84
6.2.1. The Naive Approach . . . 85
6.2.2. Sharing on the Integers and Opening with Abort . . . 85
6.2.3. Binarity Proof from Batch Product Verification . . . 87
6.2.4. Binarity Proof from Masking and Cut-and-Choose Strategy . . . 90
6.2.5. Asymptotic Analysis . . . 91
6.3. Protocols and Security Proofs . . . 92
6.3.1. Protocol with Batch Product Verification . . . 92
6.3.2. Protocol with Cut-and-Choose Strategy . . . 94
6.3.3. Decreasing the Rejection Rate . . . 96
6.4. Instantiations and Performance . . . 97
6.4.1. Subset Sum Instances . . . 97
6.4.2. Zero Knowledge Protocols . . . 98
6.4.3. Signature Schemes with Subset Sum Problem . . . 99
6.5. Digital Signatures from Boneh-Halevi-Howgrave-Graham PRF . . . 100
6.6. Conclusion . . . 105

7. Building Signatures from MQ, MinRank and Rank SD **107**
7.1. Introduction . . . 108
7.2. Methodology . . . 109
7.2.1. Matrix Multiplication Checking Protocol . . . 111
7.2.2. MPCitH Optimizations . . . 111
7.3. Signature Scheme from $\mathcal{MQ}$ . . . 112
7.4. Signature Scheme from MinRank and Rank SD . . . 116
7.4.1. Matrix Rank Checking Protocols . . . 116
7.4.2. Signature Scheme from MinRank . . . 118
7.4.3. Signature Scheme from Rank SD . . . 122
7.5. Running times . . . 126
7.6. Conclusion . . . 128

8. MPC-in-the-Head with Threshold Linear Secret Sharing **129**
8.1. Introduction . . . 130
8.2. Formalizing the MPCitH-Friendly MPC Protocols . . . 133
8.2.1. General Model of MPC Protocol . . . 133
8.2.2. Application of the MPCitH Principle . . . 137
8.3. MPC-in-the-Head with Threshold LSS . . . 139
8.3.1. General Principle . . . 139
8.3.2. Conversion to Zero-Knowledge Proofs . . . 141
8.3.3. Soundness . . . 142
8.3.4. Performance . . . 146
8.4. Further Improvements . . . 147
8.4.1. Using Threshold Ramp Linear Secret Sharing . . . 148

8.4.2. Batching Proofs with Shamir's Secret Sharing . . . 150
8.5. Applications . . . 152
8.5.1. Application to the SDitH Signature Scheme . . . 153
8.5.2. Application of the Batching Strategy . . . 157
8.6. Conclusion . . . 163

9. From Research to Specification **165**
9.1. NIST Call for Additional Signatures . . . 166
9.1.1. SD-in-the-Head Signature Scheme . . . 167
9.1.2. MIRA Signature Scheme . . . 168
9.1.3. RYDE Signature Scheme . . . 168
9.1.4. MQOM Signature Scheme . . . 169
9.2. MPC-in-the-Head Library . . . 169
9.2.1. Introduction . . . 169
9.2.2. Structure and configuration . . . 170
9.2.3. Some benchmarks . . . 172

10. Conclusion and Open Questions **175**

Appendices **177**
A. Proof of Privacy . . . 177
B. Proof of Soundness . . . 178
B.1. Technical Lemmas . . . 179
B.2. When restraining to only bad witnesses . . . 180
B.3. Building of the extractor . . . 183
C. Signature Scheme and Proof of Unforgeability . . . 187

Bibliography **199**

Chapter 1.

Introduction

1.1. Cryptology

How would you send a love declaration to someone who is far away while preventing curious people from reading its content? How could you have the guarantee that nobody will change it? And for the receiver, how can she/he be sure that the message is really sent by the right person? Unfortunately, we are clearly not living in a world where everyone can be trusted. If you want to declare your love without being troubled, two ways can be considered: either you make the journey to meet the (wo)man of your heart, or you use the *cryptography*. The latter provides many technological solutions to build secure communication over an insecure channel (e.g. phone, internet, ...) with the following features:

- *confidentiality*: the message is unintelligible for anyone except the receiver (and the sender),
- *message authentication*: the receiver can check the identity of the sender of the message,
- *data integrity*: the receiver can verify that the message has not been modified during its transfer.

Cryptography consists in the conception of sets of algorithms, usually named *cryptosystems*, which provide one or several of the above security features. On the other side, the *cryptanalysis* consists in finding ways to break the security properties of those cryptosystems. Together they form the "science of secrecy", named *cryptology*.

There exist two types of cryptosystems:

- Symmetric cryptography: in that setting, the sender and the receiver know a common secret key (it can be a number, a passphrase, ...). The sender will *encrypt* the message using this key, and the receiver will also use it to put back the message in an intelligible form.
- The asymmetric cryptography (or public-key cryptography): in that setting, the receiver produces two keys, a public key and a secret key. She sends the first one to the sender. Using it, the sender encrypts the message. Then using the private key, the receiver decrypts it.

The symmetric setting has the drawback of requiring that both parties have a common secret. For example, if you want to securely browse the website of a bank, you do not have a common

secret to establish secure communication. Asymmetric cryptography addresses this issue. The bank website can send you a public key and you can use it to encrypt the requests for the website. However, the asymmetric schemes are quite heavy in terms of bandwidth and computation. So, they will be used *only* to derive a common secret key. Then, using the latter, the user and the bank website can securely communicate using symmetric schemes.

1.2. Post-quantum cryptography

The security of the current public-key cryptosystems (for example, on the Internet) relies on the hardness of factorizing a composite number

> "Given a number n, one needs to find non-trivial p and q such that $n = p \cdot q$."

and to solve the discrete logarithm problem

> "Given a modulus n and two non-zero elements y and g, one needs to find x such that $y = g^x \mod n$."

(or its variant on elliptic curves). However, Shor [Sho94] showed that a quantum computer would be able to solve "easily" those two problems. It means that such a computer would break all the current secure communications using asymmetric schemes (*i.e.* it would decrypt the messages without the keys, impersonify people, etc...). Hopefully, this technology is not functional yet, but it is improving constantly. Quantum computers may exist in a dozen years[1].

Because of this threat, the research community and standardization organizations are searching for new cryptosystems that would be secure against quantum technology. These schemes compose what is called the *post-quantum cryptography* (also known as the quantum-safe/quantum-resilient cryptography). The goal is to replace the vulnerable algorithms with post-quantum secure ones. In 2016, the National Institute of Standards and Technology (NIST) opened a call for proposals for the future standards of post-quantum cryptosystems [NIS16]. In November 2017 (call deadline), 59 key encapsulation mechanisms (KEM) and 23 signature schemes have been submitted. We are currently in the 4th round of the standardization process. Most of the schemes have been eliminated. In July 2022, the NIST announced that 4 candidates have been selected for standardization: the key encapsulation mechanism Kyber [ABD+21], and the signature schemes Dilithium [BDK+21a], Falcon [FHK+20] and SPHINCS$^+$ [ABB+22]. In the 4th round, there are 4 KEM but no signature schemes.

Let us discuss the case of the signature schemes. There are no more signatures in the selection process of the NIST. Three schemes will be standardized. The security of two of them (Dilihium and Falcon) relies on the hardness of solving structured lattice problems. The security of the last scheme (SPHINCS$^+$) relies on the security of cryptographic hash functions. One could say that *lattice-based cryptography* (the schemes for which the security relies on the hardness of solving the lattice problems) is the true winner of the NIST post-quantum standardization process. Those schemes have the best global performance. However, we should avoid putting all our eggs in one basket. In case the cryptanalysis of the lattice-based schemes would improve (for example, one finds a polynomial quantum attack against them),

[1] Technically, some quantum computers already exist, but are not powerful enough to be a threat to the current cryptography.

we should have backup solutions. For this reason, the NIST also selected SPHINCS$^+$. The latter is very conservative, however it is a heavy scheme: it has quite a large signature size and a long signing time. Since there is no more scheme in the standardization process, the NIST chose to re-open a call for additional proposals [NIS22], but only for the signature schemes. In June 2023 (the deadline for the new call), 50 signature schemes have been submitted. Among them, 40 have been kept for the first round of the new call.

1.3. Signature Schemes

In this thesis, we focus on post-quantum signature schemes. The goal of a (digital) signature scheme is to *verify the authenticity of messages or documents*. A valid digital signature on a message ensures that the message comes from a sender known to the recipient. The principle is the following. The signer produces two keys: a public key pk and a secret key sk. Using the secret key, she can sign a message m, *i.e.* she can generate a signature σ. She will broadcast pk to everyone. Then, using this public key, everyone can check that σ is a valid signature for the message m. The desired security property for a signature scheme is the *existential unforgeability*: nobody should be able to produce a valid signature for a message (which has not been already signed) for a public key without knowing the corresponding secret key[2].

There exist two approaches to build a signature scheme. The first one is the *hash-and-sign paradigm*. To use it, one needs to have a function F which is computationally impossible to invert unless one knows a trapdoor. To sign a message m, the signer needs to hash the message

$$h \leftarrow \mathrm{Hash}(m)$$

and then she uses her trapdoor (which corresponds to the secret key) to compute a pre-image of h by the function F:

$$\sigma \leftarrow F^{-1}(h).$$

This pre-image σ corresponds to the signature of m. Then to verify σ, one just needs to compute the image of σ by F and check that it equals the hash digest of the message:

$$F(\sigma) \stackrel{?}{=} \mathrm{Hash}(m).$$

Forging a valid signature without knowing the secret key consists in finding a pre-image of F, which is computationally impossible by the definition of F. This technique leads to *short signature sizes*, but since it relies on a trapdoor, the underlying problem must have some structure and it is hence more sensitive to structural attacks. Moreover, the public key consists of the description of a function with a trapdoor (*i.e.* the function F), and thus, it is often large.

The second approach to build signature schemes uses zero-knowledge proofs and the Fiat-Shamir transformation. A *zero-knowledge proof of knowledge* is an interactive protocol enabling a prover to convince the verifier that she knows a secret satisfying some properties without revealing any information about the secret itself. For example, let us assume that the prover wants to convince the verifier that she knows the solution to a system of quadratic equations. One naive method for the prover would be to reveal this solution, but in the end, the verifier will also know the solution. Thus, a zero-knowledge proof of knowledge consists

[2] And of course, nobody should be able to find the secret key from the public key and the produced signatures.

of a discussion between the prover and the verifier such that, at the end of this interaction, the verifier will be convinced that the prover really knows such a solution without getting any clue about the solution itself. There are two security properties for such protocols:

- **Soundness:** if the prover does not know the secret she is claiming to know, then the verifier should not be convinced at the end of the discussion,
- **Zero-Knowledge:** if the prover really knows the secret, the discussion should not leak any information (even partial) about the secret itself.

A signature could be considered as a proof that the signer is the holder of the secret key which corresponds to the considered public key. However, between a signature scheme and a zero-knowledge proof of knowledge of a secret key (which corresponds to an *identification scheme*), there are two differences:

1. the proof of knowledge is a discussion (*i.e.* it is interactive), while a signature scheme should not require interaction between the signer and possible verifiers;
2. the signature involves a message, not the proof of knowledge.

Hopefully, there exists the Fiat-Shamir transformation: it enables us to convert a proof of knowledge into a signature scheme by removing the interactions of the proof system while binding each proof to a message. Signature schemes built from a zero-knowledge proof tend to lead to larger signatures (compared to the hash-and-sign approach). They do not rely on a trapdoor, so their public keys are often very small and their security is often considered conservative.

1.4. Contributions of this Thesis

In this thesis, we study how we can improve the state of the art of post-quantum signature schemes built from zero-knowledge proofs. More precisely, we focus on one promising framework of zero-knowledge proofs: the MPC-in-the-Head paradigm. The latter (presented in Chapter 3) provides a generic method to build proof systems using techniques from secure multiparty computation (MPC).

First of all, in Chapter 4 and in Chapter 5, we propose two new code-based signature schemes relying on the hardness of the syndrome decoding problem for random linear codes (in Hamming metric), one of the oldest problems of code-based cryptography. These schemes outperform the former schemes based on the same assumptions and are highly competitive with other code-based signature schemes.

Then, in Chapter 6, we describe a new MPC-in-the-Head technique to efficiently handle the case where we should manipulate small secret values over a large modulus. We use this technique to propose two signature schemes: one relying on the hardness of the subset sum problem, the other one relying on the security of the BHH pseudo-random function [BHH01]. While improving drastically the schemes based on the same assumption, the first proposal is unfortunately not competitive with the post-quantum literature. The performance of the second proposal is highly competitive, but its security is much less mature (regarding the state of quantum cryptanalysis).

In Chapter 7, we apply a systematic methodology to build signature schemes using the MPC-in-the-Head paradigm. We propose several schemes relying on the multivariate quadratic

problem, the MinRank problem, the rank syndrome decoding problem and the permuted kernel problem. For most of them, we improve the existing literature.

In Chapter 8, we propose a new MPC-in-the-Head technique to transform an MPC protocol into a zero-knowledge proof. Instead of relying on additive sharing as all the existing MPCitH-based signature schemes, we show how using a low-threshold linear secret sharing can provide interesting trade-offs in terms of communication costs vs running times. While slightly increasing the signature size, it drastically improves the running times, especially for the verifier.

Finally, in Chapter 9, we describe the process we follow to transform theoretical schemes described in research articles into practical implementations. We also briefly present some signature schemes submitted to the recent NIST call for additional post-quantum signatures [NIS22] designed in the context of this thesis.

In this manuscript, most of the theorems deal with the security properties of the proposed zero-knowledge proofs of knowledge and the signature schemes. Formally proving such theorems is verbose, with proofs taking many pages. Moreover, the contributions of this thesis are minor in these proofs since the latter correspond to standard proof techniques. For example, proving the security of a signature scheme consists of game transitions which are quite the same between all the MPCitH-based schemes. We thus refer the interested reader to the corresponding articles for the security proofs. In this manuscript, we only provide the security proofs of the theorems of Chapter 8 since these proofs differ from what we can find in other articles. However, because of the verbosity of the proofs, we provide them in appendices and we give the proof intuition in the body of the manuscript.

In what follows, you can find the articles (published or not yet) and the NIST submissions related to this thesis.

Publications in peer-reviewed venues:

- [FJR23] T. Feneuil, A. Joux, and M. Rivain. Shared Permutation for Syndrome Decoding: New Zero-Knowledge Protocol and Code-based Signature. Des. Codes Cryptogr. 91, 563-608 (2023). Content of this work appears in Chapter 4.

- [FJR22b] T. Feneuil, A. Joux, and M. Rivain. Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs. Published in the proceedings of Crypto 2022. Content of this work appears in Chapter 5.

- [FMRV22b] T. Feneuil, J. Maire, M. Rivain, and D. Vergnaud. Zero-Knowledge Protocols for the Subset Sum Problem from MPC-in-the-Head with Rejection. Published in the proceedings of Asiacrypt 2022. Content of this work appears in Chapter 6.

- [Fen22] T. Feneuil. Building MPCitH-based Signatures from MQ, MinRank, Rank SD and PKP. To appear in the proceedings of the 22nd International Conference on Applied Cryptography and Network Security, ACNS 2024. Content of this work appears in Chapter 7.

- [FR22] T. Feneuil, and M. Rivain. Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head. To appear in the proceedings of Asiacrypt 2023. Content of this work appears in Chapter 8.

Submissions to the NIST call for additional post-quantum signatures [NIS22]:

- C. Aguilar Melchor, T. Feneuil, N. Gama, S. Gueron, J. Howe, D. Joseph, A. Joux, E. Persichetti, T. Randrianarisoa, M. Rivain, and D. Yue. *The Syndrome Decoding in the Head (SD-in-the-Head) Signature Scheme.*
- N. Aragon, M. Bardet, L. Bidoux, J.-J. Chi-Domínguez, V. Dyseryn, T. Feneuil, P. Gaborit, R. Neveu, M. Rivain, and J.-P. Tillich. *MIRA (MIn RAnk) Signature Scheme*
- N. Aragon, M. Bardet, L. Bidoux, J.-J. Chi-Domínguez, V. Dyseryn, T. Feneuil, P. Gaborit, A. Joux, M. Rivain, J.-P. Tillich, and A. Vinçotte. *RYDE (Rank sYndrome DEcoding) Signature Scheme*
- T. Feneuil and M. Rivain. *MQOM: MQ on my Mind.*

Chapter 2.

Technical Background

In this chapter, we will introduce all the technical background required to understand the following chapters. After introducing some notations and vocabulary, we will present some useful standard cryptographic primitives such as the hash functions and the pseudo-random generators. Then we will explain the notions of the zero-knowledge proofs of knowledge and of the multiparty computation, which are fundamental to understand the MPC-in-the-Head paradigm. To finish, we will define the notion of signature schemes and present the Fiat-Shamir transformation.

Contents

2.1. Notations 8
2.2. Cryptographic Building Blocks 8
2.3. Zero-Knowledge Proofs 12
2.4. Secure Multiparty Computation 15
2.5. Signature Schemes 18

2.1. Notations

Throughout the manuscript, $\mathbb{F}$ shall denote a finite field. We denote $\mathbb{F}_q$ the field of order q. For any $m \in \mathbb{N}^*$, the integer set $\{1, \ldots, m\}$ is denoted $[1 : m]$. For a probability distribution D, the notation $s \leftarrow D$ means that s is sampled from D. For a finite set S, the notation $s \leftarrow S$ means that s is uniformly sampled at random from S. For an algorithm $\mathcal{A}$, $out \leftarrow \mathcal{A}(in)$ further means that out is obtained by a call to $\mathcal{A}$ on input in (using uniform random coins whenever $\mathcal{A}$ is probabilistic). Along the paper, probabilistic polynomial time is abbreviated PPT.

All the vectors are written as *column vectors*. For any matrix $H \in \mathbb{F}^{m \times n}$, the kernel of H denoted $\mathrm{Ker}(H)$, is the set of solutions to the equations $Hx = 0$. For any vector $x \in \mathbb{F}^n$, the *Hamming weight* of x, denoted $\mathrm{wt}_H(x)$, is the number of non-zero coordinates of x.

A function $\mu : \mathbb{N} \to \mathbb{R}$ is said *negligible* if, for every positive polynomial $p(\cdot)$, there exists an integer $N_p > 0$ such that for every $\lambda > N_p$, we have $|\mu(\lambda)| < 1/p(\lambda)$. When not made explicit, a negligible function in λ is denoted $\mathsf{negl}(\lambda)$ while a polynomial function in λ is denoted $\mathsf{poly}(\lambda)$. We further use the notation $\mathsf{poly}(\lambda_1, \lambda_2, ...)$ for a polynomial function in several variables.

Two distributions $\{D_\lambda\}_\lambda$ and $\{E_\lambda\}_\lambda$ indexed by a security parameter λ are (t, ε)-*indistinguishable* (where t and ε are $\mathbb{N} \to \mathbb{R}$ functions) if, for any algorithm $\mathcal{A}$ running in time at most $t(\lambda)$ we have

$$\left| \Pr[\mathcal{A}^{D_\lambda}() = 1] - \Pr[\mathcal{A}^{E_\lambda}() = 1] \right| \leq \varepsilon(\lambda) \ ,$$

with $\mathcal{A}^{Dist}$ meaning that $\mathcal{A}$ has access to a sampling oracle of distribution $Dist$. The two distributions are said

- *computationally indistinguishable* if $\varepsilon \in \mathsf{negl}(\lambda)$ for every $t \in \mathsf{poly}(\lambda)$;
- *statistically indistinguishable* if $\varepsilon \in \mathsf{negl}(\lambda)$ for every (unbounded) t;
- *perfectly indistinguishable* if $\varepsilon = 0$ for every (unbounded) t.

2.2. Cryptographic Building Blocks

2.2.1. Hash Functions

A hash function is an efficient function that maps data of arbitrary size to a fixed-size bitstring. A cryptographic hash function h must be resistant to collisions, *i.e.* it should be very hard to find x and x' such that $h(x) = h(x')$.

Definition 2.2.1 (Collision-Resistant Hash Functions)**.** *A family of functions* $\{\mathrm{Hash}_k : \{0,1\}^* \to \{0,1\}^{\ell(\lambda)} \ ; \ k \in \{0,1\}^{\kappa(\lambda)}\}_\lambda$ *indexed by a security parameter* λ *is collision-resistant, for some* $\ell, \kappa : \mathbb{N} \to \mathbb{N}$*, if there exists a negligible function* ν *such that, for any PPT algorithm* $\mathcal{A}$*, we have*

$$\Pr\left[\begin{array}{c} x \neq x' \\ \cap \ \mathrm{Hash}_k(x) = \mathrm{Hash}_k(x') \end{array} \middle| \begin{array}{c} k \leftarrow \{0,1\}^{\kappa(\lambda)}; \\ (x, x') \leftarrow \mathcal{A}(k) \end{array} \right] \leq \nu(\lambda) \ .$$

Cryptographic hash functions are central in cryptography. They can be used to build other cryptographic primitives such as message authentication codes (MAC), block ciphers, pseudo-random generators, commitment schemes, ...

2.2.2. Pseudo-Random Generator

A pseudo-random generator (PRG) is a deterministic algorithm that takes an initial value named *seed* and produces a bitstring that looks like a random string. Formally, it is defined as below.

Definition 2.2.2 (Pseudorandom Generator (PRG))**.** *Let $G : \{0,1\}^* \to \{0,1\}^*$ and let $\ell(\cdot)$ be a polynomial such that for any input $s \in \{0,1\}^\lambda$ we have $G(s) \in \{0,1\}^{\ell(\lambda)}$. Then, G is a (t,ϵ)-secure pseudorandom generator if the following two conditions hold:*

- *Expansion: $\ell(\lambda) > \lambda$;*
- *Pseudorandomness: the distributions*

$$\{G(s) \mid s \leftarrow \{0,1\}^\lambda\} \quad \textit{and} \quad \{r \mid r \leftarrow \{0,1\}^{\ell(\lambda)}\}$$

  are computationally (t,ε)-indistinguishable.

The first condition simply states that the algorithm must produce more bits than those received as input. The second condition states that there exists no efficient algorithm which can distinguish the PRG output from true randomness, as soon as we choose the seed randomly.

This primitive is very useful in cryptography. One reason is that producing true randomness is very expensive (since we need to rely on physical phenomena). If we need to have a large quantity of randomness, it can be an issue. A solution would be to produce a seed using true randomness and to rely on a secure PRG initialized with this seed to get the desired data. Another reason could be to save communication. In some cryptosystems, we need to send large random data. If we generate the latter thanks to a pseudo-random generator, we just need to send the used seed.

Another interesting primitive we will use in the following chapters is puncturable pseudo-random functions (puncturable PRF).

Definition 2.2.3 (Puncturable PRF)**.** *A puncturable PRF family F on the set $[1:N]$ is a set of functions*

$$\{F_k : [1:N] \to \{0,1\}^*\}_k$$

indexed by a key k such that

- *for all k and index i, there exists a punctured key k_i^* together with an efficient evaluation algorithm $\mathcal{A}$ such that*

$$\forall j \in [1:N]\backslash\{i\},\ \mathcal{A}(k_i^*, j) = F_k(j);$$

- *given the punctured key k_i^*, the value $F_k(i)$ should remain indistinguishable from a random value.*

Intuitively, such a primitive produces N pseudo-random values and provides a way to reveal all the values but one. The naive way to proceed would be to reveal the $N-1$ values directly, but it could lead to large communication. The challenge when building such primitives is to minimize the communication needed to reveal those $N-1$ values.

A standard construction of puncturable PRFs can be derived from the tree-based construction of [GGM86], usually named the GGM construction. The idea is to use a *tree PRG* in which one uses a pseudorandom generator to expand a root seed mseed into N subseeds in a structured way. The principle is to label the root of a binary tree of depth $\lceil \log_2 N \rceil$ with mseed. Then, one inductively labels the children of each node with the output of a standard PRG applied to the node's label. The subseeds $(\mathsf{seed}_i)_{i \in [N]}$ are defined as the labels of the N leaves of the tree. To reveal $N-1$ subseeds, one reveals the siblings of all nodes in the path from the punctured seed to the tree root, it will be the punctured key. Thus, the communication cost scales with $\lceil \log_2(N) \rceil$, and not $N-1$. We can generalize this process to reveal all the subseeds but a small subset $E \subset [1:N]$ by only revealing at most $|E| \cdot \log(N/|E|)$ labels of the tree. The idea is to reveal the labels on the siblings of the paths from the root of the tree to leaves $i \in E$ (excluding the labels of those paths themselves). Those labels allow someone to reconstruct $(\mathsf{seed}_i)_{i \notin E}$ while still hiding $(\mathsf{seed}_i)_{i \in E}$.

2.2.3. Commitment Schemes

We now formally introduce the notion of commitment scheme which is instrumental in many zero-knowledge protocols. Such schemes allow one to commit a chosen value while keeping it hidden to other people, with the ability to reveal the committed value later. Informally, they can be seen as the digital equivalent of a sealed envelope: whenever someone wants to commit to a message m, she puts m in the envelope. At a later moment, she can open the envelope to publicly reveal the message she committed to.

Definition 2.2.4 (Commitment Scheme)**.** *A commitment scheme is a triplet of algorithms* $(\mathsf{KeyGen}, \mathsf{Com}, \mathsf{Verif})$ *such that*

- KeyGen *is a PPT algorithm that, on input* 1^λ*, outputs some public parameters* $\mathsf{pp} \in \{0,1\}^{\mathsf{poly}(\lambda)}$ *containing a definition of the message space, the randomness space and the commitment space.*

- Com *is a deterministic polynomial-time algorithm that, on input the public parameters* pp*, a message* x *and the randomness* ρ*, outputs a commitment* c*.*

- Verif *is a deterministic polynomial-time algorithm that, on input the public parameters* pp*, a message* x*, a commitment* c *and the randomness* ρ*, outputs a bit* $b \in \{0,1\}$*.*

It should verify the following properties:

- ***Correctness:*** *for any message* x *and any randomness* ρ*:*

$$\Pr[\mathsf{Verif}_{\mathsf{pp}}(x, c, \rho) = 1 \mid c \leftarrow \mathsf{Com}_{\mathsf{pp}}(x; \rho)] = 1 \ .$$

- ***Hiding:*** *for any two messages* x_0 *and* x_1*, the following distributions*

$$\{c \mid c \leftarrow \mathsf{Com}_{\mathsf{pp}}(x_0; \rho), \rho \text{ random}\} \text{ and } \{c \mid c \leftarrow \mathsf{Com}_{\mathsf{pp}}(x_1; \rho), \rho \text{ random}\}$$

  are indistinguishable.

- ***Binding:*** *there exists a negligible function ν such that, for every (PPT) algorithm $\mathcal{A}$, we have*

$$\Pr\left[\begin{array}{c} x \neq x' \\ \cap\ \mathsf{Verif}_{\mathsf{pp}}(x, c, \rho) = 1 \\ \cap\ \mathsf{Verif}_{\mathsf{pp}}(x', c, \rho') = 1 \end{array} \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{KeyGen}(); \\ (x, x', \rho, \rho', c) \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] \leq \nu(\lambda)\ ,$$

  where the probability is taken over the randomness of $\mathcal{A}$ and KeyGen*. If we restrict $\mathcal{A}$ to being PPT, then the scheme is computationally binding. If the computation time of $\mathcal{A}$ is unbounded, then the scheme is statistically binding.*

In this manuscript, the public parameter input pp will be made implicit in the calls to Com and Verif.

Remark 2.2.5. *We can build a commitment scheme from a cryptographic hash function h. To commit a value v, we sample a random string ρ and we compute*

$$c \leftarrow h(v \parallel \rho).$$

To open the commitment c, we just need to reveal v and ρ, and the verifier can check that c has been well-built. The binding property of the obtained scheme comes from the collision resistance of the hash function while the hiding property comes from the randomness ρ and the pre-image resistance. We denote this scheme as the hash-based commitment scheme*. In some settings in which we need a weaker hiding property, when the value v has a large enough entropy, we can omit the randomness ρ.*

A *vector commitment scheme* is a commitment scheme enabling to commit a vector of values $(v_1, \ldots, v_n)$. Then, instead of revealing all those values, one can decide to reveal a *subset of them*. A naive method would consist in committing all these values separately:

$$c_i \leftarrow Commit(v_i; \rho_i),$$

but sending $(c_1, \ldots, c_n)$ could be too expensive. We can use a collision-resistant hash function to have a more efficient solution thanks to the *Merkle trees* (also known as *hash trees*) [Mer88]. A Merkle tree is a binary tree in which every leaf node is labelled with the commitment digest of the data block v_i, and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes. Given a collision-resistant hash function $\mathrm{Hash}(\cdot)$, the Merkle hash root for $N = 2^n$ input data blocks $v_1, \ldots, v_N$, denoted $\mathrm{MerkleTree}(v_1, \ldots, v_N)$, is hence defined as

$$\mathrm{MerkleTree}(v_1, \ldots, v_N) = \begin{cases} \mathrm{Hash}\left(\mathrm{MerkleTree}(v_1, \ldots, v_{N/2}) \parallel \mathrm{MerkleTree}(v_{N/2+1}, \ldots, v_N)\right) & \text{if } N > 1 \\ \mathsf{Com}(v_1; \rho) & \text{if } N = 1 \end{cases}$$

A Merkle tree makes it possible to show the consistency of a small subset $E \subset [N]$ of revealed inputs $(v_i)_{i \in E}$ with the hash root $h = \mathrm{MerkleTree}(v_1, \ldots, v_N)$ without having to communicate all the other inputs $(v_i)_{i \notin E}$ (or their corresponding hash). The principle is to reveal the sibling paths of $(v_i)_{i \in E}$ in the Merkle tree, that we shall denote $\mathsf{auth}((v_1, \ldots, v_N), E)$, and which contains at most $|E| \cdot \log(N/|E|)$ hash values.

2.3. Zero-Knowledge Proofs

2.3.1. Interactive Protocols

A two-party protocol is a triplet $\Pi = (\text{Init}, \mathcal{A}, \mathcal{B})$ where Init is an initialization algorithm that, on input 1^λ, produces a pair $(in_\mathcal{A}, in_\mathcal{B})$, and where $\mathcal{A}$ and $\mathcal{B}$ are two stateful algorithms, called the *parties*. The parties originally receive their inputs $in_\mathcal{A}$ and $in_\mathcal{B}$ then interact by exchanging messages, and finally one of the parties, say $\mathcal{B}$, produces the output of the protocol. More formally, an execution of the protocol consists of a sequence:

$$\begin{aligned}
&\mathsf{state}_\mathcal{A} \leftarrow \mathcal{A}(in_\mathcal{A}) \\
&\mathsf{state}_\mathcal{B} \leftarrow \mathcal{B}(in_\mathcal{B}) \\
&(\textsc{Msg}_\mathcal{A}[0], \mathsf{state}_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{state}_\mathcal{A}) \\
&\quad\vdots \\
&(\textsc{Msg}_\mathcal{B}[i], \mathsf{state}_\mathcal{B}) \leftarrow \mathcal{B}(\mathsf{state}_\mathcal{B}, \textsc{Msg}_\mathcal{A}[i-1]) \\
&(\textsc{Msg}_\mathcal{A}[i], \mathsf{state}_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{state}_\mathcal{A}, \textsc{Msg}_\mathcal{B}[i]) \\
&\quad\vdots \\
&out \leftarrow \mathcal{B}(\mathsf{state}_\mathcal{B}, \textsc{Msg}_\mathcal{A}[n])
\end{aligned}$$

The sequence of exchanged messages is called the *transcript* of the execution, which is denoted

$$\text{View}(\langle \mathcal{A}(in_\mathcal{A}), \mathcal{B}(in_\mathcal{B}) \rangle) := (\textsc{Msg}_\mathcal{A}[0], \textsc{Msg}_\mathcal{B}[1], \ldots, \textsc{Msg}_\mathcal{A}[n]) \ .$$

An execution producing an output *out* is further denoted

$$\langle \mathcal{A}(in_\mathcal{A}), \mathcal{B}(in_\mathcal{B}) \rangle \rightarrow out \ .$$

In our exposition, the state of the parties shall be made implicit. We shall then say that an algorithm has *rewindable black-box access* to a party $\mathcal{A}$ if this algorithm can copy the state of $\mathcal{A}$ at any moment, relaunch $\mathcal{A}$ from a previously copied state, and query $\mathcal{A}$ (with its current state) on input messages. A variable x is said to be *extractable* from $\mathcal{A}$ if there exists a PPT algorithm $\mathcal{E}$ which, given a rewindable black-box access to $\mathcal{A}$, returns x after a polynomial number of queries to $\mathcal{A}$.

2.3.2. Proofs of Knowledge

We will focus on a special kind of two-party protocol called an *interactive proof* which involves a *prover* $\mathcal{P}$ and a *verifier* $\mathcal{V}$. In such a protocol, $\mathcal{P}$ tries to prove a statement to $\mathcal{V}$. The first message sent by $\mathcal{P}$ is called a *commitment*, denoted $\textsc{Com}$. From this commitment $\mathcal{V}$ produces a first *challenge* $\textsc{Ch}_1$ to which $\mathcal{P}$ answers with a response $\textsc{Rsp}_1$, followed by a next challenge $\textsc{Ch}_2$ from $\mathcal{V}$, and so on. After receiving the last response $\textsc{Rsp}_n$, $\mathcal{V}$ produces a binary output: either $\textsc{Accept}$, meaning that she was convinced by $\mathcal{P}$, or $\textsc{Reject}$ otherwise. Such an m-round interactive proof with $m = 2n + 1$ (1 commitment + n challenge-response pairs) is illustrated on Protocol 1.

$\underline{\mathcal{P}}$ $in_{\mathcal{P}}$		$\underline{\mathcal{V}}$ $in_{\mathcal{V}}$
[...]	$\xrightarrow{\textsc{Com}}$	
	$\xleftarrow{\textsc{Ch}_1}$	
	$\xrightarrow{\textsc{Rsp}_1}$	
	$\vdots$	
	$\xleftarrow{\textsc{Ch}_n}$	
	$\xrightarrow{\textsc{Rsp}_n}$	
		Return $out \in \{\textsc{Accept}, \textsc{Reject}\}$

Protocol 1: Structure of a m-round interactive proof with $m = 2n + 1$.

The sequence of exchanged messages is called the *transcript* of the execution, which is denoted

$$\mathrm{View}(\langle \mathcal{P}(in_{\mathcal{P}}), \mathcal{V}(in_{\mathcal{V}})\rangle) := (\textsc{Com}, \textsc{Ch}_1, \textsc{Rsp}_1, \ldots, \textsc{Ch}_n, \textsc{Rsp}_n)$$

where $in_{\mathcal{P}}$ and $in_{\mathcal{V}}$ respectively denote the prover and verifier inputs. An execution producing an output $out \in \{\textsc{Accept}, \textsc{Reject}\}$ is further denoted

$$\langle \mathcal{P}(in_{\mathcal{P}}), \mathcal{V}(in_{\mathcal{V}})\rangle \to out \ .$$

Definition 2.3.1 (Proof of Knowledge)**.** *Let x be a statement of language L in NP, and $W(x)$ the set of witnesses for x such that the following relation holds:*

$$\mathcal{R} = \{(x, w) : x \in L, w \in W(x)\} \ .$$

A proof of knowledge for relation $\mathcal{R}$ with soundness error ε is a two-party protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ with the following two properties:

- *Perfect completeness: If $(x, w) \in \mathcal{R}$, then a prover $\mathcal{P}$ who knows a witness w for x succeeds in convincing the verifier $\mathcal{V}$ of his knowledge. More formally:*

  $$\Pr[\langle \mathcal{P}(x, w), \mathcal{V}(x)\rangle \to \textsc{Accept}] = 1,$$

  i.e. given the interaction between the prover $\mathcal{P}$ and the verifier $\mathcal{V}$, the probability that the verifier is convinced is 1.

- *Soundness: If there exists a PPT prover $\tilde{\mathcal{P}}$ such that*

  $$\tilde{\varepsilon} := \Pr[\langle \tilde{\mathcal{P}}(x), \mathcal{V}(x)\rangle \to \textsc{Accept}] > \varepsilon,$$

  then there exists an algorithm $\mathcal{E}$ (called an extractor) which, given rewindable black-box access to $\tilde{\mathcal{P}}$, outputs a witness w' for x in time $\mathsf{poly}(\lambda, (\tilde{\varepsilon} - \varepsilon)^{-1})$ with probability at least $1/2$.

Chapter 2

Informally, a proof of knowledge has soundness error ε if a prover $\tilde{\mathcal{P}}$ without knowledge of the witness cannot convince the verifier with probability greater than ε assuming that the underlying problem (recovering a witness for the input statement) is hard. Indeed, if a prover $\tilde{\mathcal{P}}$ can succeed with a probability greater than ε, then the existence of the extractor (algorithm $\mathcal{E}$) implies that $\tilde{\mathcal{P}}$ can be used to compute a witness $w' \in W(x)$ in polynomial time.

We now recall the notion of honest-verifier zero-knowledge proof:

Definition 2.3.2 (Honest-Verifier Zero-Knowledge Proof)**.** *A proof of knowledge is {computationally, statistically, perfectly} honest-verifier zero-knowledge (HVZK) if there exists a PPT algorithm $\mathcal{S}$ (called simulator) whose output distribution is {computationally, statistically, perfectly} indistinguishable from the distribution* $\mathrm{View}(\langle \mathcal{P}(x,w), \mathcal{V}(x)\rangle)$ *obtained with an honest $\mathcal{V}$.*

Informally, the previous definition says that a genuine execution of the protocol can be simulated without any knowledge of the witness. In other words, the transcript of an execution between the prover and an honest verifier does not reveal any information about the witness.

Definition 2.3.3 (Public-Coin Proof)**.** *A proof of knowledge is said* public-coin *if the verifier challenges $\mathrm{CH}_1, \ldots, \mathrm{CH}_n$ are chosen following a public probability distribution.*

In practice, the challenges of most of the public-coin proofs of knowledge are chosen uniformly at random from public challenge sets.

2.3.3. Schwartz-Zippel Lemma and Variants

In some of our zero-knowledge proofs, we will rely on the Schwartz-Zippel Lemma or one of its variants. This lemma is a common probabilistic tool to determine whether a given polynomial is the zero polynomial. It states that, if we evaluate a non-zero polynomial into a random point, we obtain zero only with a small probability.

Lemma 2.3.4 (Schwartz-Zippel)**.** *Let $P \in \mathbb{F}[X]$ be a polynomial of degree $d > 0$; for any $\mathbb{S} \subset \mathbb{F}$,*

$$\Pr_{r \leftarrow \mathbb{S}}[P(r) = 0] \leq \frac{d}{|\mathbb{S}|} \ .$$

Proof. There are $|\mathbb{S}|$ possible different draws when we uniformly sample an element in $\mathbb{S}$. But P can have at most d roots, so the studied event occurs for at most d of these draws. □

Lemma 2.3.5 (Schwartz-Zippel, multi-point variant)**.** *Let $P \in \mathbb{F}[X]$ be a polynomial of degree $d > 0$; for any $\mathbb{S} \subset \mathbb{F}$ and any $t \geq 1$,*

$$\Pr_{r_1,\ldots,r_t \leftarrow \mathbb{S}}[P(r_1) = 0 \cap \ldots \cap P(r_t) = 0 \mid \{r_i\} \text{ are distinct}] \leq \frac{\binom{d}{t}}{\binom{|\mathbb{S}|}{t}} \ .$$

Proof. There are $\binom{|\mathbb{S}|}{t}$ possible different draws when we uniformly sample t distinct elements in $\mathbb{S}$. But P can have at most d roots, so the studied event occurs for at most $\binom{d}{t}$ of these draws. □

Lemma 2.3.6 (Schwartz-Zippel, multi-point variant 2)**.** *Let $P \in \mathbb{F}[X]$ be a polynomial of degree $d > 0$; for any $\mathbb{S} \subset \mathbb{F}$ and any $t, \ell \geq 1$,*

$$\Pr_{r_1,\ldots,r_t \leftarrow \mathbb{S}} [\#\{i : P(r_i) = 0\} = \ell \mid \{r_i\} \text{ are distinct}] \leq \frac{\max_{i \leq d} \left\{ \binom{i}{\ell} \cdot \binom{|\mathbb{S}|-i}{t-\ell} \right\}}{\binom{|\mathbb{S}|}{t}} .$$

Proof. There are $\binom{|\mathbb{S}|}{t}$ possible different draws when we uniformly sample t distinct elements in $\mathbb{S}$. But P can have at most d roots. Let us denote $i \leq d$ the number of roots of P in $\mathbb{S}$. The studied event occurs for $\binom{i}{\ell} \cdot \binom{|\mathbb{S}|-i}{t-\ell}$ possible draws. We thus get

$$\Pr_{r_1,\ldots,r_t \leftarrow \mathbb{S}} [\#\{i : P(r_i) = 0\} = \ell \mid \{r_i\} \text{ are distinct}] \leq \frac{\max_{i \leq d} \left\{ \binom{i}{\ell} \cdot \binom{|\mathbb{S}|-i}{t-\ell} \right\}}{\binom{|\mathbb{S}|}{t}} .$$

□

2.4. Secure Multiparty Computation

2.4.1. Secret Sharing Schemes

Along the thesis, the sharing of a value s is denoted $[\![s]\!] := ([\![s]\!]_1, \ldots, [\![s]\!]_N)$ with $[\![s]\!]_i$ denoting the *share* of index i for every $i \in [1 : N]$. For any subset of indices $J \subseteq [1 : N]$, we shall further denote $[\![s]\!]_J := ([\![s]\!]_i)_{i \in J}$.

Definition 2.4.1 (Threshold LSSS)**.** *Let $\mathbb{F}$ be a finite field and let $\mathbb{V}_1$ and $\mathbb{V}_2$ be two vector spaces over $\mathbb{F}$. Let t and N be integers such that $1 < t \leq N$. A (t, N)-*threshold linear secret sharing scheme *is a method to share a secret $s \in \mathbb{V}_1$ into N shares $[\![s]\!] := ([\![s]\!]_1, \ldots, [\![s]\!]_N) \in \mathbb{V}_2^N$ such that the secret can be reconstructed from any t shares while no information is revealed on the secret from the knowledge of $t - 1$ shares.*

Formally, an (t, N)-threshold LSSS consists of a pair of algorithms:

$$\begin{cases} \mathsf{Share} : \mathbb{V}_1 \times R \to \mathbb{V}_2^N \\ \mathsf{Reconstruct}_J : \mathbb{V}_2^t \to \mathbb{V}_1 \end{cases}$$

where $R \subseteq \{0, 1\}^*$ denotes some randomness space and where $\mathsf{Reconstruct}_J$ is indexed by a set (and defined for every) $J \subset [1 : N]$ such that $|J| = t$. This pair of algorithms satisfies the three following properties:

1. ***Correctness:*** *for every $s \in \mathbb{V}_1$, $r \in R$, and $J \subset [1 : N]$ s.t. $|J| = t$, and for $[\![s]\!] \leftarrow \mathsf{Share}(s; r)$, we have:*

$$\mathsf{Reconstruct}_J([\![s]\!]_J) = s.$$

2. ***Perfect $(t-1)$-privacy:*** *for every $s_0, s_1 \in \mathbb{V}_1$ and $I \subset [1 : N]$ s.t. $|I| = t - 1$, the two distributions*

$$\left\{ [\![s_0]\!]_I \;\middle|\; \begin{array}{c} r \leftarrow R \\ [\![s_0]\!] \leftarrow \mathsf{Share}(s_0; r) \end{array} \right\} \quad \text{and} \quad \left\{ [\![s_1]\!]_I \;\middle|\; \begin{array}{c} r \leftarrow R \\ [\![s_1]\!] \leftarrow \mathsf{Share}(s_1; r) \end{array} \right\}$$

are perfectly indistinguishable.

Chapter 2

3. ***Linearity:*** *for every* $v_0, v_1 \in \mathbb{V}_2^t$, $\alpha \in \mathbb{F}$, *and* $J \subset [1:N]$ *s.t.* $|J| = t$,

$$\mathsf{Reconstruct}_J(\alpha \cdot v_0 + v_1) = \alpha \cdot \mathsf{Reconstruct}_J(v_0) + \mathsf{Reconstruct}_J(v_1).$$

Definition 2.4.2 (Threshold Ramp LSSS)**.** *Let* $\mathbb{F}$ *be a finite field and let* $\mathbb{V}_1$ *and* $\mathbb{V}_2$ *be two vector spaces over* $\mathbb{F}$. *Let* t_1, t_2 *and* N *be integers such that* $1 \leq t_1 < t_2 \leq N$. *A* (t_1, t_2, N)-threshold ramp linear secret sharing scheme *is a method to share a secret* $s \in \mathbb{V}_1$ *into* N *shares* $[\![s]\!] := ([\![s]\!]_1, \ldots, [\![s]\!]_N) \in \mathbb{V}_2^N$ *such that the secret can be reconstructed from any* t_2 *shares while no information is revealed on the secret from the knowledge of* t_1 *shares.*

The formal definition of (t_1, t_2, N)-threshold ramp LSSS is similar to Definition 2.4.1 with the $\mathsf{Reconstruct}_J$ function defined over $\mathbb{V}_2^{t_2}$ (instead of $\mathbb{V}_2^t$) and with cardinalities $|I| = t_1$ and $|J| = t_2$ (instead of $|I| = t - 1$ and $|J| = t$). In particular an $(t-1, t, N)$-threshold ramp LSSS is an (t, N)-threshold LSSS.

Definition 2.4.3 (Additive Secret Sharing)**.** *An additive secret sharing scheme over* $\mathbb{F}$ *is an* (N, N)*-threshold LSSS for which the* Share *algorithm is defined as*

$$\mathsf{Share} : (s\,;\,(r_1, \ldots, r_{N-1})) \mapsto [\![s]\!] := \left(r_1,\, \ldots r_{N-1},\, s - \sum_{i=1}^{N-1} r_i\right),$$

with randomness space $R = \mathbb{F}^{N-1}$, *and the* $\mathsf{Reconstruct}_{[1:N]}$ *algorithm simply outputs the sum of all the input shares.*

Definition 2.4.4 (Shamir's Secret Sharing)**.** *The Shamir's Secret Sharing over* $\mathbb{F}$ *is an* $(\ell + 1, N)$*-threshold LSSS for which the* Share *algorithm builds a sharing* $[\![s]\!]$ *of* $s \in \mathbb{F}$ *as follows:*

- *sample* $r_1, \ldots, r_\ell$ *uniformly in* $\mathbb{F}$,
- *build the polynomial* P *as* $P(X) := s + \sum_{i=1}^{\ell} r_i X^i$,
- *build the shares* $[\![s]\!]_i$ *as evaluations* $P(e_i)$ *of* P *for each* $i \in \{1, \ldots, N\}$, *where* $e_1, \ldots, e_N$ *are public non-zero distinct points of* $\mathbb{F}$.

For any subset $J \subseteq [1:N]$, *s.t.* $|J| = \ell + 1$, *the* $\mathsf{Reconstruct}_J$ *algorithm interpolates the polynomial* P *from the input* $\ell + 1$ *evaluation points* $[\![s]\!]_J = (P(e_i))_{i \in J}$ *and outputs the constant term* s.

In the following chapters, we shall frequently use the following notions:

- **Sharing of a tuple.** If v is a tuple, a secret sharing $[\![v]\!]$ is defined coordinate-wise. The algorithms Share and $\mathsf{Reconstruct}$ further apply coordinate-wise.

- **Valid sharing.** We say that a sharing $[\![v]\!]$ is *valid* when there exists v such that

$$\forall J \text{ s.t. } |J| = \ell + 1, \mathsf{Reconstruct}_J([\![v]\!]_J) = v.$$

- **Consistent shares.** We say that shares $[\![v]\!]_{i_1}, \ldots, [\![v]\!]_{i_z}$ are *consistent* when there exist other shares $[\![v]\!]_{[1:N] \setminus \{i_1, \ldots, i_z\}}$ such that $[\![v]\!]$ is a valid sharing.

For a polynomial $P \in \mathbb{F}[X]$ of degree at most d, we define its sharing $[\![P]\!]$ as the sharing of the d-tuple of its coefficients.

2.4.2. Multiparty computation

A *multiparty computation* (MPC) protocol is an interactive protocol (as formally introduced in Section 2.3.1) involving multiple –possibly more than two– parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$. Each of these parties receives as input one share of a sharing $[\![x]\!]$. All together, the parties run the MPC protocol to compute $f(x)$ for some function f. At the end of the protocol, each party $\mathcal{P}_i$ outputs its own computed value of $f(x)$, denoted $f_i(x)$. In this paper, we only consider *complete* protocols for which an execution with honest parties results in all the parties outputting the right value $f(x)$. The *view* of a party $\mathcal{P}_i$ is composed of its input share $[\![x]\!]_i$, its random tape and all its received messages from the other parties (the sent messages can further be deterministically deduced from the other elements of the view).

We simply recall hereafter the notion of t-privacy for an MPC protocol in the *semi-honest model*.

Definition 2.4.5 (Privacy in the Semi-Honest Model)**.** *Let t and N be integers such that $1 \leq t < N$. Let Π_f be an MPC protocol with N parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$, computing a function f. The protocol Π_f is* t-private in the semi-honest model *if for all $I \subset [N]$ such that $|I| \leq t$, there exists a PPT algorithm $\mathcal{S}$ such that $\mathcal{S}(I, [\![x]\!]_I, f_I(x))$ is perfectly indistinguishable from the joint distributions of the views of the parties in I, where $f_I(x) := \{f_i(x) \mid i \in I\}$.*

As explained above, an N-sharing is usually distributed to N parties, meaning that each party gets one of the N shares. From those shares, the parties can perform distributed computations. Let assume that each party $i \in [N]$ receives the shares $[\![x]\!]_i$, $[\![y]\!]_i$ and $[\![P]\!]_i$ corresponding to shared values $x, y \in \mathbb{F}$ and polynomial $P \in \mathbb{F}[X]$. If these sharings are *linear*, they can perform the following operations:

- **Addition:** the parties locally compute $[\![x+y]\!]$ by adding their respective shares:

  $$\forall i, [\![x+y]\!]_i := [\![x]\!]_i + [\![y]\!]_i \ .$$

  This process is denoted $[\![x+y]\!] = [\![x]\!] + [\![y]\!]$.

- **Addition with a constant:** for a given constant α, the parties locally compute $[\![x+\alpha]\!]$ by doing:

  $$\forall i, [\![x+\alpha]\!]_i := [\![x]\!]_i + [\![\alpha]\!]_i$$

  where $[\![\alpha]\!]$ is a public sharing of α. For example, $[\![\alpha]\!]$ can be $(\alpha, 0, \ldots, 0)$ when using additive sharings, while it can be $(\alpha, \ldots, \alpha)$ when using Shamir's secret sharings. The process to add a constant is denoted $[\![x+\alpha]\!] = [\![x]\!] + \alpha$.

- **Multiplication by a constant:** for a given constant α, the parties locally compute $[\![\alpha \cdot x]\!]$ by multiplying their respective shares:

  $$\forall i, [\![\alpha \cdot x]\!]_i := \alpha \cdot [\![x]\!]_i \ .$$

  This process is denoted $[\![\alpha \cdot x]\!] = \alpha \cdot [\![x]\!]$.

- **Polynomial evaluation:** for a given r, the parties can locally compute $[\![P(r)]\!]$ by:

  $$\forall i, [\![P(r)]\!]_i := [\![P]\!]_i(r) = \sum_{j=0}^{d} [\![P_j]\!]_i \cdot r^j \ ,$$

  where $\{[\![P_j]\!]_i\}_j$ denotes the coefficients of $[\![P]\!]_i$. This process is denoted $[\![P(r)]\!] = [\![P]\!](r)$.

Chapter 2

2.5. Signature Schemes

In this thesis, we focus on signature schemes. Such schemes are formally defined as below.

Definition 2.5.1 (Signature scheme)**.** *A* signature scheme *is a triplet of polynomial-time algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verif})$ *such that:*

- KeyGen *outputs a random pair* $(\mathsf{pk}, \mathsf{sk})$ *where* pk *is a public key and* sk *is a secret key;*
- *given a secret key* sk *and a message* $m \in \{0,1\}^*$*,* Sign *produces a signature* σ*;*
- *given a public key* pk*, a message* $m \in \{0,1\}^*$ *and a signature* σ*,* Verif *outputs* 1 *if* σ *is a valid signature for* m *under* pk *(meaning that it is a possible output* $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ *for the corresponding* sk*) and it outputs* 0 *otherwise.*

The standard security property for a signature scheme is the *existential unforgeability* against chosen message attacks (EUF-CMA): an adversary $\mathcal{A}$ given pk and oracle access to $\mathsf{Sign}(\mathsf{sk}, \cdot)$ should not be able to produce a pair (σ, m) satisfying $\mathsf{Verif}(\mathsf{pk}, \sigma, m) = 1$ (for a message m which was not queried to the signing oracle).

Fiat-Shamir transformation. We can transform a public-coin honest-verifier zero-knowledge proof of knowledge into a signature scheme thanks to the famous Fiat-Shamir transformation [FS87]. The latter consists in removing the interactions of the proof systems while binding a proof transcript with the message to sign. The goal is that, even if we remove the interaction, a malicious signer should not be able to control the randomness used to generate the verifier challenges $\textsc{Ch}_1, \ldots, \textsc{Ch}_n$. The Fiat-Shamir heuristic consists in generating the challenge $\textsc{Ch}_i$ as (using the same notations as in Protocol 1)

$$\begin{aligned} h_i &\leftarrow \mathrm{Hash}(m, \textsc{com}, \textsc{Rsp}_1, \ldots, \textsc{Rsp}_{i-1}) \\ \textsc{Ch}_i &\leftarrow \mathrm{XOF}(h_i) \end{aligned}$$

where XOF is an extendable output function (a hash function with an arbitrary output size) and m is the message to sign. Since the output of a hash function looks like random bits, a malicious prover will not be able to forge easily valid signatures.

Chapter 3.

The MPC-in-the-Head Paradigm

Zero-knowledge proofs are an important tool for many cryptographic protocols and applications. To build them, many techniques exist. In this chapter, we will present one of them: the MPC-in-the-Head framework, introduced in 2007 by [IKOS07], which provides a generic way to build zero-knowledge proofs of knowledge using techniques from secure multiparty computation. Moreover, we will provide a complete overview of the recent techniques in the state of the art of this framework.

Contents

3.1. Generic Transformations **20**
3.2. Paradigm Usage **27**
3.3. Useful techniques **27**
3.4. State of the art **29**

3.1. Generic Transformations

3.1.1. General Setting

In the seminal work [IKOS07], Ishai, Kushilevitz, Ostrovsky and Sahai propose a framework to build zero-knowledge proofs for an arbitrary NP relation $\mathcal{R}$ using techniques from secure multi-party computation (MPC). More precisely, they show how to use an MPC protocol Π verifying the relation $\mathcal{R}$ in a black-box way to get such a proof.

Assume we want to build a zero-knowledge proof of knowledge of a witness w for a statement x such that $(x, w) \in \mathcal{R}$ for some relation $\mathcal{R}$. To proceed, we shall use an MPC protocol in which N parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$ securely and correctly evaluate a function f_x on a secret witness w with the following properties:

- each party $\mathcal{P}_i$ takes a share $[\![w]\!]_i$ as input, where $[\![w]\!]$ is a sharing of w;
- the function f_x on input w outputs Accept when $(x, w) \in \mathcal{R}$ and Reject otherwise;
- the protocol is t-private in the semi-honest model, meaning that the views of any t parties leak no information about the secret witness (see Definition 2.4.5 for a formal definition).
- the protocol is r-robust in the malicious model, meaning that the protocol outputs the right values as soon as there are at most r dishonest parties.

We can use this MPC protocol to build a zero-knowledge proof of knowledge of a witness w satisfying $(x, w) \in \mathcal{R}$. The prover proceeds as follows:

- she builds a random sharing $[\![w]\!]$ of w;
- she simulates locally ("in her head") all the parties of the MPC protocol;
- she sends a commitment of each party's view to the verifier, where such a view includes the party's input share, its random tape, and its received messages (the sent messages can further be deterministically derived from those elements);
- she sends the output shares $[\![f(w)]\!]$ of the parties, which should correspond to a sharing of Accept.

Then the verifier randomly chooses t parties and asks the prover to reveal their views. After receiving them, the verifier checks that they are consistent with an honest execution of the MPC protocol and with the commitments. Since only t parties are opened, the revealed views leak no information about the secret witness w, which ensures the zero-knowledge property. On the other hand, the random choice of the opened parties ensures the soundness of the proof system. Indeed, a prover that does not know a witness w satisfying $(x, w) \in \mathcal{R}$ will need to corrupt the computation of some parties such that the emulated MPC protocol indicates that $f_x(w) =$ Accept (while it is not true), but then the verifier will catch the malicious prover as soon as she will ask to open one of these corrupted parties (see Figure 3.1), which occurs with constant probability.

The probability that a malicious prover can convince a verifier is named the *soundness error* of the proof of knowledge and it is usually denoted ε (or ϵ in some articles). To decrease

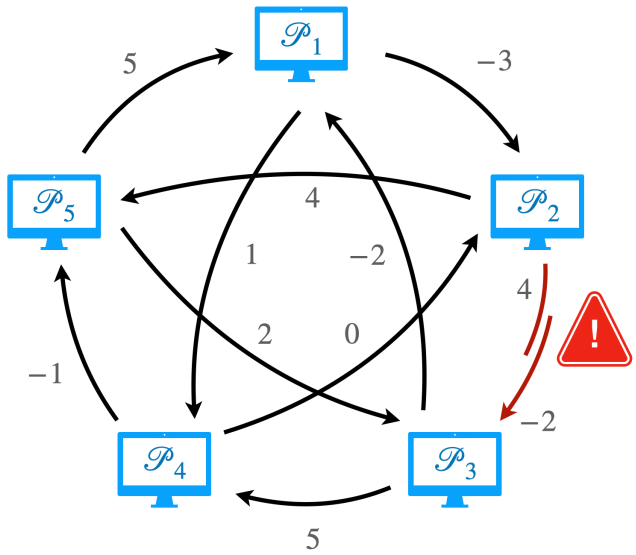


Figure 3.1.: Let us assume that we have a 2-private MPC protocol involving $N := 5$ parties. A malicious prover does not know a witness such that the MPC protocol outputs Accept, thus she needs to cheat when emulating the MPC protocol. The optimal cheating strategy (when $r := 0$) is to corrupt the communication between two parties. For example, in the figure, the party $\mathcal{P}_2$ sends the message "4" but the party $\mathcal{P}_3$ gets "−2" as message from $\mathcal{P}_2$. The verifier will ask to reveal the computation of $t := 2$ parties. The verifier will catch the malicious behavior as soon as the two parties involved in the corrupted communication are open, implying that the probability that the verifier detects the cheating is $\frac{\binom{N-2}{t-2}}{\binom{N}{t}} := \frac{1}{10}$.

this cheating probability, the zero-knowledge proof is repeated several times (we denote this number τ in this manuscript) using fresh randomness.

The exact security of the IKOS framework[1] is analyzed in [GMO16]. The soundness error of the construction is given by the following theorem.

Theorem 3.1.1 ([GMO16])**.** *Let us consider a NP relation $\mathcal{R}$. We assume that we have a t-private and r-robust MPC protocol Π evaluating a function f_x such that $\forall w, f_x(w) = \textsc{Accept} \Leftrightarrow (x, w) \in \mathcal{R}$. Then the zero-knowledge protocol built from Π using the MPCitH paradigm has a soundness error*

$$\varepsilon := \max\left\{\frac{\binom{r}{t}}{\binom{N}{t}}, \sum_{j=0}^{k} 2^j \frac{\binom{k}{j}\binom{N-2k}{t-j}}{\binom{N}{t}}\right\}$$

with $k = \lfloor r/2 \rfloor + 1$.

Remark 3.1.2. *Let us consider the case where the MPC protocol is only passively-secure, meaning that the protocol is ensured to produce the correct output only when there is no corrupted parties (i.e. $r = 0$ where r is the robustness threshold). In that case, Theorem 3.1.1*

[1] The authors of [IKOS07] only made an asymptotic analysis.

Chapter 3

gives that the soundness error of the resulting proof of knowledge is

$$\varepsilon := 1 - \frac{t(t-1)}{N(N-1)}.$$

We can remark that this quantity is minimal when t is maximal, i.e. when $t = N - 1$. In that case, the soundness error is $\varepsilon = \frac{2}{N}$.

Remark 3.1.3. *[IKOS07] considers two cases. The first one corresponds to the setup where the MPC protocol is passively-secure and the privacy threshold is 2. The soundness error of the resulting proof is $1 - \frac{1}{\binom{N}{2}}$. The second one corresponds the setup where $t = r > 0$ and $N = \Omega(t)$, for which the soundness error ε is in $2^{-\Omega(t)}$. When taking a large t, ε can be negligible without any repetition.*

Theorem 3.1.1 holds for any MPC protocol. However, the communication cost of the resulting proof system will highly depend on the used protocol. The MPC-in-the-Head framework was introduced in 2007 and has been considered as a theoretical result for almost a decade. It was only in 2016 that the first practical proof system relying on this framework was proposed [GMO16]. From then on, the framework has been much more studied and many concrete instantiations have been proposed. For instance, one year later, [CDG+17] proposed the first MPCitH-based signature schemes, Fish and Picnic. Picnic has been then submitted to the NIST call for post-quantum cryptosystems in 2017 [NIS16].

3.1.2. Broadcast-based Multiparty Computation

Submitted at the same conference than [CDG+17], the article [AHIV17] proposes a totally different proof system relying on the MPC-in-the-Head paradigm. The authors restrict the seminal work of [IKOS07] to the case of multiparty computation using only broadcast as communication (and no peer-to-peer communication anymore). While it can be seen as a subcase of the [IKOS07] framework, it enables to build more efficient proof systems. Since it relies only on broadcast, all communications are public. The prover thus does not need to commit them and can simply send them to the verifier. Then, when getting a party's view, the verifier will check that the sent messages are consistent with the broadcast communications. In Section 3.1.1, the verifier checked that the open views were consistent between them (and with the commitments), but here the verifier only needs to check that the open views are consistent with the broadcast communications. It implies that the verifier will more efficiently catch the cheating since she just needs to open a party's view with cheating (see Figure 3.2), while in the previous model, the cheating can be on peer-to-peer communication, requiring that the verifier asks the opening of the two concerned parties to detect it (as in Figure 3.1).

To build a zero-knowledge proof of knowledge using a broadcast-based MPC protocol, the prover shall proceed as follows:

- she builds a random sharing $[\![w]\!]$ of w;
- she simulates locally ("in her head") all the parties of the MPC protocol;
- she sends a commitment of each party's view to the verifier, where such a view includes the party's input share and its random tape;
- she sends all the broadcast messages;

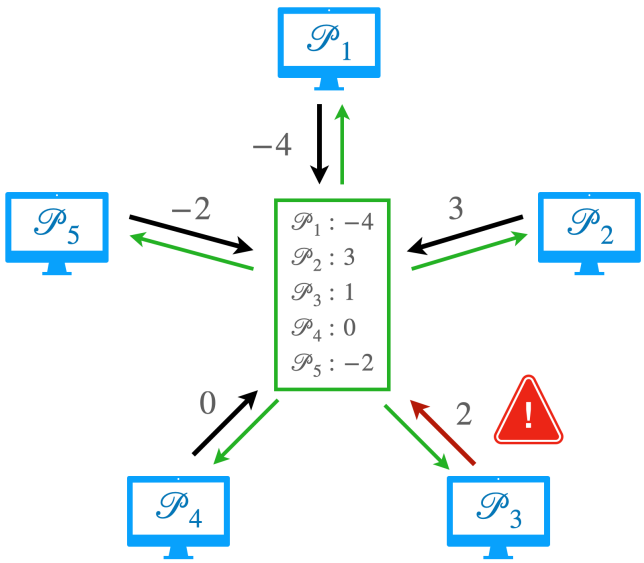


Figure 3.2.: Let us assume that we have a 2-private MPC protocol involving $N := 5$ parties. This time, we assume that the parties only use a broadcast channel, meaning that all broadcast values are known by everybody. Since the malicious prover does not know a valid witness, she needs to cheat on the computation of at least one party. Then, the verifier will ask to reveal the computation of $t := 2$ parties. The verifier will catch the malicious prover with probability $\frac{2}{N} := \frac{2}{5}$.

- she sends the output shares $[\![f(w)]\!]$ of the parties, which should correspond to a sharing of Accept.

Then the verifier randomly chooses t parties and asks the prover to reveal their views. After receiving them, the verifier checks that they are consistent with the given broadcast messages and with the commitments. While the number of messages in a peer-to-peer multiparty computation is in $O(N^2)$ with N the number of parties, it is only in $O(N)$ when using broadcast. The exchanged messages sometimes contain some redundancies. For example, it is the case when the parties broadcast shares of a Shamir's secret sharing, which are evaluations of a polynomial. In that case, the prover can save communication by compressing the broadcast messages.

The soundness error of the resulting proof system is given by the following theorem (mix of theorems of [AHIV17] and [DOT21]).

Theorem 3.1.4 ([AHIV17; DOT21])**.** *Let us consider a NP relation $\mathcal{R}$. We assume that we have a t-private and r-robust MPC protocol Π evaluating a function f_x such that $\forall w, f_x(w) =$ Accept $\Leftrightarrow (x, w) \in \mathcal{R}$. We denote δ the robustness error of Π, i.e. the probability that Π with at most r corrupted parties outputs Accept while parties' input shares do not form a valid sharing of a value w such that $(x, w) \in \mathcal{R}$. Moreover, we assume that Π only uses a* broadcast channel *as communication between parties. Then the zero-knowledge protocol built from Π using the MPCitH paradigm has a soundness error*

$$\varepsilon := \frac{\binom{N-r-1}{t}}{\binom{N}{t}} + \delta \cdot \left(1 - \frac{\binom{N-r-1}{t}}{\binom{N}{t}}\right).$$

Chapter 3

Remark 3.1.5. *The formula given in Theorem 3.1.4 differs from these in [AHIV17] and [DOT21]. In [AHIV17, Theorem 3.5], the authors simplify the equation as*

$$\varepsilon \leq \frac{\binom{N-r-1}{t}}{\binom{N}{t}} + \delta \leq \left(1 - \frac{r+1}{N}\right)^t + \delta \leq \left(1 - \frac{r}{N}\right)^t + \delta$$

while [DOT21, Theorem 1] only considers the case $r = 0$ and $t = N-1$ to get $\frac{1}{N} + \delta \cdot \left(1 - \frac{1}{N}\right)$.

Remark 3.1.6. *Let us stress that a malicious prover is not forced to commit a valid input sharing (i.e. a sharing with consistent shares). For example, when using a $(t+1, N)$-threshold Shamir's secret sharing, a malicious prover is not forced to commit evaluations of a degree-t polynomial. The MPC protocol should then check that the sharing is valid, e.g. by using a Reed-Solomon proximity test. Let us observe that this issue is irrelevant in the case of the additive sharing scheme since any N values $(v_1, \ldots, v_N)$ form a valid sharing of $\sum_{i=1}^{N} v_i$.*

The MPCitH-based constructions relying on broadcast-based MPC protocols can be split into two categories:

- sublinear proof systems, relying on multiparty computation which is secure in the malicious model ($r > 0$),
- linear proof systems targeting small circuits (e.g. to construct efficient signature schemes), relying on passively-secure multiparty computation ($r = 0$).

3.1.2.1. Sublinear Proof Systems

In [AHIV17], the authors proposed a proof system fitting this broadcast setting, named Ligero. The latter relies on proximity tests to get a robust MPC protocol. Combined with packed secret sharings[2], it achieves sublinear proof size. More precisely, the communication cost scales in $O(\sqrt{|C|})$ where $|C|$ is the number of gates in the considered circuit.

Since it relies on packed secret sharings, the used field must be large enough to enable the existence of such sharings. To support Boolean circuits, it requires embedding the computation in a larger field. In that case, a Boolean value will be represented by a field element and quadratic constraints on the form $X^2 - X = 0$ will ensure that the field elements really correspond to 0 or 1. A few years later, [GSV21] proposed a tailored version of Ligero (named Booligero) in this context of Boolean circuits.

3.1.2.2. Proof Systems for Small Circuits

One year after [AHIV17], [KKW18] proposed a proof system that fits the broadcast setting, but with a totally different construction. The proposed construction only uses additive sharings and has the maximal privacy threshold $t := N - 1$, meaning that the prover will reveal all the parties' views except one. Without optimization, the achieved size would be huge, but [KKW18] proposes several tricks to achieve practical sizes:

[2] The packed secret sharing scheme is a generalization of the Shamir's secret sharing scheme: to share simultaneously $s_1, \ldots, s_{|s|} \in \mathbb{F}$, it samples a random polynomial P such that $P(e'_j) = s_j$ for all j, and evaluates it into the parties' evaluation points $\{e_i\}_i$, where $\{e_i\}_i$ and $\{e'_j\}_j$ are public distinct points of $\mathbb{F}$.

- Since the input sharing $[\![w]\!]$ is additive, each share except the last one can be derived from a seed using a pseudo-random generator. Thus, revealing an input share simply consists in sending the corresponding seed.

- With the previous point, the prover only needs to reveal $N-1$ seeds if the last party remains hidden, $N-2$ seeds and an uncompressed share otherwise. In order to further decrease communication, [KKW18] suggests to generate these seeds using a puncturable pseudo-random function. More precisely, the authors propose to use a GGM tree (see Section 2.2.2), decreasing the number of revealed seeds to $\log_2(N)$.

This framework (passively-secure broadcast-based protocol with additive sharings) inspires many works such as [BN20], [DOT21] and [KZ22] (see Section 3.4 for a more complete list). Here are the main ideas to develop an efficient MPC protocol in this framework:

- [KKW18] relies on a preprocessing phase (independent of the secret) to prepare a set of Beaver triples (a Beaver triple is a triple (a, b, c) such that (a, b) is chosen uniformly at random and $c = a \cdot b$). Thanks to a cut-and-choose strategy, the verifier has the guarantee that these triples are well-formed (see Section 3.3.1 for details). Then, [KKW18] proposes an MPC protocol that computes the circuit which represents f_x and which outputs $f_x(w)$.

- Instead of having an MPC protocol which computes $f_x(w)$, [BN20] proposes an MPC protocol which validates the statement $f_x(w) \stackrel{?}{=} \textsc{Accept}$. It enables us to avoid the time-consuming cut-and-choose phase of [KKW18].

In this line of research, the prover needs to emulate N parties by repetition, where N is a flexible parameter of the proof system. Since the soundness error of one repetition is greater than $\frac{1}{N}$, the prover needs to perform at least $\frac{\lambda}{\log_2 N}$ repetitions to achieve a security of λ bits, thus the prover needs to emulate at least $\lambda\frac{N}{\log_2 N}$ times the party's computation. Similarly, the verifier needs to re-emulate at least $\frac{\lambda(N-1)}{\log_2 N}$ parties. Since emulating the MPC protocol was the computational bottleneck of these schemes, most of the works in this setting proposed several trade-offs of their construction: the larger N, the shorter the proof and the slower the computation.

This state of affairs changed in 2023. [AGH+23] shows that these numbers can be reduced to $\lambda\frac{\log_2 N+1}{\log_2 N}$ for the prover and $\lambda\frac{\log_2 N}{\log_2 N} = \lambda$ for the verifier *without impacting the signature size*. With their optimization, N has only a slight impact on the cost of emulating the MPC protocol anymore, but it still has a linear impact on the commitment phase where the prover needs to commit the input shares of all the parties for each repetition.

3.1.3. Linear Broadcast-based Multiparty Computation

The schemes presented in Section 3.1.2.2 achieve very competitive performance mainly thanks to the additive sharing and the tweak of generating shares via GGM trees. The latter is not possible for another type of sharing. The achieved sizes in Section 3.1.2.1 are not competitive when targeting small circuits/statements.

The recent article [FR22] proposes a new approach to build MPCitH-based proof systems. It considers a subcase of the broadcast setting: it deals with broadcast-based multiparty computation which only performs *linear operations* on sharings. For example, given two

sharings $[\![a]\!]$ and $[\![b]\!]$, it prevents from computing the sharing $[\![a \cdot b]\!]$ by multiplying each share $[\![a]\!]_i$ by the share $[\![b]\!]_i$, which would be accurate when using Shamir's secret sharings (it is used in Ligero [AHIV17]). This subclass of MPC protocols satisfies the following property: if Π is a broadcast-based MPC protocol that only performs linear operations on $(t+1, N)$-threshold linear secret sharings, then restricting Π to any subset of $t+1$ parties forms a *valid* MPC protocol which computes the *same functionality*. The procedure to build a zero-knowledge proof is exactly the same as in Section 3.1.2, but this subclass of MPC protocol enables a refined analysis of the soundness, as given by the following theorem.

Theorem 3.1.7 ([FR22])**.** *Let us consider a* NP *relation $\mathcal{R}$. We assume that we have a t-private (passively-secure) MPC protocol Π evaluating a randomized function f_x such that*

- *for w satisfying $(x, w) \in \mathcal{R}$, $f_x(w) = \textsc{Accept}$,*
- *for w such that $(x, w) \notin \mathcal{R}$, $f_x(w) = \textsc{Reject}$ with high probability.*

Moreover, we assume that Π only uses a broadcast channel as communication between parties and only performs linear operations on $(t, t+1+\Delta, N)$-threshold ramp linear secret sharings. Let us denote p the probability that $f_x(w) = \textsc{Accept}$ when $(x, w) \notin \mathcal{R}$. Then the zero-knowledge protocol built from Π using the MPCitH paradigm has a soundness error

$$\varepsilon := \frac{\binom{t+\Delta}{t}}{\binom{N}{t}} + p \cdot \frac{t}{t+\Delta+1} \cdot \binom{N-t}{\Delta+1}.$$

When considering threshold sharings ($\Delta := 0$), we get a soundness error of

$$\varepsilon := \frac{1}{\binom{N}{t}} + p \cdot \frac{t \cdot (N-t)}{t+1}.$$

Remark 3.1.8. *When we apply Theorem 3.1.7 to the case of the additive sharing case ($t := N-1$, $\Delta := 0$), we obtain the soundness error*

$$\varepsilon := \frac{1}{N} + p \cdot \left(1 - \frac{1}{N}\right),$$

which corresponds to the soundness obtained by the schemes in Section 3.1.2.2. Thus, in that setting, we can stress that allowing only linear operations does not impact the soundness.

Theorem 3.1.7 is especially interesting when considering a very-low privacy threshold t. For example, let us consider the case where $t := 1$ (and with $\Delta = 0$). The soundness error of the resulting scheme is

$$\varepsilon := \frac{1}{N} + p \cdot \frac{N-1}{2}.$$

When p is negligible, we obtain the same soundness as when using the additive sharing scheme, but here the verifier asks to open only $t = 1$ party. Since verifying a proof transcript consists in re-emulating the open parties, revealing only t parties (with t small) leads to a very efficient scheme. [FR22] shows that the prover also benefits from this small privacy threshold. However, the obtained communication costs in that setting are a bit larger than those of Section 3.1.2.2. Theorem 3.1.7 thus tends to produce faster schemes but with a larger communication cost.

The formalization of this linear broadcast-based multiparty computation model is a contribution of this thesis, which is extensively described in Chapter 8.

3.2. Paradigm Usage

In the state of the art, the MPC-in-the-Head paradigm has been studied to achieve two different goals. The first one consists in designing zero-knowledge proofs for a class of circuits, while the second one consists in designing identification schemes (and signature schemes through the Fiat-Shamir transformation).

On one hand, the challenge of the first goal is to design an MPC protocol that can be converted into an efficient proof system for a chosen class of circuits (e.g. an arithmetic circuit). The statement to be proved with the proof scheme is straightforward.

On the other hand, when designing identification schemes, the statement to be proved is often left to the designers as soon as it corresponds to a secure one-way function. The challenge is then to choose a statement and design a dedicated MPC protocol which can be converted into an efficient identification scheme. Many articles explored this setting, searching for the statement which would give the most efficient schemes with the existing MPC-in-the-Head techniques. Three trends can be observed in the state of the art:

- some identification schemes are relying on standard symmetric primitives like AES (e.g. [BDK+21b]);
- some are relying on MPC-friendly primitives like LowMC or Rain (e.g. [DKR+21]);
- as the last option, some are relying on hard problems that have existed for a long time and are thus well understood, like the syndrome decoding problem.

The first option leads to the more conservative schemes, but the achieved communication costs are quite large. The second option currently gives the shortest communication costs, but since it relies on recent primitives, it is less conservative. The last option is between the two other ones in terms of performance and can be used as a trade-off between performance and security concerns.

3.3. Useful techniques

To design an MPC protocol that aims to be used with the MPC-in-the-Head framework, designers can rely on some techniques that have been proposed in the literature. In what follows, we describe some of them.

3.3.1. Protocols with Helper

One can assume that the parties take an auxiliary input which is a sharing $[\![y]\!]$ of a random value y satisfying some public equation $E(y) = 0$ (where E might depend on the public statement x). In practice, this assumption is verified thanks to a cut-and-choose strategy run before the emulation of the MPC protocol. Introduced by [KKW18] using a preprocessing phase, this technique has been formalized in [Beu20] as *protocols with helper*.

Using a "helper" is very powerful. When receiving the shares $[\![y]\!]$ of y, the parties can use them to perform advanced computation assuming $E(y) = 0$, without needing to perform any check on the received shares. For example, [KKW18] uses this technique to produce Beaver triples (*i.e.* the MPC protocol can take as input three sharings $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ for random a, b and c satisfying $c = a \cdot b$) while [FMRV22b] uses it to ensure that some parts of the MPC input encode a random binary vector.

This technique presents the advantage that the communication cost is independent of the size of y and the expression of E, since it relies only on seeds. However, it suffers from a high computational cost, implying that the number N of parties involved in the MPC protocol must stay low to obtain a practical scheme.

3.3.2. Statistical checkings

The high-level idea of the MPC-in-the-Head paradigm consists in building a zero-knowledge proof of knowledge of a witness w that satisfies $(x, w) \in \mathcal{R}$ using a multiparty computation evaluating a functionality f_x such that

$$(x, w) \in \mathcal{R} \Leftrightarrow f_x(w) = \textsc{Accept}.$$

In practice, we can relax this construction. Instead of being a deterministic function, f_x could be a randomized function performing a statistical test on w:

- if $(x, w) \in \mathcal{R}$, then f_x outputs Accept;
- if $(x, w) \notin \mathcal{R}$, then f_x outputs Reject with *high probability*.

It means that some false positive events can occur: f_x can sometimes output Accept even if $(x, w) \notin \mathcal{R}$. The probability that it occurs is called the *false positive rate* or the *false positive probability*. To sum up, we can consider a setting in which the output of the protocol has a probability distribution of the form described in Table 3.1. To have a sound proof of knowledge, the randomness of the statistical test should not be controlled by the prover. The solution consists in letting the verifier provide this randomness *after* the commitment of the input shares of the MPC protocol.

	Output of f_x	
	Accept	Reject
w s.t. $(x, w) \in \mathcal{R}$	1	0
w s.t. $(x, w) \notin \mathcal{R}$	p	$1 - p$

Table 3.1.: Probability distribution of the output of the MPC protocol, where p is the false positive rate.

Relaxing the definition of f_x enables us to use lighter MPC protocols, which leads to more efficient MPCitH-based proofs of knowledge. We could imagine relaxing the definition of f_x even more by allowing false negative events (*i.e.* f_x can output Reject when $(x, w) \in \mathcal{R}$ with small probability), but there is currently no such proposal in the literature, and so it is not sure it would be useful.

In the MPC-in-the-Head state of the art, the first scheme using this relaxation is the [BN20] protocol. To deal with multiplications in MPC (*i.e.* to compute a sharing $[\![x \cdot y]\!]$ from $[\![x]\!]$ and $[\![y]\!]$), a standard technique consists in using Beaver triples [Bea92]: the protocol takes as inputs a random triple of sharings $([\![a]\!], [\![b]\!], [\![c]\!])$ such that $a \cdot b = c$ and uses them to compute $[\![x \cdot y]\!]$ from $[\![x]\!]$ and $[\![y]\!]$. When using it in an MPC-in-the-Head paradigm, this technique presents the disadvantage that the verifier should be convinced that $a \cdot b = c$. Since the desired property is independent of the secret, we can use protocols with helper to

proceed. However, it implies that the obtained scheme would not be efficient because of the computational bottleneck of the protocols with helper.

[BN20] proposes an alternative solution. In the MPC-in-the-Head paradigm, the secret is known by the prover. If the MPC protocol involves such a multiplication between two secret values x and y, she can easily compute their product $z := x \cdot y$. Then, instead of computing a sharing $[\![z]\!]$ from $[\![x]\!]$ and $[\![y]\!]$, the prover can directly give $[\![z]\!]$ as input of the MPC protocol. Then one just needs to check that the sharings $[\![x]\!]$, $[\![y]\!]$ and $[\![z]\!]$ satisfy $z = x \cdot y$. Checking this relation is lighter than computing $[\![z]\!]$. [BN20] proposes an MPC protocol that takes as input the three sharings $([\![x]\!], [\![y]\!], [\![z]\!])$ and a random Beaver triple $([\![a]\!], [\![b]\!], [\![c]\!])$ and which outputs

- Accept when $z = x \cdot y$ and $c = a \cdot b$,
- Reject when $z \neq x \cdot y$ or $c \neq a \cdot b$ with high probability.

This alternative solution also uses a Beaver triple as the previous one, however the verifier does not need to be convinced that the Beaver triple is well-formed since the MPC protocol makes the checking itself. While it introduces a false positive rate, it avoids using the protocols with helper and leads to more efficient schemes.

3.3.3. MPCitH with Rejection

Sharing a value x using an additive N-sharing over $\mathbb{Z}/q\mathbb{Z}$ consists in sampling $[\![x]\!]_1, \ldots, [\![x]\!]_N \in \mathbb{Z}/q\mathbb{Z}$ such that

$$x = [\![x]\!]_1 + \ldots + [\![x]\!]_N \mod q.$$

A proof system relying on the MPC-in-the-Head always needs to include some shares in the proof transcript, so the bit size of a share has a direct impact on the proof size. In some contexts, one may want to share a small value (small compared to q). For example, [FMRV22b] needs to share a binary vector x while working with $q \approx 2^{256}$. In that case, sending a share of x is much more expensive than sending x. [FMRV22b] proposes to share such a small value in an alternative way: instead of sharing over $\mathbb{Z}/q\mathbb{Z}$, one can share over small integers. It consists in sampling $[\![x]\!]_1, \ldots, [\![x]\!]_N$ in some range $\{-A, \ldots, A\}$ such that

$$x = [\![x]\!]_1 + \ldots + [\![x]\!]_N \text{ over } \mathbb{Z}.$$

Such sharings leak information about the shared value x, but [FMRV22b] shows that one can prevent leakage using rejection, *i.e.* by allowing the prover to abort the protocol in some specific situations. Taking a small A leads to a short proof size but implies a high rejection rate. Thus selecting the parameter A consists in choosing a trade-off between the proof size and the computational performance.

3.4. State of the art

Since [GMO16], there have been a lot of works dealing with the MPC-in-the-Head paradigm. This section proposes a short overview of the state of the art. Let us stress that it is not an exhaustive list of the existing works, but rather a list of the main results.

We can divide the contributions of the works into several categories:

1. works which propose a new MPC-in-the-Head transformation for a class of MPC protocols (or some optimizations of an existing transformation);
2. works which propose a new (optimized) MPC protocol that leads to efficient proof systems when using one of the MPC-in-the-Head transformations;
3. works which find the best specialized MPC protocol for a specific circuit and it often leads to efficient signature schemes;
4. works which aim to produce schemes with advanced functionalities (such as ring signature scheme) thanks to the MPCitH paradigm.

One article can propose contributions in several categories. For example, [AHIV17] introduces the MPC-in-the-Head transformation with broadcast and also proposes Ligero, a sublinear proof of knowledge for the arithmetic circuits.

Contributions from the first category have already been presented in Section 3.1. In what follows, we describe the works of the second and third categories. Regarding the contributions of the last category, the literature remains still sparse and it is difficult to provide a clear overview.

3.4.1. Zero-knowledge proofs of knowledge for arbitrary circuits

In 2016, [GMO16] proposed ZKBoo, the first practical zero-knowledge proof system relying on the MPC-in-the-Head paradigm.While the achieved proof size was not competitive with the schemes from the SNARK technology, ZKBoo significantly outperformed them in terms of proving time. One year later, [CDG+17] optimized ZKBoo by reducing the proof size by a factor of two at no additional computational cost. This optimized proof system was named ZKB++.

In 2017, [AHIV17] proposed Ligero, a sublinear proof system for arithmetic circuits. It relies on an MPC protocol which is robust in the malicious model thanks to a code proximity test. It achieves sublinear communication costs by packing several witness coordinates in one sharing, which is made possible by the use of Shamir's secret sharing.

From 2018, there is a series of works proposing efficient proof systems for arithmetic circuits for which the communication cost is linear into the number n of multiplicative gates:

- [KKW18] uses Beaver triples to evaluate an arithmetic circuit in MPC and relies on a protocol with helper to generate these triples;
- [BN20] proposes a solution to avoid the helper from [KKW18] thanks to their sacrificing-based multiplication verification, obtaining a 5-round proof system which achieves a communication cost in around $5n$;
- [BDK+21b] proposes Banquet[3] which uses the polynomial-based technique of [BBC+19b] to verify multiplications, which is a 7-round proof system achieving a communication cost scaling in $n + O(\sqrt{n})$;
- [DOT21] proposes Limbo which can be considered as a generalization of Banquet for more rounds, which is a $\Theta(\log_2 n)$-round proof system achieving a communication cost in $n + O(\log_2 n)$;

[3]Banquet can be easily adapted to a generic proof system even if it has not been proposed as such.

- [KZ22] proposes BN++, a highly-optimized version of the [BN20] scheme achieving a communication cost in around $2n$;
- [KZ22] proposes Helium, which is a 7-round proof system relying on the polynomial-based multiplication verification as Banquet and achieving a communication cost of around $2n$.

[KZ22] provides a nice overview of all the existing MPCitH techniques. Currently, there are two main approaches to deal with multiplications:

- The *sacrificing-based verification* proposed by [LN17; BN20]. It consists in using Beaver triples to check multiplications, but it does not require that the prover explicitly convinces the verifier that these Beaver triples are well-built. As shown in [KZ22], this verification can be optimized and batched for a large number of multiplications. It can be also adapted to the matrix setting (see [Fen22] for example).

  Let us describe the batched version of the MPC protocol. We want to check n multiplication triples $([\![x_i]\!], [\![y_i]\!], [\![z_i]\!])$, *i.e.* check that $z_i = x_i \cdot y_i$ for all i. To proceed, the MPC protocol takes as input a random *dot-product tuple* $(([\![a_i]\!], [\![b_i]\!])_{i\in[n]}, [\![c]\!])$ verifying $c = \langle a, b\rangle$ to sacrifice and runs the following computation:

  1. the parties get a random $\varepsilon \in \mathbb{F}^n$ (from the verifier in the MPCitH paradigm);
  2. the parties locally set $[\![\alpha]\!] = \varepsilon \circ [\![x]\!] + [\![a]\!]$ and $[\![\beta]\!] = [\![y]\!] + [\![b]\!]$, where $\circ$ is the coordinate-wise multiplication;
  3. the parties broadcast $[\![\alpha]\!]$ and $[\![\beta]\!]$ to obtain α and β;
  4. the parties locally set $[\![v]\!] = -[\![c]\!] + \langle \varepsilon, [\![z]\!]\rangle + \langle \alpha, [\![b]\!]\rangle + \langle \beta, [\![a]\!]\rangle - \langle \alpha, \beta\rangle$;
  5. the parties broadcast $[\![v]\!]$ to obtain v;
  6. the parties output Accept if $v = 0$ and Reject otherwise.

  If one of the multiplication triples is invalid (or if the dot-product tuple is not well-built), this MPC protocol outputs Reject except with probability $\frac{1}{|\mathbb{F}|}$. In some settings, we can remove the computation of $[\![\beta]\!]$ and save the associated communication (see [KZ22] for details).

- The *polynomial-based verification* proposed by [BBC+19b; BDK+21b]. Let us assume that, given (x_i, y_i, z_i), we want to check that $z_i = x_i \cdot y_i$ for all i. The idea is to represent the values $(x_1, \ldots, x_n)$, $(y_1, \ldots, y_n)$ and $(z_1, \ldots, z_n)$ respectively into three polynomials S, T and P such that

$$\begin{aligned} S(\gamma_1) &= x_1 & T(\gamma_1) &= y_1 & & P(\gamma_1) &= z_1 \\ &\vdots & &\vdots & \text{and} & &\vdots \\ S(\gamma_n) &= x_n, & T(\gamma_n) &= y_n, & & P(\gamma_n) &= z_n \end{aligned}$$

  for some distinct public points $\gamma_1, \ldots, \gamma_n$. Let us stress that computing these polynomials is communication-free in MPC since the interpolation formula is linear into the secret values. Then, the *polynomial-based verification* consists in checking that $S \cdot T - P$ is zero over $\{\gamma_1, \ldots, \gamma_n\}$ thanks to the Schwartz-Zippel lemma. The communication cost of this verification depends on the number of rounds. If the MPC protocol asks the verifier for randomness only once, then the communication cost due to the verification

will be linear into the number n of multiplications we want to check. If we allow an additional round (see [BDK+21b]), the communication cost scales in $O(\sqrt{n})$. More generally [DOT21], if the MPC protocol asks the verifier for randomness t times, the cost scales in $O(t \cdot \sqrt[t]{n})$, which is $O(\log n)$ for $t := \log n$.

The two above techniques highly depend on the size of the underlying field. They will be efficient for large fields, but not for small fields. For example, if we work on the binary field, the MPC protocol for sacrificing-based verification has a false positive rate of 1/2, and it will have a high impact on the performance of the produced scheme. [KZ22] compiles several methods to deal with small fields:

- the simplest idea consists in lifting all the computations in a field extension;
- the second idea consists in repeating the MPC protocol several times to get a small global false positive rate;
- the latter idea consists in using reverse multiplication friendly embedding (RMFE) to lift the computation.

Other techniques are possible, but they are specific to the context. All these techniques enable us to decrease the impact of working over a small field. We refer the reader to [KZ22] for details.

3.4.2. Zero-knowledge proofs of knowledge for dedicated circuits

If we want to build a proof system for a specific circuit, we can rely on the schemes of the previous subsection. For example, if we want to build a proof of knowledge for an AES key k satisfying $y = \mathrm{AES}_k(0)$ for a public y, we can transform the statement into an arithmetic circuit and use a generic proof system (as Limbo or Helium) to get the desired scheme. While this approach presents the advantage of not relying on a new construction, it could prevent achieving smaller communication costs because converting a statement into a circuit often does not take advantage of the arithmetic properties of the statement.

Another approach would consist in building a proof system from an MPC protocol optimized for the considered statement. The goal is to exploit the arithmetic structure of the statement to produce a more efficient MPC protocol. It does not lead to better schemes when considering a statement which can be naturally written as a circuit (as those derived from some block ciphers), but it is more efficient when considering an arithmetic statement as the syndrome decoding problem (as in Chapter 4 and in Chapter 5).

Let us remark that focusing on a specific statement instead of a large class of statements is interesting to build efficient signature schemes.

Here are the articles proposing dedicated MPC protocols:

- LegRoast and PorcRoast [BD20] rely on MPC protocols for Legendre PRF and for higher-power residue characters;
- [FJR22b] (described in Chapter 5) proposed an MPC protocol optimized for the syndrome decoding problem in Hamming metric;
- [FMRV22b] (described in Chapter 6) proposed several MPC protocols for the subset sum problem;

- [FMRV22b] (described in Chapter 6) also proposed an MPC protocol for the [BHH01] PRF;
- [ARV22] proposed an MPC protocol for the MinRank problem;
- [Fen22] (described in Chapter 7) proposed two MPC protocols to check the rank of a matrix which can be used for the MinRank problem and the rank syndrome decoding problem;
- [Fen22] (described in Chapter 7) also proposed MPC protocols for the multivariate quadratic problems and the permuted kernel problem.

Chapter 4.

Shared Permutation for Syndrome Decoding

In 2020, the state of the art regarding the zero-knowledge proofs of knowledge for the syndrome decoding problem was sparse and not competitive (in terms of communication cost). In this chapter, we propose a first proof of knowledge using ideas from the MPC-in-the-Head paradigm. Thanks to the Fiat-Shamir transformation, we get a first signature scheme that outperforms the former code-based schemes based on the same assumption and which is competitive with some quantum-safe schemes.

The results presented in this chapter have been published in collaboration with Antoine Joux and Matthieu Rivain in the journal *Designs, Codes and Cryptography* [FJR23].

Contents

4.1. Introduction **36**
4.2. A Zero-Knowledge Protocol for Syndrome Decoding **37**
4.3. The Five-Round Zero-Knowledge Protocol **43**
4.4. The Signature Scheme **48**
4.5. Performance **51**
4.6. Comparison **54**
4.7. Conclusion **57**

4.1. Introduction

One of the few directions for post-quantum technologies is (error correcting) code-based cryptography. The *generic decoding* problem, or equivalently the *(computational) syndrome decoding* (SD) problem is a classic problem:

Definition 4.1.1 (Syndrome Decoding Problem)**.** *Let $\mathbb{F}$ be a finite field. Let m, k and w be positive integers such that $m > k$ and $m > w$. The syndrome decoding problem with parameters $(\mathbb{F}, m, k, w)$ is the following problem:*

> *Let H, x and y be such that:*
>
> 1. *H is uniformly sampled from $\mathbb{F}^{(m-k)\times m}$,*
> 2. *x is uniformly sampled from $\{x \in \mathbb{F}^m : \mathrm{wt}_H(x) = w\}$,*
> 3. *y is defined as $y := Hx$.*
>
> *From (H, y), find x.*

For random linear codes –*i.e.* for a random matrix H– this problem is widely believed to be robust for practical parameters. In this chapter, we only consider the binary version of the syndrome decoding problem, *i.e.* we only consider $\mathbb{F} := \mathbb{F}_2$.

In a pioneering work from three decades ago [Ste94], Stern proposed a zero-knowledge protocol to prove the knowledge of a solution to a syndrome decoding instance. This protocol achieves a *soundness error* of $2/3$ which means that a malicious prover can fool the verifier with probability $2/3$. Although an arbitrary security of $(2/3)^\tau$ can be achieved by repeating the protocol τ times, the induced communication cost is significant, which is partly due to this high soundness error. Since the work of Stern, a few papers have proposed optimizations and implementations (see for instance [Vér96; GG07; AGS11; ACBH13]) but for random linear codes with standard security levels, the communication cost is still heavy.

In this chapter, we propose a new zero-knowledge protocol for the SD problem which achieves a soundness error of $1/N$ with complexity $\mathcal{O}(N)$ for an *arbitrary* chosen N. In a nutshell, and as in Stern protocol, the solution x is masked by the application of a random permutation σ. However, instead of revealing either $\sigma(x)$ or σ, we always reveal $\sigma(x)$ and prove the existence of a permutation σ. To this purpose, we decompose σ into N masked permutations $\sigma(\cdot) + s := (\sigma_1(\cdot) + s_1) \circ \cdots \circ (\sigma_N(\cdot) + s_N)$ which are all committed by the prover and we let the verifier choose $N-1$ of them to be revealed. This way, we can maintain the privacy of σ while obtaining the desired soundness error of $1/N$.

Our construction requires the verifier to *trust* some of the variables sent by the prover. This can be ensured by a so-called *cut-and-choose* phase since these variables are independent of the secret solution x. While composing our technique with a cut-and-choose phase, the obtained protocol has a similar structure as the zero-knowledge proof for Boolean circuits proposed by Katz, Kolesnikov and Wang [KKW18] as an efficient instantiation of the *MPC-in-the-head* paradigm [IKOS07]. We can therefore apply the same optimizations (such as the merging of the cut-and-choose phase) to obtain a 5-round zero-knowledge protocol with arbitrary soundness error $2^{-\lambda}$. We further detail how to make our zero-knowledge proof non-interactive and turn it into a signature scheme by applying the Fiat-Shamir transform [FS87; AABN02]. For a 128-bit security level, the scheme achieves a signature size ranging between 17 KB (compact version) and 24 KB (fast version).

As explained in the thesis introduction, there are two different strategies to build code-based signatures: applying the Fiat-Shamir transform to an identification scheme or using hash-and-sign paradigm with a code-based trapdoor function. In the former case, standard techniques result in identification schemes with high communication costs (implying large signatures) because of the non-negligible soundness error. The Schnorr-Lyubashevsky approach [Sch90; Lyu09] is a promising way to mitigate this cost, however in the case of code-based signatures, it does not seem to work as well as for lattice-based signatures. In the rank metric, Durandal [ABG+19] is a code-based scheme that follows this approach. In the hash-and-sign paradigm, existing schemes based on trapdoor functions are more sensitive to structural attacks. Such a signature scheme named Wave [DST19] has been proposed in 2018 and is still secured against the current cryptanalysis state of the art, but it suffers a huge public key and a slow signing time. The signature described in this chapter has the advantage of being based on one of the oldest and hardest problems in code-based cryptography: syndrome decoding of random linear codes while still competing with existing schemes based on other (presumably weaker) problems in terms of public key and signature size.

We present the basic protocol (achieving $\frac{1}{N}$ soundness) in Section 4.2 and an optimized version (achieving arbitrary soundness) in Section 4.3. Then, we describe a signature scheme obtained through the Fiat-Shamir transform in Section 4.4. To conclude, we provide performance estimations for different sets of parameters in Section 4.5 and compare our construction with other zero-knowledge proofs and signature schemes from the state of the art in Section 4.6.

4.2. A Zero-Knowledge Protocol for Syndrome Decoding

Let us have an instance (H, y) of the syndrome decoding problem and let us denote x a solution of this instance (*i.e.* $y = Hx$ and $\mathrm{wt}_H(x) = w$). In what follows, we will describe a zero-knowledge proof of knowledge for this problem. More precisely, we will present a two-party protocol where a prover wants to convince a verifier that she knows a solution of the considered SD instance.

4.2.1. General Idea

As in the Stern protocol [Ste94], the prover first generates a random permutation σ and a random mask $r \in \mathrm{Ker}(H)$, and reveals

$$(v := \sigma(x)\ ,\ \tilde{x} := x + r\ ) \tag{4.1}$$

to the verifier. The verifier can then verify that

- $y = H\tilde{x}$ holds: this ensures that $\tilde{x} = x + r$ for some mask $r \in \mathrm{Ker}(H)$,
- $\mathrm{wt}_H(v) = w$ holds: this ensures that $x = \sigma^{-1}(v)$ is of weight w for any permutation σ.

In order to complete the proof, the prover then needs to convince the verifier that there exists a pair (σ, r) which *jointly* satisfies:

1. $Hr = 0$, and

2. $\sigma(\tilde{x}) = v + \sigma(r)$.

The first property is independent of the solution x which makes it provable using a cut-and-choose approach (see details in Section 4.2.4). Our key idea is a way to prove the second property while achieving an arbitrary soundness error $1/N$.

Our method introduces an affine transformation $A(\cdot) = \sigma(\cdot) + s$ for a random mask s so that we have:

$$A(\tilde{x}) = \sigma(\tilde{x}) + s = \sigma(x) + \underbrace{\sigma(r) + s}_{q} \ . \tag{4.2}$$

If the verifier trusts that the value q equals $A(r)$ for some $r \in \mathrm{Ker}(H)$, then she only needs to verify the equality $A(\tilde{x}) = v + q$ to be convinced. Since q is independent of x, ensuring a trustworthy q can also be achieved through cut-and-choose (see details in Section 4.2.4). This leaves us with the task of proving $A(\tilde{x}) = v + q$.

Let note $u := A(\tilde{x})$. If the prover sends u, the verifier can check $u = q + v$. To prove $A(\tilde{x}) = u$, we decompose A into a composition of N affine transformations $A_i(\cdot) = \sigma_i(\cdot) + s_i$ such that $A = A_N \circ \cdots \circ A_1$. The permutation σ and mask s are then defined as

$$\sigma = \sigma_N \circ \cdots \circ \sigma_1 \quad \text{and} \quad s = s_N + \sigma_N(\ldots + \sigma_2(s_1)) \ . \tag{4.3}$$

The main utility of this decomposition is that revealing all the A_i except one gives no information on A, provided that A_i is uniformly sampled for every i. In this setting, the prover first commits all the A_i and reveals

$$\begin{aligned} u_1 &:= A_1(u_0) \\ u_2 &:= A_2(u_1) \\ &\ldots \\ u_N &:= A_N(u_{N-1}) \end{aligned}$$

where $u_0 := \tilde{x}$ and $u_N = u$ by definition. Then, the verifier chooses a random i^* such that the prover reveals all the A_i except A_{i^*}. The verifier can then check $u_i = A_i(u_{i-1})$ for every $i \in [1 : N] \setminus \{i^*\}$. The only chance for the prover to cheat is to guess i^* in advance. Therefore, the maximum probability that a malicious prover can convince the verifier that $u = A(\tilde{x})$ is at most $1/N$.

To sum up,

1. The prover samples the random mask r from $\mathrm{Ker}(H)$ and N affine transformations $A_i(\cdot) := \sigma_i(\cdot) + s_i$. We denote $A(\cdot) := \sigma(\cdot) + s$ the composition $A_N \circ \cdots \circ A_1$, with σ and s satisfying Equation (4.3).

2. The prover reveals $q := \sigma(r) + s$, $v := \sigma(x)$ and $\tilde{x} := x + r$.

3. The verifier checks $y = H\tilde{x}$ and $\mathrm{wt}_H(v) = w$.

4. The prover reveals $u_i := A_i(u_{i-1})$ for $i \in \{1, ..., N\}$, with $u_0 := \tilde{x}$;

5. The verifier checks $u_N = q + v$.

6. The prover commits all the A_i, *i.e* all the (σ_i, s_i);

7. The verifier generates $i^* \leftarrow [1 : N]$ and sends it to the prover as challenge;

8. The prover reveals $\{A_i\}_{i \neq i^*}$.
9. The verifier checks $u_i = A_i(u_{i-1})$ for all $i \neq i^*$.

If all the checks have passed, the verifier deduces $u = A(\tilde{x})$. Moreover, if q is *trusted*, then a proof that $u = A(\tilde{x})$ constitutes a proof of knowledge of a solution x to the syndrome decoding instance. This protocol is a zero-knowledge protocol with a soundness error of $1/N$ under the trust hypothesis on q. As previously mentioned, this trust hypothesis can be fulfilled by using a cut-and-choose approach that we detail in Section 4.2.4.

4.2.2. Description of the Protocol

We give a formal description of our new zero-knowledge proof for syndrome decoding in Protocol 2. For the sake of simplicity, this description assumes that the value $q = \sigma(r) + s$ is *trusted* by the verifier. We denote this trust requirement with a star in superscript: $q^\star$.

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$x \in \mathbb{F}_2^m$ *s.t.* $\mathrm{wt}_H(x) = w$		
$H \in \mathbb{F}_2^{(m-k)\times m}$, $y := Hx$		H, $y := Hx$
For i in $[1:N]$:		
$\rho_i \leftarrow \{0,1\}^\lambda$		
$(\sigma_i, s_i) \leftarrow \mathcal{S}_m \times \mathbb{F}_2^m$		
$c_i = \mathsf{Com}((\sigma_i, s_i); \rho_i)$		
$r \leftarrow \mathrm{Ker}(H)$		
$\sigma = \sigma_N \circ \cdots \circ \sigma_1$		
$s = s_N + \sigma_N(\ldots + \sigma_2(s_1))$		
$q^\star = \sigma(r) + s$		
$v = \sigma(x)$		
$\tilde{x} = x + r$		
$u_0 = \tilde{x}$		
For i in $[1:N]$:		
$u_i = \sigma_i(u_{i-1}) + s_i$		
	$\xrightarrow{c_1,\ldots,c_N}$	
	$\xrightarrow{q^\star}$	
	$\xrightarrow{v,\tilde{x}}$	
	$\xrightarrow{u_1,\ldots,u_N}$	$i^* \leftarrow [1:N]$
	$\xleftarrow{i^*}$	
	$\xrightarrow{(\sigma_i,s_i,\rho_i)_{i\neq i^*}}$	$u_0 = \tilde{x}$
		For all $i \neq i^*$:
		Check $\mathsf{Verif}((\sigma_i, s_i), c_i, \rho_i) = 1$
		Check $u_i = \sigma_i(u_{i-1}) + s_i$
		Check $u_N = v + q^\star$
		Check $\mathrm{wt}_H(v) = w$
		Check $y = H\tilde{x}$
		Return Success

Protocol 2: Zero-knowledge proof for syndrome decoding – Simplified version with trusted $q^\star$.

4.2.3. Security Proofs

The following theorems state the completeness, zero-knowledge and soundness (with trusted q) of Protocol 2. We refer the reader to [FJR23] for the proofs of Theorems 4.2.2 and 4.2.3.

Theorem 4.2.1 (Completeness)**.** *Protocol 2 is perfectly complete, i.e. a prover $\mathcal{P}$ who knows a solution x to the syndrome decoding instance (H, y) and who follows the steps of the protocol always succeeds in convincing the verifier $\mathcal{V}$.*

Proof. For any sampling of the random coins of $\mathcal{P}$ and $\mathcal{V}$, if the computation described in Protocol 2 is genuinely performed then all the checks of $\mathcal{V}$ pass. □

Theorem 4.2.2 (Zero-Knowledge)**.** *Let the commitment scheme* Com *used in Protocol 2 be {computationally, statistically, perfectly} hiding. For all malicious PPT verifier $\tilde{\mathcal{V}}$, there exists a PPT simulator $\mathcal{S}$ which, given rewindable black-box access to $\tilde{\mathcal{V}}$, outputs a simulated transcript which is {computationally, statistically, perfectly} indistinguishable from a real transcript between $\mathcal{P}$ and $\tilde{\mathcal{V}}$.*

Theorem 4.2.3 (Soundness)**.** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input (H, y),*

- *builds q honestly, i.e. for any $\{(\sigma_i, s_i), \rho_i\}_i$ extractable from $\tilde{\mathcal{P}}$ and such that*

  $$\mathsf{Verif}((\sigma_i, s_i), c_i, \rho_i) = 1,$$

  there exists $r \in \mathrm{Ker}(H)$ such that $q = \sigma(r) + s$ with $\sigma = \sigma_N \circ \cdots \circ \sigma_1$ and $s = s_N + \sigma_N(\ldots + \sigma_2(s_1))$;

- *convinces the honest verifier $\mathcal{V}$ on input H, y to accept with probability*

  $$\tilde{\varepsilon} := \Pr[\langle \tilde{\mathcal{P}}, \mathcal{V} \rangle (H, y) \to 1] > \varepsilon$$

  where the soundness error ε is equal to $1/N$.

Then, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given rewindable black-box access to $\tilde{\mathcal{P}}$, produces with either a witness x such that $y = Hx$ and $\mathrm{wt}_H(x) = w$, or a commitment collision, by making in average

$$\frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{2 \cdot \ln(2)}{\tilde{\varepsilon} - \varepsilon}\right)$$

calls to $\tilde{\mathcal{P}}$.

4.2.4. Producing the Trusted Vector

In order to obtain a sound zero-knowledge proof, we need a procedure to build the trusted vector q. One possible technique is to use the *cut-and-choose* methodology. Concretely, the prover generates many different vectors q in a *verifiable* way, *i.e.*, by committing the randomness used for their generation. Then, the verifier will ask the prover to reveal how she built some of the vectors q, namely to *open* these vectors q. This opening consists in revealing the corresponding $r \in \mathrm{Ker}(H)$ as well as the (previously committed) randomness used to generate the affine transformation $A(\cdot)$. The verifier can then check the consistency of the commitments, the belonging of r to $\mathrm{Ker}(H)$ and the correct computation of $q = A(r)$. Since the prover's secret (the solution x to the syndrome decoding instance) is not involved, such an opening does not break the zero-knowledge property of the protocol. However the

opened vectors q become unusable for the protocol (because the used randomness is revealed). Thus, the prover must use one of the vectors q for which the building was not opened.

Let M be the number of generated and committed vectors. In the case of the simple protocol with soundness error $1/N$, one only needs a single trusted vector q. The verifier then asks for the opening of $M-1$ of the generated pairs before running the "trusted" part of the proof. If the prover cheated for more than one vector, then the verification will always fail. If the prover cheated on one vector, then the probability that the opening of this vector is not requested by the verifier during the cut-and-choose method is $1/M$.

The generation of a trusted vector q via the cut-and-choose method is described in Protocol 3.

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$H \in \mathbb{F}_2^{(m-k)\times m}$		H
For j in $[1:M]$:		
For i in $[1:N]$:		
$\rho_i^{[j]} \leftarrow \{0,1\}^\lambda$		
$(\sigma_i^{[j]}, s_i^{[j]}) \leftarrow \mathcal{S}_m \times \mathbb{F}_2^m$		
$c_i^{[j]} = \mathsf{Com}((\sigma_i^{[j]}, s_i^{[j]}); \rho_i^{[j]})$		
$r^{[j]} \leftarrow \mathrm{Ker}(H)$		
$\sigma^{[j]} = \sigma_N^{[j]} \circ \cdots \circ \sigma_1^{[j]}$		
$s^{[j]} = s_N^{[j]} + \sigma_N^{[j]}(\ldots + \sigma_2^{[j]}(s_1^{[j]}))$		
$q^{[j]} = \sigma^{[j]}(r^{[j]}) + s^{[j]}$		
	$\xrightarrow{\{c_i^{[j]}\}_{i\in[1:N], j\in[1:M]}}$	
	$\xrightarrow{\{q^{[j]}\}_{j\in[1:M]}}$	$j^* \leftarrow [1:M]$
	$\xleftarrow{j^*}$	
	$\xrightarrow{\left\{r^{[j]}, (\sigma_i^{[j]}, s_i^{[j]}, \rho_i^{[j]})_{i\in[1:N]}\right\}_{j\neq j^*}}$	
		For all $j \neq j^*$:
		For all i:
		Check $\mathsf{Verif}((\sigma_i^{[j]}, s_i^{[j]}), c_i^{[j]}, \rho_i^{[j]}) = 1$
		$\sigma^{[j]} = \sigma_N^{[j]} \circ \ldots \circ \sigma_1^{[j]}$
		$s^{[j]} = s_N^{[j]} + \sigma_N^{[j]}(\ldots)$
		Check $Hr^{[j]} = 0$
		Check $q^{[j]} = \sigma^{[j]}(r^{[j]}) + s^{[j]}$
		Return Success

Protocol 3: Cut-and-choose protocol to produce a trusted vector q.

This protocol can be composed with the sound phase (Protocol 2) to obtain a standalone 5-round protocol. This protocol is a zero-knowledge proof for the syndrome decoding problem with a soundness error of

$$\max\left\{\frac{1}{M}, \frac{1}{N}\right\}.$$

Indeed, the prover has two possible ways to cheat: either she cheats for one of the M vectors during the cut-and-choose phase, but if this vector is not selected the protocol is aborted (the cheating is discovered); or she can try to cheat during the sound phase. The prover can thus try to introduce a mistake in the first phase ($\frac{1}{M}$) or in the second phase ($\frac{1}{N}$), but not in both[1].

For completeness, we depict this standalone version in Protocol 4.

[1] If the malicious prover cheats on the first phase and that the cheating is not discovered, then she wins and she does not need to cheat on the second phase.

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$x \in \mathbb{F}_2^m$ *s.t.* $\mathrm{wt}_H(x) = w$		
$H \in \mathbb{F}_2^{(m-k)\times m}$, $y := Hx$		H, $y := Hx$
For $j \in [1:M]$		
For $i \in [1:N]$:		
$\rho_i^{[j]} \leftarrow \{0,1\}^\lambda$		
$(\sigma_i^{[j]}, s_i^{[j]}) \leftarrow \mathcal{S}_m \times \mathbb{F}_2^m$		
$c_i^{[j]} = \mathsf{Com}((\sigma_i^{[j]}, s_i^{[j]}); \rho_i^{[j]})$		
$r^{[j]} \leftarrow \mathrm{Ker}(H)$		
$\sigma^{[j]} = \sigma_N^{[j]} \circ \ldots \circ \sigma_1^{[j]}$		
$s^{[j]} = s_N^{[j]} + \sigma_N^{[j]}(\ldots + \sigma_2^{[j]}(s_1^{[j]}))$		
$q^{[j]} = \sigma^{[j]}(r^{[j]}) + s^{[j]}$		
	$\xrightarrow{\{c_i^{[j]}\}_{i\in[1:N], j\in[1:M]}}$	
	$\xrightarrow{\{q^{[j]}\}_{j\in[1:M]}}$	$j^* \leftarrow [1:M]$
	$\xleftarrow{j^*}$	
	$\xrightarrow{\left\{r^{[j]}, (\sigma_i^{[j]}, s_i^{[j]}, \rho_i^{[j]})_{i\in[1:N]}\right\}_{j\neq j^*}}$	For all $j \neq j^*$:
		$\sigma^{[j]} = \sigma_N^{[j]} \circ \ldots \circ \sigma_1^{[j]}$
		$s^{[j]} = s_N^{[j]} + \sigma_N^{[j]}(\ldots)$
		Check $Hr^{[j]} = 0$
		Check $q^{[j]} = \sigma^{[j]}(r^{[j]}) + s^{[j]}$
$v = \sigma^{[j^*]}(x)$		
$\tilde{x} = x + r^{[j^*]}$		
$u_0 = \tilde{x}$		
For $i \in [1:N]$:		
$u_i = \sigma_i(u_{i-1}) + s_i$		
	$\xrightarrow{v, \tilde{x}}$	
	$\xrightarrow{u_1, \ldots, u_N}$	$i^* \leftarrow [1:N]$
	$\xleftarrow{i^*}$	
	$\xrightarrow{(\sigma_i^{[j^*]}, s_i^{[j^*]}, \rho_i^{[j^*]})_{i\neq i^*}}$	$u_0 = \tilde{x}$
		For all $i \neq i^*$:
		Check $\mathsf{Verif}\left((\sigma_i^{[j^*]}, s_i^{[j^*]}), c_i^{[j^*]}, \rho_i^{[j^*]}\right) = 1$
		Check $u_i = \sigma_i^{[j^*]}(u_{i-1}) + s_i^{[j^*]}$
		Check $u_N = v + q^{[j^*]}$
		Check $\mathrm{wt}_H(v) = w$
		Check $H\tilde{x} = y$
		Return Success

Protocol 4: Standalone zero-knowledge proof for syndrome decoding

Remark 4.2.4. *Since the cut-and-choose does not require the prover secret, it can be executed before the prover gets its secret key. This step is hence a* preprocessing *phase of the sound protocol.*

Remark 4.2.5. *This idea of protocols requiring some trusted values has been formalized by [Beu20] as* protocols with helper*.*

4.3. The Five-Round Zero-Knowledge Protocol

In the previous section, we described a new zero-knowledge protocol for the syndrome decoding problem with a soundness error of $\max\{1/N, 1/M\}$ and complexity $\mathcal{O}(N \cdot M \cdot \mathsf{poly}(\lambda))$ for arbitrary chosen parameters N and M and with λ being the security level of the syndrome decoding instance. In order to obtain a soundness error close to $2^{-\lambda}$ for a target security parameter λ, one can run $\lambda/(\log_2 \min\{N, M\})$ independent repetitions of the protocol. However using such a simple approach is not optimal in terms of communication.

In this section, we present various optimizations to make our protocol efficient while targeting a standard security level. As a result, we describe a 5-round HVZK protocol that achieves $2^{-\lambda}$ soundness with optimized communication cost. For instance, for $\lambda = 128$, the size of the produced proof can be made lower than 15 KB.

4.3.1. Presentation of the Optimizations

We present here various optimizations to make our proof more compact. Our first improvement consists in merging the τ cut-and-choose phases (this was suggested in [KKW18] in a similar context). Then, we show how the pseudorandom generation of the precomputed values from compact seeds can save a lot of communication. This can be further improved by using pseudorandom generation trees for the seeds (as also suggested in [KKW18]). Finally, we show that some values can be recomputed by the verifier and can hence be trade for commitments in the prover messages. We further describe how to efficiently instantiate the commitment scheme using a collision-resistant hash function.

Merging of the cut-and-choose phase. As suggested in [KKW18], instead of repeating the cut-and-choose to obtain a trusted vector τ times, we can perform the cut-and-chose to generate τ trusted vectors all at once. Specifically, the prover $\mathcal{P}$ generates M vectors (for a large enough M) and the verifier requests the opening of $M - \tau$ of these vectors. After checking that the opened vectors have honestly been generated, the verifier trusts the remaining τ vectors which are then used for the τ independent executions of the sound phase.

Let us assume that a malicious prover correctly builds k vectors and cheats for $M - k$ vectors. Without loss of generality, k satisfies $k \geq M - \tau$, otherwise the cheating prover will have to open a dishonest vector and the proof will fail. Then, the probability of this malicious prover to successfully passing the merged cut-and-choose phase is at most $\binom{k}{M-\tau} \cdot \binom{M}{M-\tau}^{-1}$. Let $\gamma = M - k \leq \tau$ denote the number of dishonest vectors. Conditioned on passing the first phase, her probability of passing the τ executions of the protocol is at most

$$\underbrace{1 \times \ldots \times 1}_{\substack{\gamma \text{ times} \\ \text{(upper bound of the} \\ \text{success probability} \\ \text{for the cheating pairs)}}} \times \underbrace{\frac{1}{N} \times \ldots \times \frac{1}{N}}_{\substack{\tau-\gamma \text{ times} \\ \text{(success probability for the} \\ \text{unchecked but honest pairs)}}} = \frac{1}{N^{\tau-\gamma}} = \frac{1}{N^{k-M+\tau}} .$$

The soundness error is therefore

$$\varepsilon(M, N, \tau) := \max_{M-\tau \leq k \leq M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot N^{k-M+\tau}} \right\} .$$

Some examples of values for M, N and τ to reach a target security $\lambda \in \{128, 256\}$ are given in the Table 4.1.

Table 4.1.: Example values of M, N and τ to achieve statistical security $\lambda \in \{128, 256\}$.

	For 128-bit security						For 256-bit security					
N	4	8	16	32	64	128	4	8	16	32	64	128
M	256	293	512	606	842	1291	512	583	1024	1199	2144	3379
τ	64	43	32	26	22	19	128	86	64	52	43	37

Note: These parameters have been obtained while minimizing τ for a given N. Then given N and the underlying minimal τ, M is taken to be as small as possible. Other strategies provide different trade-offs.

Working with seeds and PRG. In the basic protocol, the variables σ_i and s_i are uniformly sampled at random for every i. This imposes the prover to reveal $N-1$ pairs (σ_i, s_i) in its final response which is expensive since s_i belongs to $\mathbb{F}_2^m$ and σ_i is a permutation of $[1:m]$. Instead, we can sample a random seed sseed_i and use a pseudo-random generator (PRG) to generate the pair (σ_i, s_i) from sseed_i. This way, whenever the prover wants to reveal (σ_i, s_i), she simply reveals seed_i. For a target security of λ bits, the seed only needs to make λ bits: we hence trade a message of $\log_2(m!) + m$ bits for one of λ bits. And to avoid revealing the commitment randomness ρ_i, the prover can derive it from another seed seed_i. But since the prover wants to reveal both ρ_i and sseed_i (to get the couple (σ_i, s_i)) when she opens the commitment, she can sample ρ_i and sseed_i from the same seed seed_i.

On the other hand, when a vector q is chosen by the verifier to be opened in the cut-and-choose phase, the prover must reveal (σ_i, s_i) for every $i \in [1:N]$ as well as r. In order to save further communication, all the $\{\mathsf{seed}_i\}_i$, as well as the random mask r, can be generated from a *master seed*: $(r, (\mathsf{seed}_i)_{i\in[1:N]}) \leftarrow \mathrm{PRG}(\mathsf{mseed})$. This way, when the vector q must be open, the prover sends mseed instead of $\left(r, (\mathsf{seed}_i)_{i\in[1:N]}\right)$ and replace $k + \lambda \cdot N$ bits of communication by λ bits.

Let $j \in \{1, ..., M\}$ denote the index of a precomputed set of variables during the cut-and-choose phase. In what follows, the master seed will be denoted $\mathsf{mseed}^{[j]}$, the N intermediary seeds will be denoted $\mathsf{seed}_i^{[j]}$ and the N subseeds will be denoted $\mathsf{sseed}_i^{[j]}$. Figure 4.1 illustrates the relation between the three types of seeds and the sampled variables.

For $j \in [1:M]$:
  $(r^{[j]}, (\mathsf{seed}_i^{[j]})_i) \leftarrow \mathrm{PRG}(\mathsf{mseed}^{[j]})$
  For $i \in [1:N]$:
    $\rho_i^j, \mathsf{sseed}_i^j \leftarrow \mathrm{PRG}(\mathsf{seed}_i^{[j]})$
    $(\sigma_i^{[j]}, s_i^{[j]}) \leftarrow \mathrm{PRG}(\mathsf{sseed}_i^{[j]})$

Figure 4.1.: Pseudorandom generation in the cut-and-choose phase.

Using GGM trees. The previous optimization enables to reduce the communication by replacing large variables (vectors of $\mathbb{F}_2^m$ and $[m] \to [m]$ permutations) by compact seeds. As suggested in [KKW18], we can go one step further by reducing the number of seeds that

must be revealed by the prover using the GGM construction as puncturable PRF.

In the above description, we generate N intermediary seeds $\mathsf{seed}_i^{[j]}$ from a master seed $\mathsf{mseed}^{[j]}$. If all the intermediary seeds must be opened, then the prover only reveals the master seed, but if the index j is chosen for the second phase then only $N-1$ of them must be revealed, which implies $(N-1)\cdot\lambda$ bits of communication. In order to avoid this factor $(N-1)$ we can use a GGM tree (see description in Section 2.2.2). The principle is to label the root of a binary tree of depth $\log_2 N$ with $\mathsf{mseed}^{[j]}$. Then, one inductively labels the children of each node with the output of a PRG applied to the node's label. The subseeds $(\mathsf{seed}_i^{[j]})_{i\in[1:N]}$ are defined as the labels of the N leaves of the tree. To reveal $(\mathsf{seed}_i^{[j]})_{i\neq i^*}$, it suffices to reveal the labels on the siblings of the path from the root of the tree to leaf i^*. Those labels allow the verifier to reconstruct $(\mathsf{seed}_i^{[j]})_{i\neq i^*}$ while still hiding $\mathsf{seed}_{i^*}^{[j]}$. Applying this optimization reduces the communication complexity to $\lambda\cdot\log_2 N$ for revealing the seeds $(\mathsf{seed}_i^{[j]})_{i\neq i^*}$. The same strategy can further be applied to the master seeds $(\mathsf{mseed}^{[j]})_{j\in[1:M]}$ in the cut-and-choose phase. By using a generation tree to generate the master seeds from a *grandmaster seed*, the communication cost of revealing $M-\tau$ master seeds out of M is decreased to $\lambda\cdot\tau\log_2\frac{M}{\tau}$ bits (instead of $\lambda\cdot(M-\tau)$ bits).

For the sake of simplicity we shall omit this optimization from the description of our protocol. However, we stress that it can be applied as is without impact on our security statements.

Keeping the bare minimum. In the basic protocol, the prover sends all the vectors $\{q^{[j]}\}$. However those are large m-bit vectors which can actually be recomputed by the verifier. Indeed,

- if j is among the opened indexes in the cut-and-choose phase, the verifier has access to the master seed $\mathsf{mseed}^{[j]}$. Thus, she can re-sample the values $r^{[j]}$, $(\sigma_i^{[j]}, s_i^{[j]})_i$ from which $q^{[j]}$ can be recomputed;
- if j is not among the opened indexes, then the verifier has the following relation: $u_N^{[j]} = v^{[j]} + q^{[j]}$, so that $q^{[j]}$ can be re-computed as $q^{[j]} = u_N^{[j]} - v^{[j]}$.

The verifier still needs to check that the prover knew the values of $\left\{q^{[j]}\right\}$ at the beginning of the interaction. That is, sending $\left\{q^{[j]}\right\}$ is replaced by sending commitments of $\left\{q^{[j]}\right\}$.

Another similar optimization can be applied in the second phase of the protocol. In the basic version, the prover sends all the u_i to the verifier but since the latter eventually knows $(\sigma_i, s_i)_{i\neq i^*}$, she can recompute all the u_i's, except u_{i^*}, thanks to the relation: $u_i = \sigma(u_{i-1}) + s_i$. Therefore, the prover only needs to send u_{i^*} to the verifier. Once again, such a modification implies that the u_i must be committed by the prover before receiving i^*.

Hash-based commitments. Since we strive at simplicity and efficiency, whenever possible we use a simple hash-based commitment scheme defined by $\mathsf{Com} : x \mapsto \mathrm{Hash}(x)$ and $\mathsf{Verif} : (x, c) \mapsto (c = \mathrm{Hash}(x))$. Such a scheme is known to be computationally binding under the collision-resistance of Hash, but not computationally hiding.

To keep the zero-knowledge property for the protocol, the commitments $c_i^{[j]}$ on the pairs $(\sigma_i^{[j]}, s_i^{[j]})$ must be hiding. For those, we must hence keep a hiding commitment scheme.[2]

[2] We might for instance use a computationally hiding hash-based commitment scheme defined as $\mathsf{Com} : (x, \rho) \mapsto \mathrm{Hash}(x \parallel \rho)$ for a long-enough random nonce ρ.

However for the other commitments, the inputs are not secret: the vector $q^{[j]}$ is committed with the corresponding $(c_i^{[j]})_i$ in a common hash-based commitment $h_j := \mathrm{Hash}(q^{[j]}, c_1^{[j]}, \ldots, c_N^{[j]})$. All these h_j commitments are further regrouped into a single hash value $h := \mathrm{Hash}(h_1, \ldots, h_M)$ which forms the initial commitment of the prover. During the second phase, after receiving the set $J \subseteq [M]$ of the τ trusted indexes as challenge from the verifier, the prover commits all the values $(u_i^{[j]})_{i\in[1:N],j\in J}$ into a single commitment $h' := \mathrm{Hash}((u_i^{[j]})_{i\in[1:N],j\in J})$.

4.3.2. Description of the Protocol

After applying the various optimizations described above, we obtain a 5-round HVZK protocol with much more compact computation which is depicted in Protocol 5. The protocol makes use of one commitment scheme Com and four hash functions Hash_1, Hash_2, Hash_3 and Hash_4, whose output ranges are assumed to be consistent with the protocol description.

4.3.3. Security Proofs

We prove hereafter that Protocol 5 achieves completeness, honest-verifier zero-knowledge, and $2^{-\lambda}$-soundness (for appropriately chosen parameter M, N and τ). We refer the reader to [FJR23] for the proofs of Theorems 4.3.2 and 4.3.3.

Theorem 4.3.1 (Completeness)**.** *Protocol 5 is perfectly complete.*

Proof. For any sampling of the random coins of $\mathcal{P}$ and $\mathcal{V}$, if the computation described in Protocol 5 is genuinely performed then all the checks of $\mathcal{V}$ pass. □

Theorem 4.3.2 (Honest-Verifier Zero-Knowledge)**.** *Let the PRG used in Protocol 5 be (t, ε_{PRG})-secure and the commitment scheme Com be (t, ε_{Com})-hiding. There exists an efficient simulator $\mathcal{S}$ which, given random challenges J and L outputs a transcript which is $(t, \varepsilon_{PRG} + \varepsilon_{Com})$-indistinguishable from a real transcript of Protocol 5.*

Theorem 4.3.3 (Soundness of Protocol 5)**.** *Let*

$$\varepsilon := \max_{M-\tau\le k\le M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot N^{k-M+\tau}} \right\} . \tag{4.4}$$

Suppose there is an efficient prover $\tilde{\mathcal{P}}$ such that

$$\tilde{\varepsilon} := \Pr[\langle \tilde{\mathcal{P}}, \mathcal{V}\rangle(H, y) \to 1] > \varepsilon , \tag{4.5}$$

where (H, y) is a random syndrome decoding instance. Then, there exists an extraction algorithm $\mathcal{E}$ which, given rewindable black-box access to $\tilde{\mathcal{P}}$, produces with either a witness x such that $y = Hx$ and $\mathrm{wt}_H(x) = w$, or a commitment collision, by making in average

$$\frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{8M}{\tilde{\varepsilon} - \varepsilon}\right)$$

calls to $\tilde{\mathcal{P}}$.

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$x \in \mathbb{F}_2^m$ *s.t.* $\mathrm{wt}_H(x) = w$ $H \in \mathbb{F}_2^{(m-k)\times m}$, $y := Hx$		H, $y := Hx$

$\mathsf{mseed}^{[0]} \leftarrow \{0,1\}^\lambda$
$(\mathsf{mseed}^{[j]})_{j\in[1:M]} \leftarrow \mathrm{PRG}(\mathsf{mseed}^{[0]})$
For $j \in [1:M]$:
  # $r^{[j]}$ *is sampled from* $\mathrm{Ker}(H)$
  $r^{[j]}, (\mathsf{seed}_i^{[j]})_i \leftarrow \mathrm{PRG}(\mathsf{mseed}^{[j]})$
  For $i \in [1:N]$:
  $\mathsf{sseed}_i^{[j]}, \rho_i^{[j]} \leftarrow \mathrm{PRG}(\mathsf{seed}_i^{[j]})$
  $c_i^{[j]} = \mathsf{Com}(\mathsf{sseed}_i^{[j]}; \rho_i^{[j]})$
  $(\sigma_i^{[j]}, s_i^{[j]}) \leftarrow \mathrm{PRG}(\mathsf{sseed}_i^{[j]})$
  $\sigma^{[j]} = \sigma_N^{[j]} \circ \ldots \circ \sigma_1^{[j]}$
  $s^{[j]} = s_N^{[j]} + \sigma_N^{[j]}(\ldots + \sigma_2^{[j]}(s_1^{[j]}))$
  $q^{[j]} = \sigma^{[j]}(r^{[j]}) + s^{[j]}$
  $h_j = \mathrm{Hash}_1(q^{[j]}, c_1^{[j]}, \ldots, c_N^{[j]})$
$h = \mathrm{Hash}_2(h_1, \ldots, h_M)$
	$\xrightarrow{h}$
	$\xleftarrow{J}$
		$J \leftarrow \{J \subset [M]\ ;\ \lvert J\rvert = \tau\}$

For $j \in J$:
  $v^{[j]} = \sigma^{[j]}(x)$
  $\tilde{x}^{[j]} = x + r^{[j]}$
  $u_0^{[j]} = \tilde{x}^{[j]}$
  For $i \in [1:N]$:
  $u_i^{[j]} = \sigma_i^{[j]}(u_{i-1}^{[j]}) + s_i^{[j]}$
  $h'_j = \mathrm{Hash}_3(u_1^{[j]}, \ldots, u_N^{[j]})$
$h' = \mathrm{Hash}_4((h'_j)_{j\in J})$
	$\xrightarrow{(\mathsf{mseed}^{[j]})_{j\in[1:M]\setminus J},\ h',\ (v^{[j]},\ \tilde{x}^{[j]})_{j\in J}}$
	$\xleftarrow{L}$
		$L = \{\ell_j\}_{j\in J} \leftarrow [1:N]^\tau$

$I_j = [1:N]\setminus\{\ell_j\}$	$\xrightarrow{\left((\mathsf{seed}_i^{[j]})_{i\in I_j},\ u_{\ell_j}^{[j]},\ c_{\ell_j}^{[j]}\right)_{j\in J}}$	

		For $j \in [1:M]\setminus J$:
		  $h_j \leftarrow \mathsf{mseed}^{[j]}$
		For $j \in J$:
		  $u_0^{[j]} = \tilde{x}^{[j]}$
		  For $i \in I_j$:
		  $\mathsf{sseed}_i^{[j]}, \rho_i^{[j]} \leftarrow \mathrm{PRG}(\mathsf{seed}_i^{[j]})$
		  $c_i^{[j]} = \mathsf{Com}(\mathsf{sseed}_i^{[j]}; \rho_i^{[j]})$
		  $(\sigma_i^{[j]}, s_i^{[j]}) \leftarrow \mathsf{sseed}_i^{[j]}$
		  $u_i^{[j]} = \sigma_i^{[j]}(u_{i-1}^{[j]}) + s_i^{[j]}$
		  $q^{[j]} = u_N^{[j]} - v^{[j]}$
		  $h_j = \mathrm{Hash}_1(q^{[j]}, c_1^{[j]}, \ldots, c_N^{[j]})$
		  $h'_j = \mathrm{Hash}_3(u_1^{[j]}, \ldots, u_N^{[j]})$
		  Check $y = H\tilde{x}^{[j]}$
		  Check $\mathrm{wt}_H(v^{[j]}) = w$
		Check $h = \mathrm{Hash}_2(h_1, \ldots, h_M)$
		Check $h' = \mathrm{Hash}_4((h'_j)_{j\in J})$
		Return Success

Protocol 5: Five-round HVZK proof for the syndrome decoding problem.

4.4. The Signature Scheme

In this section, we show how to turn our 5-round HVZK protocol into a signature scheme using the Fiat-Shamir transform [FS87; AABN02]. After explaining the transformation, we give the description of the signature scheme and then provide a security proof in the random oracle model (ROM).

4.4.1. Transformation into a Non-Interactive Scheme

With a straight application of Fiat-Shamir to Protocol 5, we would compute the challenges J and L as:

$$J := \mathrm{Hash}'_1(m, h)$$

and

$$L := \mathrm{Hash}'_2\left(m, h, (\mathsf{mseed}^{[j]})_{j\in[M]\setminus J}, (v^{[j]})_{j\in J}, (\tilde{x}^{[j]})_{j\in J}, h'\right) ,$$

where m is the input message and where Hash'_1 and Hash'_2 are some hash functions.

But doing so would imply an overhead on the size of the challenges J and L for the security to hold. Indeed, in [KZ20a], Kales and Zaverucha describe a forgery attack against signature schemes obtained by applying the Fiat-Shamir transform to 5-round protocols. Adapting this attack to our context yields a forgery cost of

$$\mathrm{cost}_{\mathrm{forge}} := \min_{M-\tau\le k\le M}\left\{\frac{\binom{M}{M-\tau}}{\binom{k}{M-\tau}} + N^{k-M+\tau}\right\}$$

which is substantially lower than the target forgery cost ε^{-1}, for ε being the soundness error of Protocol 5 (see Theorem 4.3.3).

A numerical analysis of the parameters shows that we get a more efficient signature scheme by turning Protocol 5 into a 3-round protocol before applying the Fiat-Shamir transform. Instead of waiting the challenge J from the verifier, the prover can directly commit $v^{[j]}$, $\tilde{x}^{[j]}$ and $\{u_i^{[j]}\}$ for all $j \in \{1, \dots M\}$. Thus the verifier can send both challenges J and L at the same time. For this purpose, the prover will compute

$$h'_j = \mathrm{Hash}(v^{[j]}, \tilde{x}^{[j]}, u_1^{[j]}, \dots, u_N^{[j]})$$

for all $j \in \{1, \dots, M\}$ and she will send $h' = \mathrm{Hash}(h'_1, \dots, h'_M)$ to the verifier. After receiving challenge (J, L), the prover sends h'_j for $j \notin J$ to enable the verifier to rebuild h'. In order to decrease the cost of sending all these h'_j, she can use a Merkle tree: after committing $h' = \mathrm{Merkle}(h'_1, \dots, h'_M)$, proving the consistence of $(h'_j)_{j\in J}$ with the hash root h' can be done by revealing at most $\tau \cdot \log_2\left(\frac{M}{\tau}\right)$ labels of the Merkle tree (instead of $M - \tau$ labels).

The resulting 3-round protocol is also an honest-verifier zero-knowledge protocol with the same soundness. It can indeed be checked that the described modification has essentially no impact on the proofs of Theorem 4.3.2 (honest verifier zero-knowledge) and Theorem 4.3.3 (soundness). While the communication cost is slightly greater for this 3-round version than for the original 5-round protocol, the transformation into a non-interactive scheme does not suffer the aforementioned attack, which allows much better parameters. Now the application of Fiat-Shamir applies to compute the challenges J and L as

$$(J, L) := \mathrm{Hash}'(m, h, h') ,$$

where Hash' is a hash function. Moreover since we have $h = \mathrm{Hash}_2(h_1, \ldots, h_M)$ and the h_j's are known by the verifier, we can directly compute the challenges J and L as:

$$(J, L) := \mathrm{Hash}'(m, h_1, \ldots, h_M, h') \ .$$

On the other hand, since we work in the random oracle model for the signature, we can replace the commitment scheme Com of the Protocol 5 by a single hash function Hash_0. We can then avoid sampling a commitment randomness and hence we can merge the seeds $\mathsf{seed}_i^{[j]}$ and $\mathsf{sseed}_i^{[j]}$.

Finally, to avoid that seed collisions produce the commitment collisions in distinct signatures, we add more entropy by using a *salt* as suggested in [Cha22].

4.4.2. Description of the Signature Scheme

In our signature scheme, the key generation algorithm randomly samples a syndrome decoding instance (H, y) of the syndrome decoding problem with solution x, with security parameter λ. In order to make the key pair compact, the matrix H is pseudorandomly generated from a λ-bit seed. Specifically, a call to the KeyGen algorithm outputs a pair $(pk, sk) := ((\mathsf{seed}_H, y), \mathsf{mseed})$ generated as follows:

1. $\mathsf{mseed} \leftarrow \{0,1\}^\lambda$
2. $(\mathsf{seed}_H, x) \leftarrow \mathrm{PRG}(\mathsf{mseed})$ where x is sampled in $\{x \in \mathbb{F}_2^m \mid \mathrm{wt}_H(x) = w\}$
3. $H \leftarrow \mathrm{PRG}(\mathsf{seed}_H)$
4. $y = Hx$; $pk = (\mathsf{seed}_H, y)$; $sk = \mathsf{mseed}$

For the sake of simplicity, we omit the re-generation of H and x from the seeds in the exposition below and assume $pk = (H, y)$ and $sk = (H, y, x)$.

The signature algorithm Given a secret key $sk = (H, y, x)$ and a message $m \in \{0,1\}^*$, the algorithm Sign proceeds as follows:

Step 0:

1. Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$.
2. Choose uniform $\mathsf{mseed}^{[0]} \in \{0,1\}^\lambda$.
3. Compute the seeds $\mathsf{mseed}^{[1]}, \ldots, \mathsf{mseed}^{[M]}$ with $\mathrm{TreePRG}(\mathsf{salt}, \mathsf{mseed}^{[0]})$.

Step 1: For each $j \in [M]$:

1. Use $\mathsf{mseed}^{[j]}$ to generate values $\mathsf{seed}_1^{[j]}, \ldots, \mathsf{seed}_N^{[j]}$ and $r^{[j]} \in \mathrm{Ker}(H)$ with $\mathrm{TreePRG}(\mathsf{salt}, \mathsf{mseed}^{[j]})$.
2. For $i \in [1:N]$, sample $\sigma_i^{[j]}, s_i^{[j]}$ using $\mathsf{seed}_i^{[j]}$ and compute $c_i^{[j]} := \mathrm{Hash}_0(\mathsf{salt}, j, i, \mathsf{seed}_i^{[j]})$.
3. (Cut-and-choose phase) Compute

$$\begin{aligned}
\sigma^{[j]} &:= \sigma_N^{[j]} \circ \cdots \circ \sigma_1^{[j]} \\
s^{[j]} &:= s_N^{[j]} + \sigma_N^{[j]}(\cdots + \sigma_2^{[j]}(s_1^{[j]})) \\
q^{[j]} &:= \sigma^{[j]}(r^{[j]}) + s^{[j]}
\end{aligned}$$

Chapter 4

4. (Sound phase) Compute

$$\begin{aligned} v^{[j]} &:= \sigma^{[j]}(x) \\ \tilde{x}^{[j]} &:= x + r^{[j]} \\ u_0^{[j]} &:= \tilde{x}^{[j]} \\ u_i^{[j]} &:= \sigma_i^{[j]}(u_{i-1}^{[j]}) + s_i^{[j]} \text{ for all } i \in [1:N] \end{aligned}$$

5. Compute $h_j := \mathrm{Hash}_1(q^{[j]}, c_1^{[j]}, \ldots, c_N^{[j]})$ and $h'_j := \mathrm{Hash}_2(v^{[j]}, \tilde{x}^{[j]}, (u_i^{[j]})_i)$.

Step 2: Compute

$$(J, L) := \mathrm{Hash}'(m, \mathsf{salt}, h_1, \ldots, h_M, h')$$

with $h' := \mathrm{Merkle}(h'_1, \ldots, h'_M)$, where $J \subset [1:M]$ is a set of size τ and L is a list $\{\ell_j\}_{j \in J}$ with $\ell_j \in [1:N]$. The signature includes (J, L) and salt.

Step 3: For each $j \in J$, the signer includes $v^{[j]}$, $\tilde{x}^{[j]}$, $c_{\ell_j}^{[j]}$, $u_{\ell_j}^{[j]}$ and $\mathsf{nodes}^{[j]} := \mathsf{nodes}(\mathsf{mseed}^{[j]}, [1:N] \backslash \{l_j\})$, where $\mathsf{nodes}(\cdot)$ returns the minimal nodes which enable to rebuild the tree leaves at input indices in the generation tree with input master seed (see Section 2.2.2). Also, the signer includes $\mathsf{nodes}^{\mathrm{M}} := \mathsf{nodes}(\mathsf{mseed}^{[0]}, [M] \backslash J)$ and $\mathsf{auth}^{\mathrm{Merkle}} := \mathsf{auth}((h'_1, \ldots, h'_M), J)$, where $\mathsf{auth}(\cdot)$ returns the Merkle nodes needed to open the paths for the leaves at indices $i \in J$ in the corresponding Merkle tree (see Section 2.2.3).

The verification algorithm Given a public key $pk = (H, y)$, a signature s and a message $m \in \{0,1\}^*$, the algorithm Verif proceeds as follows:

Step 0: Parse the signature s as

$$(\mathsf{salt}, J, L, \mathsf{nodes}^{\mathrm{M}}, \mathsf{auth}^{\mathrm{Merkle}}, \{v^{[j]}, \tilde{x}^{[j]}, \mathsf{nodes}^{[j]}, c_{\ell_j}^{[j]}, u_{\ell_j}^{[j]}\}_{j \in J}) \ .$$

Step 1: Use $\mathsf{nodes}^{\mathrm{M}}$ to rebuild $\mathsf{mseed}^{[j]}$ for each $j \notin J$. Then for every $j \notin J$, use $\mathsf{mseed}^{[j]}$ to compute h_j as in the signature algorithm.

Step 2: For every $j \in J$:

1. Use $\mathsf{nodes}^{[j]}$ to rebuild $\mathsf{seed}_i^{[j]}$ for each $i \neq \ell_j$.
2. For $i \neq \ell_j$, set $c_i^{[j]} := \mathrm{Hash}_0(\mathsf{salt}, j, i, \mathsf{seed}_i^{[j]})$ and get $\sigma_i^{[j]}, s_i^{[j]}$ using $\mathsf{seed}_i^{[j]}$.
3. Compute

$$\begin{aligned} u_i^{[j]} &= \sigma_i^{[j]}(u_{i-1}^{[j]}) + s_i^{[j]} \text{ for all } i \neq \ell_j \\ q^{[j]} &= u_N^{[j]} - v^{[j]} \end{aligned}$$

4. Compute $h_j := \mathrm{Hash}_1(q^{[j]}, c_1^{[j]}, \ldots, c_N^{[j]})$.

5. Compute $h'_j := \mathrm{Hash}_2(v^{[j]}, \tilde{x}^{[j]}, (u_i^{[j]})_i)$.
6. Check $H\tilde{x}^{[j]} = y$.
7. Check $\mathrm{wt}_H(v^{[j]}) = w$.

Step 3: Rebuild h' as $\mathrm{MerkleTree}(h'_1, \ldots, h'_M)$ using $\{h'_j\}_{j \in J}$ and $\mathsf{auth}^{\mathrm{Merkle}}$. Then check that (J, L) equals $\mathrm{Hash}'(m, \mathsf{salt}, h_1, \ldots, h_M, h')$.

4.4.3. Security Proof

We now state the security of our signature scheme in the following theorem. We refer the reader to [FJR23] for the proof.

Theorem 4.4.1. *Suppose the PRG used is (t, ϵ_{PRG})-secure and any adversary running in time t has at most an advantage ε_{SD} against the underlying syndrome decoding problem. Model* Hash_0, Hash_1, Hash_2 *and* Hash' *as random oracles where* Hash_0, Hash_1, Hash_2 *have 2λ-bit output length. Then any adaptive chosen-message adversary, running in time t, making q_s signing queries, and making q_0, q_1, q_2, q' queries, respectively, to the random oracles, succeeds in producing a valid forgery with probability upper bounded as*

$$p_{forge} \leq O(q_s \cdot \tau \cdot \varepsilon_{PRG}) + O\left(\frac{(q_0 + q_1 + q_2 + Mnq_s)^2}{2^\lambda}\right) + \epsilon_{SD} + q' \cdot \epsilon(M, N, \tau),$$

where

$$\varepsilon(M, N, \tau) := \max_{M-\tau \leq k \leq M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot N^{k-M+\tau}} \right\}.$$

4.5. Performance

4.5.1. Communication Cost and Signature Size

In the following analysis, we exclude the challenges from the communication cost since they are of very moderate impact and they do not appear in the signature. The communication then consists into

- $\textsc{Com} := h$,
- $\textsc{Res}_1 := ((\mathsf{mseed}^{[j]})_{j \notin J}, h', (v^{[j]}, \tilde{x}^{[j]})_{j \in J})$ and
- $\textsc{Res}_2 := ((\mathsf{seed}_i^{[j]})_{i \neq \ell_j},\ u_{\ell_j}^{[j]},\ c_{\ell_j}^{[j]})_{j \in J}$.

Whereas $u_{\ell_j}^{[j]}$ are full vectors of $\mathbb{F}_2^m$, $v^{[j]}$ are only vectors of weight w and $\tilde{x}^{[j]}$ are vectors from $\{\tilde{x} \mid H\tilde{x} = y\}$, which can be respectively encoded in $\log_2 \binom{m}{w}$ bits and k bits.

Thanks to the use of GGM trees, the communication cost for master seeds is at most $\tau \cdot \log_2 \frac{M}{\tau} \cdot \lambda$ bits (for $\tau \cdot \log_2 \frac{M}{\tau}$ intermediate tree seeds instead of $M - \tau$ leaf seeds) and the communication cost for subseeds is $\log_2(N) \cdot \lambda$ bits (for $\log_2(N)$ intermediate tree seeds instead of $N - 1$ leaf seeds).

So, the total communication cost (in bits) of the protocol is

$$\begin{aligned}\text{Cost} &= \text{Cost}_{\text{COM}} + \text{Cost}_{\text{RES}_1} + \text{Cost}_{\text{RES}_2} \\ &= 4\lambda + \lambda \cdot \tau \cdot \log_2 \frac{M}{\tau} + \tau \cdot \left[2\lambda + (m+k) + \log_2 \binom{m}{w} + \lambda \cdot \log_2(N)\right].\end{aligned}$$

The signature is composed of the same elements, except on two points:

- We need to add the sibling paths of $\{h'_j\}_{j \in J}$ in the Merkle tree to be able to rebuild h'. The communication cost for those paths is at most $(2\lambda) \cdot \tau \cdot \log_2 \frac{M}{\tau}$ bits (for $\tau \cdot \log_2 \frac{M}{\tau}$ intermediate tree labels).
- Since both challenges are merged, there is a single hash value to represent them. And using it and the other components in the signature, it is possible to deduce h and h'.

Thus the signature size (in bits) is

$$\text{Size} = 2\lambda + (3\lambda) \cdot \tau \cdot \log_2 \frac{M}{\tau} + \tau \cdot \left[2\lambda + (m+k) + \log_2 \binom{m}{w} + \lambda \cdot \log_2(N)\right].$$

4.5.2. Choice of the SD Parameters

In order to simplify the analysis, we place ourselves in a parameter range where the Hamming weight of the secret solution x is close but slightly below the Gilbert-Varshamov bound. This ensures the unicity of the solution with high probability while increasing the difficulty of finding solutions. Furthermore, this choice prevents the applicability of GBA methods. As a consequence, we only need to estimate the complexity of ISD algorithms. Previous studies of such parameters such as [TS16; BBC+19a] have argued that the algorithm of May, Meurer and Thomae [MMT11] is the most practical choice in the cryptographic range, as it outperforms algorithms with better asymptotics, e.g. the algorithm from [BJMM12]. Since the algorithm is quite sophisticated and involves several levels of recursion, we simplify the analysis by only considering the cost of the ISD loop, the size of lists at the top-level of their matching technique and the cost of merging these lists. More precisely, with m, k and w being given as input we choose two parameters ℓ and p and compute the following lower bound on the complexity of the attack:

$$\frac{\binom{m}{w}}{\binom{k+\ell}{p}\binom{m-k-\ell}{w-p}} \cdot \left(L + \frac{L^2}{2^{\ell-p}}\right) \quad \text{with} \quad L := \frac{\binom{k+\ell}{p/2}}{2^p}.$$

Optimizing for ℓ and p yields a slightly conservative estimate for the security level, which we used while choosing our parameters. Given these considerations, we suggest the following concrete parameters to achieve $\lambda \in \{128, 192, 256\}$ bits of security:

– for $\lambda = 128$: $(m = 1280\,,\quad k = m/2\,,\quad w = 132)$
– for $\lambda = 192$: $(m = 1920\,,\quad k = m/2\,,\quad w = 200)$
– for $\lambda = 256$: $(m = 2432\,,\quad k = m/2\,,\quad w = 258)$

4.5.3. Impact of the Other Parameters

We have fixed the SD parameters, we now need to choose the remaining parameters: N, M and τ. For a given N, the best proof size is achieved by minimizing τ, and then we take the minimal possible value for M. This is illustrated by Figure 4.2.

Figure 4.2 gives the proof size and the required parameter M w.r.t. the parameter N for 128-bit and 256-bit security. We observe that the value of M explodes when we try to achieve very compact signature by increasing N. Since M has a direct impact on the computation time, this parameter provides a trade-off between proof size and computation.

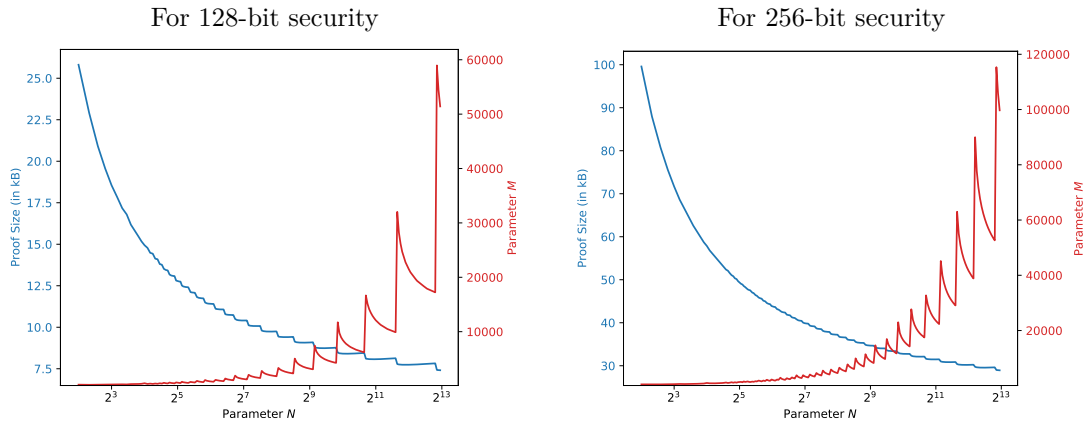


Figure 4.2.: Best proof size (and the used M) according to the parameter N.

We implemented the signature scheme in C. To sample the permutations, we choose the Fisher–Yates approach where we sample numbers by groups in order to minimize the pseudo-randomness consumption and the rejection rate. In our implementation, the pseudo-randomness is generated using AES in counter mode and the hash function is instantiated with SHAKE. We benchmarked our scheme on a 3.8 GHz Intel Core i7 CPU which supports AVX2 and AES instructions. All the reported timings were measured on this CPU while disabling Intel Turbo Boost.

We instantiate two trade-offs per security level: the first one lowering communication cost to produce short signatures, and the second one lowering computational cost to get a fast signature computation. We obtain the parameters and sizes described in Table 5.3. We provide the measured running times of our signature implementation in Table 5.4.

The overhead in the computation of our implementation is the uniform sampling (and its application) of the $M \cdot N$ permutations. We tried to implement them as efficiently as possible without introducing a bias using AVX instructions set, but it remains quite expensive. A way to improving the running times of the scheme would be to have a very efficient algorithm for random permutations (for example, by restricting the set in which the permutations are sampled).

Table 4.2.: Parameters (n, M, τ) with the achieved communication costs (in bytes).

		Parameters				Protocol	Signature	
λ	Aim	n	M	τ	$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{proof}\rvert$ (max)	$\lvert\mathsf{sgn}\rvert$ (max)	$\lvert\mathsf{sgn}\rvert$ (avg, std)
128	Fast	8	187	49	96	21 232	24 372	23 102, 196
128	Short	32	389	28	96	13 952	17 540	16 344, 236
192	Fast	8	283	73	144	47 400	54 396	51 635, 364
192	Short	32	578	42	144	31 344	39 396	36 639, 434
256	Fast	8	379	97	184	80 832	93 220	88 412, 559
256	Short	32	767	56	184	53 888	68 196	63 165, 655

Note: The average signature size and standard deviation has been measured over 10 000 experiments.

Table 4.3.: Benchmarks of our signature implementation.

λ	Aim	Keygen		Sign		Verify	
		Mcycles	ms	Mcycles	ms	Mcycles	ms
128	Fast	0.08	0.02	51	12.9	47	12.2
128	Short	0.08	0.02	249	62.3	221	56.6
192	Fast	0.14	0.04	133	33.9	125	32.5
192	Short	0.14	0.04	570	142.9	492	125.6
256	Fast	0.23	0.06	249	64.2	241	62.7
256	Short	0.22	0.06	1 029	259.4	896	229.9

Note: Timings are the averaged over 10 000 measurements. The CPU clock cycles have been measured using SUPERCOP (https://bench.cr.yp.to/supercop.html).

4.6. Comparison

In this section, we compare our scheme to different code-based and post-quantum signature schemes.

4.6.1. Comparison with Other Code-Based Signature Schemes

In the state of art, there exists two types of signatures. On one hand, there are schemes based on the Fiat-Shamir transform of identification schemes. However, classical approaches to produce identification schemes from code-based problems, like the famous Stern protocol, give large signatures because of the large soundness error of the underlying identification scheme (2/3 or 1/2). To avoid this issue, a solution consists in relying on different code-based problems. For instance, LESS is a recent scheme for which the security relies on the hardness of the Linear Code Equivalence problem [BMPS20; BBPS21]. Another direction is to find a way to adapt the Schnorr-Lyubashevsky approach to code-based cryptography. Durandal is

a recent scheme following this approach [ABG+19]. More recently, the authors of [GPS22] propose a zero-knowledge protocol with better soundness using a protocol with helper [Beu20] and relying on the syndrome decoding problem on larger fields. On the other hand, the hash-and-sign paradigm using trapdoors is also a popular way to derive signature schemes. Wave is such a recent code-based signature scheme in this paradigm [DST19]. However, such schemes are often more vulnerable to structural attacks.

Table 4.4.: Comparison of our scheme with signatures from the literature (128-bit security).

Scheme Name	Year	$\lvert\mathsf{sgn}\rvert$	$\lvert\mathsf{pk}\rvert$	t_{sgn}	t_{verif}
Stern	1993	62.5 KB	0.09 KB	-	-
Wave	2019	2.07 KB	3.2 MB	300 ms	-
Durandal - I	2018	3.97 KB	14.9 KB	4 ms	5 ms
Durandal - II	2018	4.90 KB	18.2 KB	5 ms	6 ms
LESS-FM - I	2020	15.2 KB	9.77 KB	-	-
LESS-FM - II	2020	5.25 KB	206 KB	-	-
LESS-FM - III	2020	10.39 KB	11.57 KB	-	-
[GPS22]-256	2021	22.2 KB	114 B	-	-
[GPS22]-1024	2021	19.5 KB	122 B	-	-
Our scheme (fast)	2021	22.6 KB	96 B	13 ms	12 ms
Our scheme (short)	2021	16.0 KB	96 B	62 ms	57 ms

Note: The performance of Stern protocol is computed for the same SD parameters as us. Reported timings are from the original publications: Wave has been benchmarked on a 3.5 Ghz Intel Xeon E3-1240 v5, while Durandal on a 2.8 Ghz Intel Core i5-7440HQ.

Our proposal is a signature scheme built from a zero-knowledge identification protocol with arbitrary soundness error. In Table 4.4, we compare the performance of our scheme with the current code-based signature state of the art, for the 128-bit security level.

Our scheme is comparable to Durandal, LESS and [GPS22] for the $\lvert\mathsf{sgn}\rvert + \lvert\mathsf{pk}\rvert$ metric, and much lighter than Wave which features heavy public keys. Regardless of the key size, Wave currently achieves the shortest signatures (but has a slow signing time). In terms of security, our scheme has the advantage of relying on the hardness of one of the oldest problem of the code-based cryptography: the syndrome decoding of random linear codes in Hamming weight metric. Previous schemes based on this problem are obtained from the Stern identification protocol (and its variants). As we observe from Table 4.4 signatures obtained with our scheme are three times shorter. [GPS22] also relies on the syndrome decoding problem. The technique they use gives a protocol for which the soundness error depends on the size of the base field of the syndrome decoding instance. For this reason, the underlying field must be large enough in order to achieve practical signature sizes. In contrast, the soundness error of our construction is independent of the base field and we can achieve practical signature sizes with the binary field. Let us remark that it would be possible to generalize our protocol to larger fields as well (we simply need to use isometries as in [GPS22] instead of permutations), but this would tend to increase the size of the obtained signatures.

4.6.2. Comparison with other Post-Quantum Signature Schemes

Finally, we compare in Table 5.6 our construction with some signature schemes aiming at post-quantum security and which are based from symmetric cryptography primitives (either based on hash tree or on the MPC-in-the-Head paradigm).

Table 4.5.: Comparison of our scheme with some post-quantum signature based on symmetric cryptography primitives.

	For 128-bit security			For 256-bit security		
Scheme Name	$\lvert\mathsf{sgn}\rvert$	t_{sgn}	t_{verif}	$\lvert\mathsf{sgn}\rvert$	t_{sgn}	t_{verif}
Our scheme (short)	16.0 K	62	57	61.7 K	259	230
Our scheme (fast)	22.6 K	13	12	86.3 K	64	62
SPHINCS$^+$ (short)	7.7 K	239	0.7	29.1 K	310	1.5
SPHINCS$^+$ (fast)	16.7 K	14	1.7	48.7 K	39	2.9
Picnic3	12.3 K	5.2	4.0	47.6 K	18	13
BBQ	31.6 K	-	-	133.7 K	-	-
Banquet (short)	13.0 K	44	40	52.8 K	191	175
Banquet (fast)	19.3 K	6	5	81.5 K	28	22

Note: The sizes are in bytes and the timings are in milliseconds. The benchmarks for the other schemes have been obtained on an Intel Xeon W-2133 CPU at 3.60GHz. the values for SPHINCS$^+$ and Banquet are extracted from the Banquet article [BDK+21b] and the values for Picnic3 are extracted from its original article [KZ20b].

All these schemes have short public and private keys (all under 100 bytes for 128-bit security), which is why we omit the key sizes in the comparison table. Compared to our scheme, Picnic3 [KZ20b] –which also relies on the KKW scheme [KKW18]– has better performance. On the other hand, our scheme is arguably more conservative in terms of security since Picnic is based on the hardness of inverting LowMC [ARS+15], a cipher with unconventional design choices, while our scheme is based on the hardness of the syndrome decoding problem on linear codes, which has a long cryptanalysis history and is believed to be very robust. BBQ [DDOS19] and Banquet [BDK+21b] are two other schemes based on the MPC-in-the-Head paradigm and for which the security is based on the hardness of inverting AES (instead of LowMC) which is a more conservative choice. Our signature has better performance than BBQ and is comparable to Banquet. In contrast, it is not competitive with SPHINCS$^+$ [BHK+19] which can achieve shorter signature sizes and very efficient verification. Let us stress that our implementation is a proof of concept which has not been deeply optimized and that some speed-up could probably be obtained by a thorough implementation study (in particular for the sampling and application of permutations). This issue is let open to further research.

4.7. Conclusion

In this chapter, we proposed a new zero-knowledge proof of knowledge for the syndrome decoding problem. In the latter, the prover decomposes the considered permutation σ into N sub-permutations $\sigma_1, \ldots, \sigma_N$ such that $\sigma = \sigma_N \circ \ldots \circ \sigma_1$ and reveals all these sub-permutations except a random one to the verifier.

The construction has been improved in [BG22]: the authors show that we can save a few kilobytes by re-arranging the computation. They also show that we can remove the helper when our goal is to build a proof of knowledge for a problem that has *some algebraic structure* (linearity, cyclicity, ...). For example, they applied their protocol to the case of the *permuted kernel problem* and achieved competitive performance: their optimized construction currently leads to the shortest signature relying on this problem (with sizes close to 9 KB).

In a more recent work, [BBP+23] shows how to design a new problem which leads to short communication costs with the existing proofs of knowledge. They propose the *Restricted Syndrome Decoding Problem* and achieve signature sizes of about 7-8 kilobytes for 128-bit security using optimized constructions of [BG22].

The technique of the "shared permutation" developed in this chapter leads to very efficient signature schemes as soon as the statement we want to prove in the underlying zero-knowledge protocol has a *great affinity with isometries* (like the permuted kernel problem or the restricted syndrome decoding problem). When this is not the case (as for the syndrome decoding problem in the Hamming metric or the rank metric), there will be more efficient schemes (see the next chapter for the case of the syndrome decoding problem in the Hamming metric).

Chapter 5.

Syndrome Decoding in the Head

At the end of 2021, the scheme proposed in the previous chapter led to the shortest code-based signatures based on the Fiat-Shamir transformation. However, it was still not competitive with some other post-quantum schemes (as SPHINCS$^+$). In this chapter, we propose a second zero-knowledge proof for the syndrome decoding problem. Instead of using permutations like most of the existing protocols, we rely on the MPC-in-the-head paradigm in which we reduce the task of proving the low Hamming weight of the SD solution to proving some relations between specific polynomials. Specifically, we propose a 5-round zero-knowledge protocol that proves the knowledge of a vector x such that $y = Hx$ and $\mathrm{wt}_H(x) \leq w$ and which achieves a soundness error close to $1/N$ for an arbitrary N. While turning this protocol into a non-interactive scheme, we get a signature scheme that outperforms all the former code-based signature schemes for the common "signature size + public key size" metric and which is competitive with SPHINCS$^+$, one of the future NIST post-quantum standards.

The results presented in this chapter have been published in collaboration with Antoine Joux and Matthieu Rivain in the proceedings of the international conference *Crypto 2022* [FJR22b].

Contents

5.1. Introduction **60**
5.2. A Zero-Knowledge Protocol for Syndrome Decoding **62**
5.3. The Signature Scheme **71**
5.4. Comparison **77**
5.5. Conclusion **79**

5.1. Introduction

The zero-knowledge proof proposed in the previous chapter has been inspired by the MPC-in-the-Head paradigm. But, even if it uses a sharing of the secret, it does not fit the MPCitH formalism presented in Chapter 3, in which we have an MPC protocol that checks that the shared secret is a pre-image of the corresponding one-way function (here, the syndrome decoding problem). In the previous chapter, the MPC protocol would only consist in applying the shared permutation in series.

In what follows, we build a new zero-knowledge protocol to prove the knowledge of a syndrome decoding solution using the MPCitH paradigm. We further turn this protocol into an efficient code-based signature scheme. In this chapter, we do not restrict ourselves to the binary field: $\mathbb{F}$ represents any finite field.

While proving that $y = Hx$ is communication-free in the MPC-in-the-Head paradigm, the hard part consists in proving that x is a small-weight vector. We propose here an efficient way to prove that $\mathrm{wt}_H(x) \leq w$ through a multi-party computation which is simulated by the prover ("in her head"). The key idea is to prove the equality $x \circ v = 0$ where $\circ$ is the component-wise multiplication and where the coefficients of the vector v are the evaluations of a polynomial Q of degree w. By definition, v has at most w zero coordinates, so the relation $x \circ v = 0$ proves that x has at most w non-zero coordinates (*i.e.* $\mathrm{wt}_H(x) \leq w$). The roots of the polynomial Q encode the non-zero positions of the vector x. In order to prove the relation $x \circ v = 0$, we use techniques borrowed from the Banquet signature scheme [BDK+21b] with further adaptations. To check that all $x_j \cdot v_j$ are equal to zero, we arrange the input x into a polynomial S, provide a product polynomial $F \cdot P$ as part of the witness, and check that $(F \cdot P)(\cdot)$ indeed equals the product of $S(\cdot)$ and $Q(\cdot)$. This can be done efficiently by only verifying a few products of these polynomials evaluated at some random points. However, instead of revealing the multiplication operands like in [BDK+21b], we rely on the product checking protocol proposed in [LN17; BN20] and its batch version recently introduced in [KZ22].

Let us note that the idea of encoding the non-zero positions in a polynomial to prove a Hamming weight inequality was already used in [DLO+18]. However, their zero-knowledge protocol relies on a linearly homomorphic commitment scheme, and such schemes do not have yet practical proposals in the post-quantum setting.

Thanks to the Fiat-Shamir transform [FS87], we convert our protocol into a signature scheme. Our scheme outperforms all the existing code-based signatures for the "signature size + public key size" metric. When relying on the hardness of the syndrome decoding problem over $\mathbb{F}_{256}$, our scheme is below 10 KB for this metric, which makes it competitive with Picnic3 [KZ20b] and SPHINCS$^+$ [BHK+19]. Compared to other code-based signature schemes (such as Wave [DST19] and Durandal [ABG+19]), our scheme has the significant advantage of relying on a non-structured decoding problem which has been widely studied over the last decades.

5.1.1. The d-split Syndrome Decoding Problem

To provide more flexibility, we introduce a parameter d in the definition of the syndrome decoding problem. The idea is, instead of having a constraint for the global weight of the secret vector x, to split x into d chunks $x := (x_1 \mid \ldots \mid x_d)$ and to have a constraint on the weight of each chunk. We call this alternative problem the *d-split syndrome decoding*

problem.

Definition 5.1.1 (d-Split Syndrome Decoding Problem)**.** *Let $\mathbb{F}$ be a finite field. Let m, k, w be positive integers such that $m > k$, $m > w$, $d \mid w$ and $d \mid m$. The d-split syndrome decoding problem with parameters $(\mathbb{F}, m, k, w)$ is the following problem:*

Let H, x and y be such that:

1. *H is uniformly sampled from $\mathbb{F}^{(m-k)\times m}$,*
2. *x is uniformly sampled from*

$$\left\{(x_1 \mid \ldots \mid x_d) \in \mathbb{F}^m \ : \ \forall i \in [1:d],\, x_i \in \mathbb{F}^{m/d},\, \mathrm{wt}_H(x_i) = \frac{w}{d}\right\} ,$$

3. *y is defined as $y := Hx$.*

From (H, y), find x.

By taking $d = 1$, we get the standard syndrome decoding problem. Let us note that the d-split syndrome decoding problem can be seen as a generalization of the *regular syndrome decoding* problem introduced by [AFS03], for which the ratio w/d is equal to 1. However, both problems (d-split and regular) do not aim to be used in the same way. In the d-split case, d is a small constant. When selecting parameters for it, we estimate the cost of the standard SD problem and the exact security loss. Thus, d-split SD problem can be considered as conservative as the standard problem ($d = 1$). In the regular case, using Theorem 5.1.2 would lead to a too large security loss. Instead, we select the parameters for it by relying on it *own* existing cryptanalysis.

The following theorem gives a way to estimate the difficulty of solving the d-split syndrome decoding problem. We provide a security reduction from this variant to the standard problem which allows us to compensate the security loss by a slight increase of the parameters. This so-called *d-split syndrome decoding problem* offers us more flexibility to find better size-performance trade-offs for our signature scheme.

Theorem 5.1.2. *Let $\mathbb{F}$ be a finite field. Let m, k, w be positive integers such that $m > k$, $m > w$, $d \mid w$ and $d \mid m$. Let $\mathcal{A}_d$ be an algorithm that solves a random $(\mathbb{F}, m, k, w)$-instance of the d-split syndrome decoding problem in time t with success probability ε_d. Then there exists an algorithm $\mathcal{A}_1$ which solves a random $(\mathbb{F}, m, k, w)$-instance of the standard syndrome decoding problem in time t with probability ε_1, where*

$$\varepsilon_1 \geq \frac{\binom{m/d}{w/d}^d}{\binom{m}{w}} \cdot \varepsilon_d \ .$$

Informally, an instance of the standard syndrome decoding problem is an instance of the d-split syndrome decoding problem with probability $\binom{m/d}{w/d}^d / \binom{m}{w}$. Moreover, a standard syndrome decoding instance can be "randomized" and input to the d-split adversary as much as desired. A formal proof of the above theorem is just after.

Proof. To prove the theorem, we build below an algorithm $\mathcal{A}_1$ to solve the traditional SD problem of parameters (m, k, w) using an algorithm $\mathcal{A}_d$ which solves the d-split SD problem with the same parameters.

Algorithm $\mathcal{A}_1$ (on input an SD instance (H, y)):

1. Sample a permutation σ of $\{1, \ldots, m\}$.
2. Permute the columns of H using σ to get $\hat{H}$.
3. Run $\mathcal{A}_d$ on input $(\hat{H}, y)$ to get $\hat{x}$.
4. If $\hat{x} = \perp$, return $\perp$.
5. Permute the coordinates of $\hat{x}$ using σ^{-1} to get x.
6. Return x.

The probability of transforming an SD instance into a d-split SD instance in Step 2 is $\binom{m/d}{w/d}^d / \binom{m}{w}$. Thus we have

$$\begin{aligned}
\varepsilon_1 &:= \Pr[\mathcal{A}_1(H, y) \neq \perp] \\
&\geq \Pr[\mathcal{A}_1(H, y) \neq \perp \cap (\hat{H}, y) \text{ is a } d\text{-split SD}] \\
&= \frac{\binom{m/d}{w/d}^d}{\binom{m}{w}} \cdot \Pr[\mathcal{A}_1(H, y) \neq \perp \mid (\hat{H}, y) \text{ is a } d\text{-split SD}] \\
&= \frac{\binom{m/d}{w/d}^d}{\binom{m}{w}} \cdot \Pr[\mathcal{A}_d(\hat{H}, y) \neq \perp \mid (\hat{H}, y) \text{ is a } d\text{-split SD}] \\
&= \frac{\binom{m/d}{w/d}^d}{\binom{m}{w}} \cdot \varepsilon_d
\end{aligned}$$

□

For example, let us assume we want a d-split parameter set with 128-bit security such that $\frac{m}{d} \leq 256$. We could take the parameters $(m, k, w) = (1536, 888, 120)$ with $d = 6$ over the binary field. We show in Section 5.3.4 that the standard syndrome decoding problem with the same parameters (but $d = 1$) has a security of 145 bits and, thanks to the Theorem 5.1.1, we know there is a security loss of at most 16 bits while switching to $d = 6$.

5.2. A Zero-Knowledge Protocol for Syndrome Decoding

Let us consider an instance (H, y) of the (d-split) syndrome decoding problem, and let us denote x a solution of this instance. We denote $\mathbb{F}_{\text{SD}}$ the field on which the instance is defined.

Without loss of generality, we assume that H is in the systematic form, *i.e.* that $H = (H' | I_{m-k})$ for some $H' \in \mathbb{F}_{\text{SD}}^{(m-k)\times k}$. Thus the solution x can be written as $(x_A | x_B)$ such that we have the linear relation

$$y = H' x_A + x_B \ . \tag{5.1}$$

This implies that one simply needs to send x_A ($k \cdot \log |\mathbb{F}_{\text{SD}}|$ bits) to reveal the solution of the instance (H, y).

In the following sections, we first build an MPC protocol that takes a sharing of $[\![x_A]\!]$, builds the corresponding $[\![x]\!]$ thanks to Equation (5.1), and checks that $[\![x]\!]$ corresponds to a vector with a Hamming weight of at most w/d on each chunk. Since $[\![x]\!]$ would satisfy $y = Hx$ by construction, this MPC protocol verifies that $[\![x_A]\!]$ corresponds to a solution

of the syndrome decoding instance (H, y). Then, in Section 5.2.3, we transform it into a zero-knowledge protocol which proves the knowledge of a solution of the syndrome decoding instance (H, y) thanks to the MPC-in-the-Head paradigm.

5.2.1. Standard Case ($d = 1$)

We first focus on the case where (H, y) is an instance of the standard syndrome decoding problem (*i.e.* we have $d = 1$). We will then show how to extend the protocol to the general case of any d. We consider a field extension $\mathbb{F}_{\text{poly}} \supseteq \mathbb{F}_{\text{SD}}$ such that $|\mathbb{F}_{\text{poly}}| \geq m$ (we recall that m is the length of the secret x, *i.e.* $x \in \mathbb{F}_{\text{SD}}^m$). We denote $\phi : \mathbb{F}_{\text{SD}} \to \mathbb{F}_{\text{poly}}$ the usual inclusion of $\mathbb{F}_{\text{SD}}$ into $\mathbb{F}_{\text{poly}}$. Let us take a bijection γ between $\{1, \ldots, |\mathbb{F}_{\text{poly}}|\}$ and $\mathbb{F}_{\text{poly}}$. Then, to ease the notation, we denote γ_i for $\gamma(i)$.

The protocol must check that $y = Hx$ and $\text{wt}_H(x) \leq w$. As explained in the introduction of the section, the input for the MPC protocol will be $[\![x_A]\!]$, then it will build the sharing $[\![x]\!]$ using the linear Equation (5.1). Thus we directly have that $y = Hx$. It remains to check that $\text{wt}_H(x) \leq w$.

To prove that $\text{wt}_H(x) \leq w$, the prover builds the three following polynomials:

- The polynomial $S \in \mathbb{F}_{\text{poly}}[X]$ satisfying

$$\forall i \in [1 : m],\ S(\gamma_i) = \phi(x_i)\ ,$$

  as well as $\deg S \leq m - 1$. This S is unique and can be computed by interpolation.

- The polynomial $Q \in \mathbb{F}_{\text{poly}}[X]$ defined as

$$Q(X) := \prod_{i \in E} (X - \gamma_i)$$

  for some $E \subset [1 : m]$ such that $|E| = w$ and $\{i \in [1 : m]\ :\ x_i \neq 0\} \subset E$, implying $\deg Q = w$.

- The polynomial $P \in \mathbb{F}_{\text{poly}}[X]$ defined as

$$P := (Q \cdot S)/F \quad \text{with} \quad F(X) := \prod_{i=1}^{m} (X - \gamma_i)\ .$$

We stress some useful properties of these polynomials:

- The polynomial Q is a monic polynomial of degree w. Moreover, for every $i \in [1 : m]$, we have

$$x_i \neq 0 \ \Rightarrow\ i \in E \ \Rightarrow\ Q(\gamma_i) = 0\ .$$

- The polynomial F divides $Q \cdot S$. Indeed, for every $i \in [1 : m]$, we have

$$(Q \cdot S)(\gamma_i) = 0$$

  since $S(\gamma_i) \neq 0 \ \Rightarrow\ x_i \neq 0 \ \Rightarrow\ Q(\gamma_i) = 0$. The polynomial P is hence well defined.

- The polynomial P has degree $\deg P \leq w - 1$.

If the prover convinces the verifier that there exists two polynomials P (with $\deg P \leq w-1$) and Q (with $\deg Q = w$) such that $Q \cdot S - P \cdot F = 0$ where S and F are built as described above, then the verifier can deduce the following:

$$\begin{aligned} &\forall i \in [1:m],\ (Q \cdot S)(\gamma_i) = P(\gamma_i) \cdot F(\gamma_i) = 0 \\ &\qquad\qquad \Rightarrow\ \forall i \in [1:m],\ Q(\gamma_i) = 0 \ \text{ or } \ S(\gamma_i) = \phi(x_i) = 0 \end{aligned}$$

Since Q has at most w roots, the verifier concludes that $\phi(x_i) \neq 0$ in at most w positions. Thus $\mathrm{wt}_H(x) \leq w$.

We now explain how to prove this statement in the MPCitH paradigm. For this purpose, we describe an MPC protocol, which on input x, P and Q outputs Accept if the above condition is verified and Reject otherwise, except with a small false positive probability. The parties' inputs are defined as the shares of $[\![x_A]\!]$, $[\![Q]\!]$ and $[\![P]\!]$. Let us recall that a sharing of a polynomial is naturally defined as a sharing of its coefficients (see Section 2.4.1). However, for the sharing of Q, we share all of its coefficients except the leading one. Indeed since Q is monic, its leading coefficient is publicly known and is equal to 1. Moreover, it enables to convince the verifier that Q is of degree *exactly* w, which is important since otherwise, a malicious prover could take Q as the zero polynomial.

From its inputs, the MPC protocol first builds the polynomial S from x_A. Then, to verify $Q \cdot S = P \cdot F$, it evaluates the two sides of the relation on t random points r_1, ..., r_t (sampled by the verifier in the MPCitH setting). If the relation is not verified, the probability to observe $Q(r_j) \cdot S(r_j) = P(r_j) \cdot F(r_j)$ for all $j \in [1:t]$ will be low, which stems from the Schwartz-Zippel Lemma (see Section 2.3.3). The larger the set from which the evaluation points r_j are sampled, the smaller the false positive probability p. For this reason, we take these evaluation points in a field extension $\mathbb{F}_{\text{points}}$ of $\mathbb{F}_{\text{poly}}$. Such a field extension allows us to have more points and so to detect more efficiently when $Q \cdot S \neq P \cdot F$. In practice, given an evaluation point r_j, the parties of the MPC protocol verify the relations $Q(r_j) \cdot S(r_j) = (P \cdot F)(r_j)$ by sacrificing multiplication triples as described in Section 3.4.1. To proceed, the prover must previously build t multiplication triples $([\![a_j]\!], [\![b_j]\!], [\![c_j]\!])$ for random elements $a_j, b_j, c_j \in \mathbb{F}_{\text{points}}$ satisfying $a_j \cdot b_j = c_j$ for $j \in [1:t]$ and include them to the parties' inputs (each party getting its corresponding share from $[\![a_j]\!]$, $[\![b_j]\!]$ and $[\![c_j]\!]$).

The MPC protocol runs as follows:

1. The parties sample t random points $r_1, \ldots, r_t$ of $\mathbb{F}_{\text{points}}$.

2. The parties locally compute $[\![x]\!]$ from $[\![x_A]\!]$ using Equation (5.1).

3. The parties locally compute $[\![S(r_j)]\!]$, $[\![Q(r_j)]\!]$ and $[\![(F \cdot P)(r_j)]\!]$ for all $j \in [1:t]$. Let us remark that $[\![S(r_j)]\!]$ can be computed from $[\![x]\!]$ by the parties without any interaction thanks to the linearity of Lagrange interpolation formula:

$$[\![S(r_j)]\!] = \sum_{i \in [1:m]} [\![x_i]\!] \prod_{\ell \in [1:m], \ell \neq i} \frac{r_j - \gamma_\ell}{\gamma_i - \gamma_\ell}\ .$$

   On the other hand $[\![(F \cdot P)(r_j)]\!]$ is computed as $F(r_j) \cdot [\![P(r_j)]\!]$ since F is publicly known.

4. For every $j \in [1:t]$, the parties run an MPC verification of the multiplication triple $([\![S(r_j)]\!], [\![Q(r_j)]\!], [\![(F \cdot P)(r_j)]\!])$ by sacrificing the triple $([\![a_j]\!], [\![b_j]\!], [\![c_j]\!])$:

- The parties sample a random $\varepsilon_j \in \mathbb{F}_{\text{points}}$.
- The parties locally set

$$[\![\alpha_j]\!] = \varepsilon_j \cdot [\![Q(r_j)]\!] + [\![a_j]\!] \quad \text{and} \quad [\![\beta_j]\!] = [\![S(r_j)]\!] + [\![b_j]\!].$$

- The parties broadcast $[\![\alpha_j]\!]$ and $[\![\beta_j]\!]$ to obtain α_j and β_j.
- The parties locally set

$$[\![v_j]\!] = \varepsilon_j \cdot [\![(F \cdot P)(r_j)]\!] - [\![c_j]\!] + \alpha_j \cdot [\![b_j]\!] + \beta_j \cdot [\![a_j]\!] - \alpha_j \cdot \beta_j \ .$$

- The parties broadcast $[\![v_j]\!]$ to obtain v_j.

5. The parties output Accept if $v = 0$ (*i.e.* if $v_j = 0$ for all j) and Reject otherwise.

Note that we do not need to specify how the random values r_j's and ε_j's are sampled by the parties since they will be provided as challenges from the verifier while turning to the zero-knowledge setting.

The above MPC protocol computes a non-deterministic function f which takes x, Q and P (and t multiplication triples) as input and which outputs Accept or Reject. The randomness of this function comes from the random evaluations points $r_1, \ldots, r_t$ and from the random challenges $\varepsilon_1, \ldots, \varepsilon_t$ used by the product checking protocol. Whenever x indeed satisfies $\mathrm{wt}_H(x) \leq w$ and the polynomials P and Q are genuinely computed as described above, the protocol outputs Accept with probability one. Whenever the protocol input is not of this form, the protocol shall output Reject except with a small false positive probability p. In other words, the output of the above protocol follows the distribution depicted in Table 3.1 where a good witness here means an x of weight at most w and polynomials P and Q which are correctly built.

Let us make explicit the false positive probability p. We shall denote $\Delta := |\mathbb{F}_{\text{points}}|$. Whenever the protocol input is not a good witness, *i.e.* $\mathrm{wt}_H(x) > w$, P or Q are not correctly built, we have $Q \cdot S \neq F \cdot P$. In the above protocol, both sides of the relation are evaluated in t random points. The probability to have the equality for i evaluation points among the t points is at most

$$\frac{\max_{\ell \leq m+w-1} \left\{ \binom{\ell}{i} \binom{\Delta-\ell}{t-i} \right\}}{\binom{\Delta}{t}}$$

since $Q \cdot S - F \cdot P$ is a polynomial of degree at most $m + w - 1$. This holds from a simple extension of the Schwartz-Zippel Lemma that we provide in Section 2.3.3. When this event occurs, the probability to obtain Accept as output is

$$\left(\frac{1}{\Delta}\right)^{t-i} ,$$

which corresponds to the probability to get the $t - i$ false positives in the verification of multiplication triples (for the $t - i$ remaining evaluation points r_j for which $Q(r_j) \cdot S(r_j) \neq F(r_j) \cdot P(r_j)$). Thus, the global false positive probability p satisfies

$$p \leq \sum_{i=0}^{t} \frac{\max_{\ell \leq m+w-1} \left\{ \binom{\ell}{i} \binom{\Delta-\ell}{t-i} \right\}}{\binom{\Delta}{t}} \left(\frac{1}{\Delta}\right)^{t-i} . \tag{5.2}$$

Chapter 5

5.2.2. General case (any d)

Let us now assume that (H, y) is an instance of a d-split syndrome decoding problem for some $d \geq 1$. We can easily adapt our protocol in that case. Instead of having a unique polynomial Q of degree w, we will have d polynomials $Q_1, \ldots, Q_d$ of degree exactly w/d to prove the weight bound $\mathrm{wt}_H(x_j) \leq w/d$ for each chunk x_j of the SD solution. We then have d polynomials S_j (of degree $m/d-1$) and d polynomials P_j (of degree $w/d-1$) satisfying the d relations $Q_j \cdot S_j = F \cdot P_j$ with $F := \prod_{j=1}^{m/d}(X - \gamma_j)$. To prove those d relations we evaluate each of them on t random points $r_1, \ldots, r_t$. We stress that the same t random points can be used for each chunk, *i.e.* for every $j \in [1 : d]$.

A malicious prover might try to cheat on a single relation (*i.e.* on a single chunk of the SD solution), in such a way that there exists $j_0 \in [1 : d]$ with

$$\begin{cases} Q_{j_0} \cdot S_{j_0} \neq F \cdot P_{j_0}, \\ \forall j \neq j_0,\ Q_j \cdot S_j = F \cdot P_j. \end{cases}$$

So for a given point r, we use the batched sacrificing-based checking of [KZ22] (described in Section 3.4.1) to check all the equalities $Q_j(r) \cdot S_j(r) = F(r) \cdot P_j(r)$ at once. This saves communication without impacting the soundness error compared to independent checks of the d relations.

Whenever the input x, $\{P_j\}$, $\{Q_j\}$ is not a good witness (*i.e.* whenever one x_j has a weight greater than w/d or one polynomial P_j or Q_j is not correctly built), at least one of the relations $Q_j \cdot S_j = F \cdot P_j$ is not verified. Since $Q_j \cdot S_j - F \cdot P_j$ is a polynomial of degree at most $(m+w)/d - 1$, the global false positive probability for the d-split variant becomes[1]

$$p \leq \sum_{i=0}^{t} \frac{\max_{\ell \leq (m+w)/d-1} \left\{ \binom{\ell}{i} \binom{\Delta-\ell}{t-i} \right\}}{\binom{\Delta}{t}} \left(\frac{1}{\Delta} \right)^{t-i} \tag{5.3}$$

with $\Delta := |\mathbb{F}_{\text{points}}|$. (This upper bound is equivalent to Equation (5.2) where the max degree $m + w - 1$ is replaced by $(m + w)/d - 1$).

The constraint on the size of $\mathbb{F}_{\text{poly}}$ now becomes

$$|\mathbb{F}_{\text{poly}}| \geq \frac{m}{d}$$

since we only need m/d points for the interpolation of the polynomials $S_1, \ldots S_d$. Thus using the d-split version allows us to use smaller fields for $\mathbb{F}_{\text{poly}}$ and $\mathbb{F}_{\text{points}}$.

Let us note that in practice the new communication is not smaller than before, but rather equivalent or higher, since we need to use bigger syndrome decoding instances to compensate the security loss of the d-split version. The main benefit to introduce the d-split version is to work on polynomials of smaller degree and/or on specific fields which provides better performance trade-offs (see Section 5.3.5).

5.2.3. Description of the Protocol

We now give the formal description of our zero-knowledge protocol (general case) in Protocol 6. For the sake of clarity in the protocol description, we denote $\vec{Q}$ the tuple of polynomials

[1] We here use the fact that the *optimal* strategy for a malicious prover is to build a bad witness such that only one polynomial relation is not satisfied. The resulting soundness error is thus the probability to fail to detect this cheating on this single relation.

$(Q_1, \ldots, Q_d)$. Same for the polynomials $\vec{P}$ and $\vec{S}$. The additions, substractions and polynomial evaluations of these tuples are component-wise defined. For example, for a point $r \in \mathbb{F}_{\text{points}}$, $\vec{Q}(r)$ means $(Q_1(r), \ldots, Q_d(r))$. We also use this vectorial notation for $\vec{a}_j$, $\vec{b}_j$, $\vec{\alpha}_j$, $\vec{\beta}_j$ and $\vec{\varepsilon}_j$ which shall represent vectors of $\mathbb{F}^d_{\text{points}}$. Let us recall that $\circ$ denotes the component-wise multiplication. In the scope of this protocol, the polynomial F is defined as $F(X) := \prod_{i=1}^{m/d}(X - \gamma_i)$ with $\mathbb{F}_{\text{poly}} = \{\gamma_1, \gamma_2, \ldots\}$.

5.2.4. Security Proofs

The following theorems state the completeness, zero-knowledge and soundness of Protocol 6. We refer the reader to [FJR22a] for the proofs of Theorems 5.2.2 and 5.2.3.

Theorem 5.2.1 (Completeness)**.** *Protocol 6 is perfectly complete, i.e. a prover $\mathcal{P}$ who knows a solution x to the syndrome decoding instance (H, y) and who follows the steps of the protocol always succeeds in convincing the verifier $\mathcal{V}$.*

Proof. For any sampling of the random coins of $\mathcal{P}$ and $\mathcal{V}$, if the computation described in Protocol 6 is genuinely performed then all the checks of $\mathcal{V}$ pass. □

Theorem 5.2.2 (Honest-Verifier Zero-Knowledge)**.** *Let the PRG used in Protocol 6 be (t, ε_{PRG})-secure and the commitment scheme* Com *be (t, ε_{Com})-hiding. There exists an efficient simulator $\mathcal{S}$ which, given a random challenge i^* outputs a transcript which is $(t, \varepsilon_{PRG} + \varepsilon_{Com})$-indistinguishable from a real transcript of Protocol 6.*

Theorem 5.2.3 (Soundness)**.** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input (H, y), convinces the honest verifier $\mathcal{V}$ on input H, y to accept with probability*

$$\tilde{\varepsilon} := \Pr[\langle \tilde{\mathcal{P}}, \mathcal{V} \rangle (H, y) \to 1] > \varepsilon$$

where the soundness error ε is equal to

$$p + \frac{1}{N} - p \cdot \frac{1}{N}$$

with p defined in Equation (5.3). Then, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given rewindable black-box access to $\tilde{\mathcal{P}}$, produces with either a witness $x := (x_1 \mid \ldots \mid x_d)$ such that $y = Hx$ and $\forall j, \mathrm{wt}_H(x_j) \leq w/d$, or a commitment collision, by making an average number of calls to $\tilde{\mathcal{P}}$ which is upper bounded by

$$\frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{2 \cdot \ln(2)}{\tilde{\varepsilon} - \varepsilon}\right).$$

By adapting the parameters t and Δ, we can produce a protocol with soundness error arbitrarily close to $1/N$.

5.2.5. Performance

In the following analysis, we exclude the challenges from the communication cost since they are of very moderate impact (and do not count whenever making the protocol non-interactive). The communication then consists into

- $\textsc{Com} := h$,

Inputs: Both parties have $H = (H'|I_{m-k}) \in \mathbb{F}_{\mathrm{SD}}^{(m-k)\times m}$ and $y \in \mathbb{F}_{\mathrm{SD}}^{m-k}$, the prover also holds $x := (x_1 \mid x_2 \mid \ldots \mid x_d) \in \mathbb{F}_{\mathrm{SD}}^{m}$ such that $y = Hx$ and $\mathrm{wt}_H(x_j) \leq w/d$ for $j \in [1:d]$.

Round 1: The prover computes the proof witness: for all chunk $j \in [1:d]$,

1. Choose a set $E_j \subset [1 : \frac{m}{d}]$ s.t. $|E_j| = \frac{w}{d}$ and $\{\ell : (x_j)_\ell \neq 0\} \subset E_j$.
2. Compute $Q_j(X) = \prod_{\ell \in E_j}(X - \gamma_\ell) \in \mathbb{F}_{\mathrm{poly}}[X]$.
3. Compute $S_j(X) \in \mathbb{F}_{\mathrm{poly}}[X]$ by interpolation *s.t.* $\deg S_j \leq \frac{m}{d} - 1$ and $\forall \ell \in [1 : \frac{m}{d}], S_j(\gamma_\ell) = (x_j)_\ell$.
4. Compute $P_j(X) = S_j(X)Q_j(X)/F(X) \in \mathbb{F}_{\mathrm{poly}}[X]$.

Then, the prover prepares the inputs for the multi-party computation as follows:

1. Sample a root seed: $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\lambda$.
2. Compute parties' seeds and commitment randomness $(\mathsf{seed}_i, \rho_i)_{i\in[1:N]}$ with TreePRG(seed).
3. For each party $i \in \{1, \ldots, N\}$,
   - $[\![\vec{a}_j]\!]_i, [\![\vec{b}_j]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$, for each $j \in [1:t]$
   - If $i \neq N$,
     - $\{[\![c_j]\!]_i\}_{j\in[1:t]}, [\![x_A]\!]_i, [\![\vec{Q}]\!]_i, [\![\vec{P}]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$
     - $\mathsf{state}_i = \mathsf{seed}_i$
   - Else,
     - $[\![x_A]\!]_N = x_A - \sum_{\ell\neq N}[\![x_A]\!]_\ell$
     - $[\![\vec{Q}]\!]_N = \vec{Q} - \sum_{\ell\neq N}[\![\vec{Q}]\!]_\ell$.
     - $[\![\vec{P}]\!]_N = \vec{P} - \sum_{\ell\neq N}[\![\vec{P}]\!]_\ell$.
     - $[\![c_j]\!]_N = \langle \vec{a}_j, \vec{b}_j\rangle - \sum_{\ell\neq N}[\![c_j]\!]_\ell$, for each $j \in [1:t]$
     - $\mathsf{help} = ([\![x_A]\!]_N, [\![\vec{Q}]\!]_N, [\![\vec{P}]\!]_N, \{[\![c_j]\!]_N\}_{j\in[1:t]})$
     - $\mathsf{state}_N = \mathsf{seed}_N \mid\mid \mathsf{help}$
   - Commit the party's state: $\mathsf{com}_i = \mathsf{Com}(\mathsf{state}_i; \rho_i)$.

The prover builds $h = \mathrm{Hash}(\mathsf{com}_1, \ldots, \mathsf{com}_N)$ and sends it to the verifier.

Round 2: The verifier uniformly samples, for each $j \in [1:t]$, an evaluation point $r_j \leftarrow \mathbb{F}_{\mathrm{points}}$ and a vector $\vec{\varepsilon}_j \leftarrow \mathbb{F}_{\mathrm{points}}^d$, and sends them to the prover.

Round 3: The prover simulates the MPC protocol:

1. The parties locally set $[\![x_B]\!] = y - H'[\![x_A]\!]$.
2. The parties locally compute $[\![\vec{S}]\!]$ by interpolation using $[\![x]\!] := ([\![x_A]\!] \mid [\![x_B]\!])$.
3. Then for all $j \in [1:t]$,
   - The parties locally compute $[\![\vec{S}(r_j)]\!]$, $[\![\vec{Q}(r_j)]\!]$ and $[\![\vec{P}(r_j)]\!]$.
   - They locally set $[\![\vec{\alpha}_j]\!] = \vec{\varepsilon}_j \circ [\![\vec{Q}(r_j)]\!] + [\![\vec{a}_j]\!]$.
   - They locally set $[\![\vec{\beta}_j]\!] = [\![\vec{S}(r_j)]\!] + [\![\vec{b}_j]\!]$.
   - The parties open $[\![\vec{\alpha}_j]\!]$ and $[\![\vec{\beta}_j]\!]$ to get $\vec{\alpha}_j$ and $\vec{\beta}_j$.
   - The parties locally set

$$[\![v_j]\!] = -[\![c_j]\!] + \langle \vec{\varepsilon}_j, F(r_j) \cdot [\![\vec{P}(r_j)]\!]\rangle + \langle \vec{\alpha}_j, [\![\vec{b}_j]\!]\rangle + \langle \vec{\beta}_j, [\![\vec{a}_j]\!]\rangle - \langle \vec{\alpha}_j, \vec{\beta}_j\rangle \ .$$

The prover builds $h' = \mathrm{Hash}([\![\vec{\alpha}_1]\!], [\![\vec{\beta}_1]\!], [\![v_1]\!], \ldots, [\![\vec{\alpha}_t]\!], [\![\vec{\beta}_t]\!], [\![v_t]\!])$ and sends it to the verifier.

Round 4: The verifier uniformly samples $i^* \leftarrow [1:N]$ and sends it to the prover.

Round 5: The prover sends $(\mathsf{state}_i, \rho_i)_{i\neq i^*}$, com_{i^*}, $\{[\![\vec{\alpha}_j]\!]_{i^*}\}_{j\in[1:t]}$ and $\{[\![\vec{\beta}_j]\!]_{i^*}\}_{j\in[1:t]}$.

Verification: The verifier accepts iff all the following checks succeed:

1. For each $i \neq i^*$, she computes all the commitments to the parties' states: $\mathsf{com}_i = \mathsf{Com}(\mathsf{state}_i; \rho_i)$. Then she checks that $h \stackrel{?}{=} \mathrm{Hash}(\mathsf{com}_1, \ldots, \mathsf{com}_N)$.
2. Using $\{\mathsf{state}_i\}_{i\neq i^*}$, she simulates all the parties except for i^*. From the recomputed shares, she checks that $h' \stackrel{?}{=} \mathrm{Hash}([\![\vec{\alpha}_1]\!], [\![\vec{\beta}_1]\!], [\![v_1]\!], \ldots, [\![\vec{\alpha}_t]\!], [\![\vec{\beta}_t]\!], [\![v_t]\!])$ where $[\![v_j]\!]_{i^*} := -\sum_{i\neq i^*}[\![v_j]\!]_i$.

Protocol 6: Zero-knowledge proof for syndrome decoding.

- $\textsc{Res}_1 := h'$ and
- $\textsc{Res}_2 := ((\mathsf{state}_i, \rho_i)_{i \neq i^*},\ \mathsf{com}_{i^*},\ \{[\![\vec{\alpha}_j]\!]_{i^*}\}_{j \in [1:t]},\ \{[\![\vec{\beta}_j]\!]_{i^*}\}_{j \in [1:t]})$.

For $i \neq N$, state_i simply consists in a seed of λ bits. For $i = N$, state_i contains

- a seed of λ bits,
- the share $[\![x_A]\!]_N$ of a plaintext,
- the shares $[\![\vec{Q}]\!]_N$ and $[\![\vec{P}]\!]_N$ which are $2 \cdot d$ polynomials of degree $w/d - 1$,
- and the shares $\{[\![c_j]\!]_N\}_{j \in [1:t]}$ of t points of $\mathbb{F}_{\text{points}}$.

Let us recall that seeds are sampled using a GGM tree (as defined in Section 2.2.2). Instead of sending the $N-1$ seeds and commitment randomness of $(\mathsf{state}_i, \rho_i)_{i \neq i^*}$, we can instead send the sibling path from $(\mathsf{state}_{i^*}, \rho_{i^*})$ to the tree root[2], it costs at most $\lambda \cdot \lceil \log_2(N) \rceil$ bits (we need to reveal $\lceil \log_2(N) \rceil$ nodes of the tree). Moreover com_{i^*} is a commitment of 2λ bits, and $\{[\![\vec{\alpha}_j]\!]_{i^*}\}_{j \in [1:t]}, \{[\![\vec{\beta}_j]\!]_{i^*}\}_{j \in [1:t]}$ are elements of $\mathbb{F}_{\text{points}}$. The communication cost (in bits) of the protocol is then

$$\textsc{Size} = 4\lambda + \underbrace{k \cdot \log_2 |\mathbb{F}_{\text{SD}}|}_{[\![x_A]\!]_N} + \underbrace{(2 \cdot w) \cdot \log_2 |\mathbb{F}_{\text{poly}}|}_{[\![\vec{Q}]\!]_N, [\![\vec{P}]\!]_N} + \underbrace{(2 \cdot d + 1) \cdot t \cdot \log_2 |\mathbb{F}_{\text{points}}|}_{\{[\![\vec{\alpha}_j]\!]_{i^*}, [\![\vec{\beta}_j]\!]_{i^*}, [\![c_j]\!]_N\}_{j \in [1:t]}} + \underbrace{\lambda \cdot \log_2(N)}_{(\mathsf{seed}_i)_{i \neq i^*}} + \underbrace{2\lambda}_{\mathsf{com}_{i^*}}$$

As usual, to achieve a targeted soundness error $2^{-\lambda}$, we can perform τ parallel repetitions of the protocol such that $\varepsilon^\tau \leq 2^{-\lambda}$. And instead of sending τ values for h and h', we can merge them together to send a single h and a single h'. The communication cost (in bits) of the protocol with τ repetitions is

$$\textsc{Size} = 4\lambda + \tau \cdot \Big(k \cdot \log_2 |\mathbb{F}_{\text{SD}}| + (2 \cdot w) \cdot \log_2 |\mathbb{F}_{\text{poly}}| + (2 \cdot d + 1) \cdot t \cdot \log_2 |\mathbb{F}_{\text{points}}| + \lambda \cdot \log_2(N) + 2\lambda \Big)$$

and the obtained soundness error is

$$\left(p + \frac{1}{n} - p \cdot \frac{1}{n} \right)^\tau .$$

5.2.6. Comparison

We compare our new protocol with existing zero-knowledge protocols for syndrome decoding (or equivalently for *message decoding*). We compare these protocols on two SD instances of 128-bit security:

- Instance 1 (from Chapter 4): Syndrome Decoding on $\mathbb{F}_2$ with parameters

$$(m, k, w) = (1280, 640, 132);$$

[2] As in Chapter 4, the seed seed_i of a party and the commitment randomness ρ_i are derived from the same seed, which corresponds to a leaf of the seed tree.

- Instance 2 [CVE11]: Syndrome Decoding on $\mathbb{F}_{2^8}$ with parameters

$$(m, k, w) = (208, 104, 78).$$

The comparison for a soundness error of 2^{-128} is given in the Table 5.1. For our protocol, we provide two instantiations for each syndrome decoding instance to give the reader an idea of the obtained performance while changing the number of parties. The first instantiation called "short" corresponds to an instantiation which provides small communication cost. The second one called "fast" corresponds to an instantiation with faster computation but higher communication cost. The used parameters $(N, \tau, |\mathbb{F}_{\text{poly}}|, |\mathbb{F}_{\text{points}}|, t)$ for our scheme are

- Instance 1:

  Short: $(N, \tau, |\mathbb{F}_{\text{poly}}|, |\mathbb{F}_{\text{points}}|, t) = (256, 16, 2^{11}, 2^{22}, 2) \;\Rightarrow\; \varepsilon^{\tau} = 2^{-128.0}$

  Fast: $(N, \tau, |\mathbb{F}_{\text{poly}}|, |\mathbb{F}_{\text{points}}|, t) = (32, 26, 2^{11}, 2^{22}, 1) \;\Rightarrow\; \varepsilon^{\tau} = 2^{-129.6}$

- Instance 2:

  Short: $(N, \tau, |\mathbb{F}_{\text{poly}}|, |\mathbb{F}_{\text{points}}|, t) = (256, 16, 2^{8}, 2^{24}, 2) \;\Rightarrow\; \varepsilon^{\tau} = 2^{-128.0}$

  Fast: $(N, \tau, |\mathbb{F}_{\text{poly}}|, |\mathbb{F}_{\text{points}}|, t) = (32, 26, 2^{8}, 2^{24}, 1) \;\Rightarrow\; \varepsilon^{\tau} = 2^{-130.0}$

Table 5.1.: Comparison of our protocol with state-of-the-art zero-knowledge protocols for syndrome decoding.

Protocol Name	Year	Instance 1	Instance 2	Proved statement
[Ste94]	1993	37.4 KB	46.1 KB	$y = Hx$, $\text{wt}_H(x) = w$
[Vér96]	1997	31.7 KB	38.7 KB	*message decoding*
[CVE11]	2010	-	37.4 KB	$y = Hx$, $\text{wt}_H(x) = w$
[AGS11]	2011	24.8 KB	-	$y = Hx$, $\text{wt}_H(x) = w$
[GPS22] (short)	2021	-	15.2 KB	$y = Hx$, $\text{wt}_H(x) = w$
[GPS22] (fast)	2021	-	19.9 KB	$y = Hx$, $\text{wt}_H(x) = w$
Chapter 4 (short)	2021	12.9 KB	15.6 KB	$y = Hx$, $\text{wt}_H(x) = w$
Chapter 4 (fast)	2021	20.0 KB	24.7 KB	$y = Hx$, $\text{wt}_H(x) = w$
This chapter (short)	2022	9.7 KB	6.9 KB	$y = Hx$, $\text{wt}_H(x) \leq w$
This chapter (fast)	2022	14.4 KB	9.7 KB	$y = Hx$, $\text{wt}_H(x) \leq w$

Note: The formulae for the communication costs of the different protocols and the used parameters are detailed in [FJR22b, Appendix B].

We can remark that all the previous protocols prove an equality for the Hamming weight by relying on isometries (*i.e.* permutations if $\mathbb{F}_{\text{SD}} = \mathbb{F}_2$). On our side, we only prove the inequality $\text{wt}_H(w) \leq w$. We stress that both versions (equality or inequality) can be merely equivalent for some SD parameters. Indeed, if w is chosen sufficiently below the Gilbert-Varshamov bound and if we know there exists an SD solution x of Hamming weight w, then proving the knowledge of a solution x' with $\text{wt}_H(x') \leq w$ amounts to proving the knowledge of x with overwhelming probability.

5.3. The Signature Scheme

In this section, we show how to turn our 5-round HVZK protocol into a signature scheme using the Fiat-Shamir transform [FS87; AABN02]. After explaining the transformation, we give the description of the signature scheme and then provide a security proof in the random oracle model (ROM).

5.3.1. Transformation into a Non-Interactive Scheme

To transform our protocol into a non-interactive scheme, we apply the multi-round variant of the Fiat-Shamir transform [FS87] (see *e.g.* [EDV+12; CHR+16]). Concretely, we compute the challenge CH_1 and CH_2 as

$$\begin{aligned} h_1 &= \text{Hash}_1(m, \mathsf{salt}, h) \\ \text{CH}_1 &\leftarrow \text{PRG}(h_1) \end{aligned}$$

and

$$\begin{aligned} h_2 &= \text{Hash}_2(m, \mathsf{salt}, h, h') \\ \text{CH}_2 &\leftarrow \text{PRG}(h_2) \end{aligned}$$

where m is the input message, where Hash_1 and Hash_2 are some hash functions (that shall be modeled as random oracles) and where h and h' are the Round 1 and Round 3 hash commitments merged for the τ repetitions. We introduce a value salt called *salt* which is sampled from $\{0,1\}^{2\lambda}$ at the beginning of the signing process. This value is then used for each commitment to the parties' states. Without it, the security of the signature would be at most $2^{\lambda/2}$ because of the seed collisions between several signatures. Moreover, since the signature security relies on the random oracle model, we can safely replace the commitment scheme Com of Protocol 6 by a single hash function Hash_0.

The security of the obtained scheme is lower than the soundness error of Protocol 6. Indeed, in [KZ20a], Kales and Zaverucha describe a forgery attack against signature schemes obtained by applying the Fiat-Shamir transform to 5-round protocols. Adapting this attack to our context yields a forgery cost of

$$\text{cost}_{\text{forge}} := \min_{\tau_1,\tau_2:\tau_1+\tau_2=\tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau-i}} + N^{\tau_2} \right\} \tag{5.4}$$

with p defined in Equation (5.3). This is substantially lower than the target forgery cost of $1/\varepsilon$, for ε being the soundness error of Protocol 6 (see Theorem 5.2.3). We therefore need to adapt the parameters to fill this gap.

5.3.2. Description of the Signature Scheme

In our signature scheme, the key generation algorithm randomly samples an instance (H, y) of the d-split syndrome decoding problem with solution x, with security parameter λ. In order to make the key pair compact, the matrix H is pseudorandomly generated from a λ-bit seed. Specifically, a call to the KeyGen algorithm outputs a pair $(pk, sk) := ((\mathsf{seed}_H, y), \mathsf{mseed})$ generated as follows:

1. $\mathsf{mseed} \leftarrow \{0,1\}^\lambda$

2. $(\mathsf{seed}_H, x) \leftarrow \mathrm{PRG}(\mathsf{mseed})$ where x is sampled in $\{x \in \mathbb{F}_2^m \mid \mathrm{wt}_H(x) = w\}$
3. $H \leftarrow \mathrm{PRG}(\mathsf{seed}_H)$
4. $y = Hx$; $pk = (\mathsf{seed}_H, y)$; $sk = \mathsf{mseed}$

For the sake of simplicity, we omit the re-generation of H and x from the seeds in the algorithms below and assume $pk = (H, y)$ and $sk = (H, y, x)$.

Given a secret key $sk = (H, y, x)$ and a message $m \in \{0,1\}^*$, the algorithm Sign proceeds as described in Figure 5.1. And given a public key $pk = (H, y)$, a signature σ and a message $m \in \{0,1\}^*$, the algorithm Verif proceeds as described in Figure 5.2. For the sake of clarity, as for the protocol description in Section 5.2.3, we use the vectorial notation to represent a tuple of d polynomials or of d points.

5.3.3. Signature Properties

We now state the security of our signature scheme in the following theorem.

Theorem 5.3.1. *Suppose the PRG used is (t, ϵ_{PRG})-secure and any adversary running in time t has at most an advantage ε_{SD} against the underlying d-split syndrome decoding problem. Model* Hash_0, Hash_1 *and* Hash_2 *as random oracles where* Hash_0, Hash_1 *and* Hash_2 *have 2λ-bit output length. Then chosen-message adversary against the signature scheme depicted in Figure 5.1, running in time t, making q_s signing queries, and making q_0, q_1, q_2 queries, respectively, to the random oracles, succeeds in outputting a valid forgery with probability*

$$\Pr[\mathsf{Forge}] \leq \epsilon_{SD} + \varepsilon_{PRG} + \frac{(\tau N + 2)Q^2}{2^{2\lambda}} + \Pr[X + Y = \tau],$$

with

- $Q = q_0 + q_1 + q_2 + q_s \cdot (2 + \tau(2N - 1))$,
- $X = \max_{i \in [1:q_1]}\{X_i\}$ *with* $X_i \sim \mathcal{B}(\tau, p)$, *and*
- $Y = \max_{i \in [1:q_2]}\{Y_i\}$ *with* $Y_i \sim \mathcal{B}(\tau - X, \frac{1}{N})$,

where p is defined in Equation (5.3) and $\mathcal{B}(n_0, p_0)$ denotes the binomial distribution with n_0 the number of trials and p_0 the success probability of each trial.

5.3.4. Parameters

In what follows, we propose three parameter sets which achieve a security level of 128 bits for the signature:

- the first one shall rely on the hardness to solve the SD problem on $\mathbb{F}_2$;
- the second one shall also rely on the hardness to solve the SD problem on $\mathbb{F}_2$, but we shall use a d-split version to get polynomials over a chosen field, concretely $\mathbb{F}_{256}$;
- the last one shall rely on the hardness to solve the SD problem on $\mathbb{F}_{256}$.

Inputs: A secret key $sk = (H, y, x)$ and a message $m \in \{0,1\}^*$.

Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$.

Phase 1.0: Building of the proof witness. For all chunk $j \in [1:d]$,

1. Compute $Q_j(X) = \prod_{\ell \in E_j}(X - \gamma_\ell) \in \mathbb{F}_{\text{poly}}[X]$ where $E_j = \{\ell : (x_j)_\ell \neq 0\}$.
2. Compute $S_j(X) \in \mathbb{F}_{\text{poly}}[X]$ by interpolation *s.t.* $\deg S_j \leq \frac{m}{d} - 1$ and $\forall \ell \in [1 : \frac{m}{d}], S_j(\gamma_\ell) = (x_j)_\ell$.
3. Compute $P_j(X) = S_j(X)Q_j(X)/F(X) \in \mathbb{F}_{\text{poly}}[X]$.

Phase 1.1: Preparation of the MPC-in-the-Head inputs. For each iteration $e \in [1:\tau]$,

1. Sample a root seed: $\mathsf{seed}^{[e]} \xleftarrow{\$} \{0,1\}^\lambda$.
2. Compute parties' seeds $\mathsf{seed}_1^{[e]}, \ldots, \mathsf{seed}_N^{[e]}$ with $\text{TreePRG}(\mathsf{salt}, \mathsf{seed})$.
3. For each party $i \in \{1, \ldots, N\}$,
   - $[\![\vec{a}_j^{[e]}]\!]_i, [\![\vec{b}_j^{[e]}]\!]_i \leftarrow \text{PRG}(\mathsf{salt}, \mathsf{seed}_i^{[e]})$, for each $j \in [1:t]$
   - If $i \neq N$,
     - $\{[\![c_j^{[e]}]\!]_i\}_{j \in [1:t]}, [\![x_A^{[e]}]\!]_i, [\![\vec{Q}^{[e]}]\!]_i, [\![\vec{P}^{[e]}]\!]_i \leftarrow \text{PRG}(\mathsf{salt}, \mathsf{seed}_i^{[e]})$
     - $\mathsf{state}_i^{[e]} = \mathsf{seed}_i^{[e]}$
   - Else,
     - $[\![x_A^{[e]}]\!]_N = x_A - \sum_{j \neq N} [\![x_A^{[e]}]\!]_j$
     - $[\![\vec{Q}^{[e]}]\!]_N = \vec{Q} - \sum_{\ell \neq N} [\![\vec{Q}^{[e]}]\!]_\ell$.
     - $[\![\vec{P}^{[e]}]\!]_N = \vec{P} - \sum_{\ell \neq N} [\![\vec{P}^{[e]}]\!]_\ell$.
     - $[\![c_j^{[e]}]\!]_N = \langle \vec{a}_j^{[e]}, \vec{b}_j^{[e]} \rangle - \sum_{\ell \neq N} [\![c_j^{[e]}]\!]_\ell$, for each $j \in [1:t]$
     - $\mathsf{help}^{[e]} = ([\![x_A^{[e]}]\!]_N, [\![\vec{Q}^{[e]}]\!]_N, [\![\vec{P}^{[e]}]\!]_N, \{[\![c_j^{[e]}]\!]_N\}_{j \in [1:t]})$
     - $\mathsf{state}_N^{[e]} = \mathsf{seed}_N^{[e]} \mid\mid \mathsf{help}^{[e]}$
   - Compute $\mathsf{com}_i^{[e]} = \text{Hash}_0(\mathsf{salt}, e, i, \mathsf{state}_i^{[e]})$.

Phase 2: First challenge (randomness for the MPC protocol).

1. Compute $h_1 = \text{Hash}_1(m, \mathsf{salt}, \mathsf{com}_1^{[1]}, \mathsf{com}_2^{[1]}, \ldots, \mathsf{com}_{N-1}^{[\tau]}, \mathsf{com}_N^{[\tau]})$.
2. Extend hash $\{r_j^{[e]}, \vec{\varepsilon}_j^{[e]}\}_{e \in [1:\tau], j \in [1:t]} \leftarrow \text{PRG}(h_1)$ where $r_j^{[e]} \in \mathbb{F}_{\text{points}}$ and $\vec{\varepsilon}_j^{[e]} \in \mathbb{F}_{\text{points}}^d$.

Phase 3: Simulation of the MPC protocol. For each iteration $e \in [1:\tau]$,

1. The parties locally set $[\![x_B^{[e]}]\!] = y - H'[\![x_A^{[e]}]\!]$.
2. Then for all $j \in [1:t]$,
   - The parties locally compute $[\![\vec{S}^{[e]}]\!]$ by interpolation using $[\![x^{[e]}]\!] := ([\![x_A^{[e]}]\!] \mid [\![x_B^{[e]}]\!])$.
   - They locally compute $[\![\vec{S}^{[e]}(r_j^{[e]})]\!]$, $[\![\vec{Q}^{[e]}(r_j^{[e]})]\!]$ and $[\![\vec{P}^{[e]}(r_j^{[e]})]\!]$.
   - They locally set $[\![\vec{\alpha}_j^{[e]}]\!] = \vec{\varepsilon}_j^{[e]} \circ [\![\vec{Q}^{[e]}(r_j^{[e]})]\!] + [\![\vec{a}_j^{[e]}]\!]$.
   - They locally set $[\![\vec{\beta}_j^{[e]}]\!] = [\![\vec{S}^{[e]}(r_j^{[e]})]\!] + [\![\vec{b}_j^{[e]}]\!]$.
   - The parties open $[\![\vec{\alpha}_j^{[e]}]\!]$ and $[\![\vec{\beta}_j^{[e]}]\!]$ to get $\vec{\alpha}_j^{[e]}$ and $\vec{\beta}_j^{[e]}$.
   - The parties locally set

$$[\![v_j^{[e]}]\!] = -[\![c_j^{[e]}]\!] + \langle \vec{\varepsilon}_j^{[e]}, F(r_j^{[e]}) \cdot [\![\vec{P}^{[e]}(r_j^{[e]})]\!] \rangle + \langle \vec{\alpha}_j^{[e]}, [\![\vec{b}_j^{[e]}]\!] \rangle + \langle \vec{\beta}_j^{[e]}, [\![\vec{a}_j^{[e]}]\!] \rangle - \langle \vec{\alpha}_j^{[e]}, \vec{\beta}_j^{[e]} \rangle \ .$$

Phase 4: Second challenge (parties to be opened).

1. Compute $h_2 = \text{Hash}_2(m, \mathsf{salt}, h_1, \{[\![\vec{\alpha}_j^{[e]}]\!], [\![\vec{\beta}_j^{[e]}]\!], [\![v_j^{[e]}]\!]\}_{j \in [1:t], e \in [1:\tau]})$.
2. Expand hash $\{i^{*[e]}\}_{e \in [1:\tau]} \leftarrow \text{PRG}(h_2)$ where $i^{*[e]} \in [1:N]$.

Phase 5: Building of the signature. Output the signature σ built as

$$\mathsf{salt} \mid h_1 \mid h_2 \mid \left((\mathsf{state}_i^{[e]})_{i \neq i^{*[e]}} \mid \mathsf{com}_{i^{*[e]}}^{[e]} \mid \{[\![\vec{\alpha}_j^{[e]}]\!]_{i^{*[e]}}\}_{j \in [1:t]} \mid \{[\![\vec{\beta}_j^{[e]}]\!]_{i^{*[e]}}\}_{j \in [1:t]} \right)_{e \in [1:\tau]} \ .$$

Figure 5.1.: Code-based signature scheme - Signing algorithm.

Inputs: A public key $pk = (H, y)$, a signature σ and a message $m \in \{0,1\}^*$.

1. Parse the signature σ as

$$\mathsf{salt} \mid h_1 \mid h_2 \mid \left((\mathsf{state}_i^{[e]})_{i \neq i^{*[e]}} \mid \mathsf{com}_{i^{*[e]}}^{[e]} \mid \{ [\![\overrightarrow{\alpha^{[e]}}_j]\!]_{i^{*[e]}} \}_{j \in [1:t]} \mid \{ [\![\overrightarrow{\beta^{[e]}}_j]\!]_{i^{*[e]}} \}_{j \in [1:t]} \right)_{e \in [1:\tau]} .$$

2. Extend hash $\{r_j^{[e]}, \vec{\varepsilon}_j^{[e]}\}_{e \in [1:\tau], j \in [1:t]} \leftarrow \mathrm{PRG}(h_1)$ where $r_j^{[e]} \in \mathbb{F}_{\mathrm{points}}$ and $\vec{\varepsilon}_j^{[e]} \in \mathbb{F}_{\mathrm{points}}^d$.
3. Extend hash $\{i^{*[e]}\}_{e \in [1:\tau]} \leftarrow \mathrm{PRG}(h_2)$ where $i^{*[e]} \in [1:N]$.
4. For each iteration $e \in [1:\tau]$,
   - For each $i \neq i^{*[e]}$, computes $\mathsf{com}_i^{[e]} = \mathrm{Hash}_0(\mathsf{salt}, e, i, \mathsf{state}_i^{[e]})$.
   - Using $\{\mathsf{state}_i^{[e]}\}_{i \neq i^{*[e]}}$, simulate all the parties except for $i^{*[e]}$ as in the Phase 3 of the signing algorithm and get $[\![\vec{\alpha}_1]\!], \ldots, [\![\vec{\alpha}_t]\!], [\![\vec{\beta}_1]\!], \ldots, [\![\vec{\beta}_t]\!], [\![v]\!]$ for all parties except for $i^{*[e]}$.
   - Compute $[\![v_j^{[e]}]\!]_{i^{*[e]}} := -\sum_{i \neq i^{*[e]}} [\![v_j^{[e]}]\!]_i$ for all $j \in [1:t]$.
5. Compute $h_1' = \mathrm{Hash}_1(m, \mathsf{com}_1^{[1]}, \mathsf{com}_2^{[1]}, \ldots, \mathsf{com}_{N-1}^{[\tau]}, \mathsf{com}_N^{[\tau]})$.
6. Compute $h_2' = \mathrm{Hash}_2(m, \{[\![\vec{\alpha}_j^{[e]}]\!], [\![\vec{\beta}_j^{[e]}]\!], [\![v_j^{[e]}]\!]\}_{j \in [1:t], e \in [1:\tau]})$.
7. Output Accept iff $h_1' \stackrel{?}{=} h_1$ and $h_2' \stackrel{?}{=} h_2$.

Figure 5.2.: Code-based signature scheme - Verification algorithm.

Choice of the SD parameters. Let us first describe how we estimate the security level of a syndrome decoding instance for a random linear code over $\mathbb{F}_2$. The best *practical* attack for our parameters is the algorithm of May, Meurer and Thomae [MMT11]. As argued in Section 4.5.2, we can lower bound the cost of this attack by only considering the cost of its topmost recursion step:

$$\frac{\binom{m}{w}}{\binom{k+\ell}{p}\binom{m-k-\ell}{w-p}} \cdot \left(L + \frac{L^2}{2^{\ell-p}} \right) \quad \text{with} \quad L := \frac{\binom{k+\ell}{p/2}}{2^p}.$$

As usual in an ISD algorithm we need to optimize for the parameters ℓ (a number of rows) and p (a partial Hamming weight). Since we only account for the cost of the topmost level in the algorithm, this yields a slightly conservative estimate for the security level. We use this estimate to choose the parameters of our scheme.

Given these considerations, we suggest the following concrete parameters:

- Variant 1: standard binary syndrome decoding problem. We propose the parameters

$$(q, m, k, w, d) = (2, 1280, 640, 132, 1)$$

  which achieve a security level of 128 bits according to the above formula.

- Variant 2: d-split binary syndrome syndrome decoding problem, where d is taken to have $m/d \leq 256$ so that $\mathbb{F}_{\mathrm{poly}} = \mathbb{F}_{256}$. We propose the parameters

$$(q, m, k, w, d) = (2, 1536, 888, 120, 6)$$

  which achieve a security of 129 bits. Indeed, the standard SD problem with the same parameters (but $d = 1$) has a security of 145 bits and we know, thanks to the Theorem 5.1.1, that there is a security loss of at most 16 bits while switching to $d = 6$.

Let us stress that this choice is conservative since the current state of the art does not contain attacks filling the gap of this reduction. Our aim here was to build a practical signature scheme with conservative security, but searching for more aggressive parameters for the d-split syndrome decoding problem would be an interesting direction for future research.

- Variant 3: syndrome decoding instance defined over $\mathbb{F}_{256}$. The cryptanalysis of the syndrome decoding problem on a field which is larger than $\mathbb{F}_2$ has been less studied. Previous articles [Pet10; CVE11; GPS22] propose parameters sets for syndrome decoding instances over $\mathbb{F}_{2^8}$ where the code length m is between 200 and 210. In our case, we choose $m = 256$ in such a way that the polynomial degree is equal to the field size. Besides being more conservative, this choice has the advantage of easing the use of a Fast Fourier Transform. We propose the following parameters[3] for this variant:

$$(q, m, k, w, d) = (256, 256, 128, 80, 1) \ .$$

Choice of the MPC parameters. For each variant, we suggest in Table 5.2 a parameter set for the MPC protocol.

Table 5.2.: SD and MPC parameters.

Scheme	SD Parameters					MPC Parameters			
	q	m	k	w	d	$\lvert\mathbb{F}_{\text{poly}}\rvert$	$\lvert\mathbb{F}_{\text{points}}\rvert$	t	p
Variant 1	2	1280	640	132	1	2^{11}	2^{22}	6	$\approx 2^{-69}$
Variant 2	2	1536	888	120	6	2^{8}	2^{24}	5	$\approx 2^{-79}$
Variant 3	2^8	256	128	80	1	2^{8}	2^{24}	5	$\approx 2^{-78}$

To have a short signature, we take the smallest possible field $\mathbb{F}_{\text{poly}}$ since a signature transcript includes polynomials on that field. As explained in Section 5.2, $\mathbb{F}_{\text{poly}}$ must be a field extension of $\mathbb{F}_{\text{SD}}$ which verifies the relation $|\mathbb{F}_{\text{poly}}| \geq m/d$. Then, it remains to choose $|\mathbb{F}_{\text{points}}|$ and t. These parameters are chosen to make the false positive probability p is negligible compared to $1/N$ such that the optimal forgery strategy of an attacker is to take $\tau_1 = 1$ in the Equation (5.4). As a result, we just need to increase the number of iterations τ by one compared to the interactive protocol.

5.3.5. Implementation and Performance

For each repetition, in the computation of each party, d polynomial interpolations are involved. Indeed, from $[\![x]\!]$, the parties must compute

$$[\![S_\ell]\!](X) = \sum_{i=1}^{m/d} [\![x_{\frac{m}{d}\ell+i}]\!] \cdot \prod_{j=1, j\neq i}^{m/d} \frac{X - w_j}{w_i - w_j}$$

[3]More cryptanalysis of the SD problem over $\mathbb{F}_{256}$ would be welcome to get more confidence in the choice of the parameters. Such research is out of the scope of this chapter.

for all $\ell \in [1:d]$. Then, the parties must evaluate $[\![S_\ell]\!]$ in t random evaluation points sampled by the verifier, for all $\ell \in [1:d]$. The natural way to implement that is to compute the coefficients of all the polynomials $\{[\![S_\ell]\!]\}_\ell$ from $[\![x]\!]$, then to evaluate these polynomials t times. However this implies that the signer must realize $\tau \cdot N \cdot d$ interpolations. Instead, the signer can compute the vector $u(r)$ defined as

$$u(r) = \left(\prod_{j=1, j\neq i}^{m/d} \frac{r - w_j}{w_i - w_j} \right)_{1 \leq i \leq \frac{m}{d}}$$

for each evaluation point r, and then use these vectors in the computation of all the parties as

$$[\![S_\ell(r)]\!] = \langle [\![x_\ell]\!], u(r) \rangle$$

where $[\![x_\ell]\!]$ is the ℓth chunk of $[\![x]\!]$. By proceeding this way, the number of (transposed) interpolations done by the signer is of $\tau \cdot t$.

To reduce the computational cost of the interpolations, we can make use of a Fast Fourier Transform (FFT). We are working on field extensions of $\mathbb{F}_2$, so we can use the Additive FFT independently introduced by Wang-Zhu in 1988 [WZ88] and by Cantor in 1989 [Can89], which was further improved in [GG03; GM10]. Although such additive FFT exists for any extension of $\mathbb{F}_2$, the algorithms are simpler for a field of size $2^{(2^i)}$ for some i, which is why we chose $\mathbb{F}_{\text{poly}}$ as $\mathbb{F}_{256}$. On such a field $\mathbb{F}$, we indeed have an efficient additive FFT using $\frac{1}{2}|\mathbb{F}| \log_2 |\mathbb{F}|$ multiplications to evaluate a polynomial (of degree lower than $|\mathbb{F}|$) in $|\mathbb{F}|$ points.

We implemented the signature scheme in C. In our implementation, the pseudo-randomness is generated using AES in counter mode and the hash function is instantiated with SHAKE. We benchmarked our scheme on a 3.8 GHz Intel Core i7 CPU with support of AVX2 and AES instructions. All the reported timings were measured on this CPU while disabling Intel Turbo Boost.

Remark 5.3.2. *Another motivation for using $\mathbb{F}_{poly} = \mathbb{F}_{256}$ is that some Intel processors have dedicated instructions for $\mathbb{F}_{256}$ arithmetic. We therefore expect substantial speed-ups for the instances of our signature scheme using $\mathbb{F}_{poly} = \mathbb{F}_{256}$ on these processors. Optimizing and benchmarking such implementations is left for future research.*

We instantiate two trade-offs per variant: the first one lowering communication cost to produce short signatures, and the second one lowering computational cost to get a fast signature computation. We obtain the parameters and sizes described in Table 5.3. We provide the measured running times of our signature implementation in Table 5.4.

Future investigations. We tried to optimize the implementation using some algorithmetic tricks, but we did not yet investigate the possible software optimizations like vectorization or bitslicing. Although the variants 1 and 2 are more conservative because they rely on the hardness of the binary syndrome decoding problem, variant 3 is more promising in terms of signature size and computation time. While we have investigated parameter sets where $\mathbb{F}_{\text{SD}}$ is a field extension of $\mathbb{F}_2$, more cryptanalysis for the SD problem on those fields as well as on non-binary fields would be welcome. An interesting idea would be to instantiate our scheme with a prime field $\mathbb{F}_{\text{SD}}$ for which the Number-Theoretic Transform (NTT) is defined. If $\mathbb{F}_{\text{SD}}$ is large enough, we could then take the same field for $\mathbb{F}_{\text{poly}}$ and $\mathbb{F}_{\text{SD}}$, and we would have fast polynomial interpolations and simpler multiplication operations.

Table 5.3.: Parameters (N, τ) with the achieved communication costs (in bytes).

λ	Scheme	Aim	Parameters		Signature		
			N	τ	$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{sgn}\rvert$ (max)	$\lvert\mathsf{sgn}\rvert$ (avg, std)
128	Variant 1	Fast	32	27	96	16 422	16 006, 446
128		Short	256	17	96	11 193	11 160, 127
128	Variant 2	Fast	32	27	97	17 866	17 406, 494
128		Short	256	17	97	12 102	12 066, 141
128	Variant 3	Fast	32	27	144	12 115	11 835, 302
128		Short	256	17	144	8 481	8 459, 86

Table 5.4.: Benchmarks of our signature implementation.

λ	Scheme	Aim	Keygen		Sign		Verify	
			ms	cycles	ms	cycles	ms	cycles
128	Variant 1	Fast	n/a†					
128		Short						
128	Variant 2	Fast	0.03	114k	13.4	52M	12.7	50M
128		Short	0.03	114k	64.2	251M	60.7	243M
128	Variant 3	Fast	0.01	49k	6.4	25M	5.9	24M
128		Short	0.01	49k	29.5	114M	27.1	109M

Note: Timings are averaged over 10 000 measurements. The CPU clock cycles have been measured using SUPERCOP (https://bench.cr.yp.to/supercop.html).
† *We only have a proof of concept implementation with irrelevant timings.*

5.4. Comparison

In this section, we compare our scheme to different code-based and post-quantum signature schemes from the literature.

5.4.1. Comparison with Other Code-Based Signature Schemes

We extend the comparison made in Section 4.6.1 with the scheme proposed in this chapter. We add in the comparison the work [BGKM22] which has been released just before this scheme.

Table 5.5 compares the performance of our scheme with the current code-based signature state of the art, for the 128-bit security level.[4] We observe that our scheme outperforms all the existing code-based signatures for the $\lvert\mathsf{sgn}\rvert + \lvert\mathsf{pk}\rvert$ metric. Depending on the parameters, it can even produce signatures such that $\lvert\mathsf{sgn}\rvert + \lvert\mathsf{pk}\rvert$ is below the symbolic cap of 10 KB.

[4] We did not include "Sig 3" from [BGKM22] since it is similar to [FJR21] with slight differences (*message decoding* setting) which do not improve the scheme.

Table 5.5.: Comparison of our scheme with signatures from the literature (128-bit security).

Scheme Name	Year	$\lvert\mathsf{sgn}\rvert$	$\lvert\mathsf{pk}\rvert$	t_{sgn}	t_{verif}
Wave	2019	2.07 K	3.2 M	300	-
Durandal - I	2018	3.97 K	14.9 K	4	5
Durandal - II	2018	4.90 K	18.2 K	5	6
LESS-FM - I	2020	15.2 K	9.77 K	-	-
LESS-FM - II	2020	5.25 K	206 K	-	-
LESS-FM - III	2020	10.39 K	11.57 K	-	-
[GPS22]-256	2021	24.0 K	114	-	-
[GPS22]-1024	2021	19.8 K	122	-	-
Chapter 4 (fast)	2021	22.6 K	96	13	12
Chapter 4 (short)	2021	16.0 K	96	62	57
[BGKM22] - Sig1	2022	23.7 K	91	-	-
[BGKM22] - Sig2	2022	20.6 K	171	-	-
Our scheme - Var1f	2022	15.6 K	96	-	-
Our scheme - Var1s	2022	10.9 K	96	-	-
Our scheme - Var2f	2022	17.0 K	97	13	13
Our scheme - Var2s	2022	11.8 K	97	64	61
Our scheme - Var3f	2022	11.5 K	144	6	6
Our scheme - Var3s	2022	8.26 K	144	30	27

Note: The sizes are in bytes and the timings are in milliseconds. Reported timings are from the original publications: Wave has been benchmarked on a 3.5 Ghz Intel Xeon E3-1240 v5, Durandal on a 2.8 Ghz Intel Core i5-7440HQ, while [FJR21] and our scheme on a 3.8 GHz Intel Core i7.

Regardless of the key size, Wave still achieves the shortest signatures. In terms of security, our scheme has the advantage of relying on the hardness of one of the oldest problems of the code-based cryptography, namely the syndrome decoding for random linear codes in Hamming weight metric.

5.4.2. Comparison with other Post-Quantum Signature Schemes

Finally, we compare in Table 5.6 our construction with other signature schemes aiming at post-quantum security. First of all, let us note that the lattice-based signature schemes (such as Dilithium [BDK+21a] and Falcon [FHK+20]) are currently the most efficient post-quantum signature schemes. They achieve small signature size and efficient running time. However, the goal of our construction is to propose a signature scheme based on an alternative problem for the sake of diversity of security assumptions. All the others schemes have very short public keys and secret keys (less than 150 bytes for 128-bit security), which is hence not a point for comparison. Depending on the chosen parameters, our scheme can be competitive with Picnic3 [KZ20b] and an optimized variant of Picnic proposed by [KZ22] which also rely on the MPC-in-the-Head paradigm. Like this optimized variant, we can produce signatures

with a size of around 8 KB. However, our scheme is arguably more conservative in terms of security since Picnic is based on the hardness of inverting LowMC [ARS+15], a cipher with unconventional design choices, while our scheme is based on the hardness of the syndrome decoding problem on linear codes, which has a long cryptanalysis history and is believed to be very robust. Banquet [BDK+21b] is a signature scheme for which the security is based on the hardness of inverting AES (instead of LowMC), which can also be argued to be a conservative choice. Our scheme over $\mathbb{F}_2$ is competitive with Banquet: slightly shorter and slightly slower (but the timing could be optimized). On the other hand, our scheme on $\mathbb{F}_{256}$ clearly outperforms Banquet. Our scheme can also be competitive with SPHINCS$^+$ [BHK+19] depending on the exact criteria. For similar signature sizes, our signature computation is significantly faster while our signature verification is significantly slower than those of SPHINCS$^+$.

Table 5.6.: Comparison of our scheme with signatures from the literature (128-bit security).

Scheme Name	$\lvert\mathsf{sgn}\rvert$	$\lvert\mathsf{pk}\rvert$	t_{sgn}	t_{verif}
Dilithium2	2.4 K	1.3 K	0.065	0.024
Falcon-512	0.65 K	0.88 K	0.168	0.036
SPHINCS$^+$-128f	16.7 K	32	14	1.7
SPHINCS$^+$-128s	7.7 K	32	239	0.7
Picnic3	12.3 K	32	5.2	4.0
Picnic4	7.8 K	32	≈ 20	≈ 20
Banquet (fast)	19.3 K	32	6	5
Banquet (short)	13.0 K	32	44	40
Our scheme - Var1f	15.6 K	96	-	-
Our scheme - Var1s	10.9 K	96	-	-
Our scheme - Var2f	17.0 K	97	13	13
Our scheme - Var2s	11.8 K	97	64	61
Our scheme - Var3f	11.5 K	144	6	6
Our scheme - Var3s	8.3 K	144	30	27

Note: The sizes are in bytes and the timings are in milliseconds. Reported timings for Falcon have been benchmarked on a 2.3 Ghz Intel Core i5-8259U in [FHK+20], and timings for Dilithium and our scheme have been benchmarked on a 3.8 Ghz Intel Core i7. The benchmarks of the other schemes have been realized on a Intel Xeon W-2133 CPU at 3.60GHz, the values for SPHINCS$^+$ and Banquet have been extracted from [BDK+21b] while the values for Picnic3 have been extracted from its original publication [KZ20b].

5.5. Conclusion

In this chapter, we showed how we can apply the MPC-in-the-Head paradigm to code-based cryptography. The proposed signature scheme can achieve very competitive sizes (8-9 KB for 128-bit security), which is almost half the size of the best former schemes (with the same security assumption).

The main issue of this scheme is its computational cost. To achieve such interesting sizes,

the running times of the signer and of the verifier are in tens of milliseconds. However, some follow-up works propose optimizations of the scheme:

- *The Return of the SDitH.* C. Aguilar-Melchor, N. Gama, J. Howe, A. Hülsing, D. Joseph, and D. Yue [AGH+23]. This work shows that we can drastically decrease the computational cost of the MPC emulation, achieving running times below 5 milliseconds while keeping the same signature sizes;
- *Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head.* T. Feneuil and M. Rivain [FR22]. This work (presented in Chapter 8) proposes an alternative MPCitH transformation to get a proof of knowledge from an MPC protocol. When applied to the scheme of this chapter, the construction leads to a running time of a few milliseconds for the signer and of less than 0.5 milliseconds for the verifier. However, the signature size suffers from a 2-kilobyte increase (for 128-bit security).

Before the construction proposed in this chapter, the MPC-in-the-Head paradigm had been used only on symmetric primitives (AES, LowMC, Rain, ...). Our construction demonstrates that this paradigm can also lead to efficient schemes for other types of security assumptions.

Chapter 6.

MPC-in-the-Head with Rejection

Before 2020, the zero-knowledge proofs for code-based cryptography were not efficient. This state of affairs is similar for the subset sum problem. Until recently, the communication cost for proofs relying on this problem was of few hundreds/thousands of kilobytes. In the previous chapters, we showed that the MPC-in-the-Head paradigm enables us to build competitive code-based schemes. In this chapter, we explore how to use the MPCitH paradigm to build efficient zero-knowledge proofs and signature schemes for the subset sum problem. Because of the large underlying modulus, a straightforward application of the paradigm does not give acceptable performance. Therefore, we adapt it by using a secret sharing over small integers (rather than with the modulus) to reduce the size of the arguments. Since this sharing may reveal information on the secret, we introduce the idea of rejection to the MPC-in-the-head paradigm. Special care has to be taken to balance completeness and soundness and preserve the zero-knowledge property of our arguments. By combining this idea with two techniques to prove that the secret vector is well made of binary coordinates, we obtain efficient zero-knowledge proofs for the subset sum problem, enabling us to get a signature scheme relying on the hardness of this problem. We also apply the secret sharing over small integers to build a signature scheme relying on the security of the [BHH01] pseudo-random function.

Most of the results presented in this chapter have been published in collaboration with Jules Maire, Matthieu Rivain and Damien Vergnaud in the proceedings of the international conference *Asiacrypt 2022* [FMRV22b].

Contents

6.1. Introduction **82**
6.2. General Idea **84**
6.3. Protocols and Security Proofs **92**
6.4. Instantiations and Performance **97**
6.5. Digital Signatures from Boneh-Halevi-Howgrave-Graham PRF . . **100**
6.6. Conclusion **105**

6.1. Introduction

Given integers $w_1, \ldots, w_n$, t and q, the *(modular) subset sum* problem consists in finding a subset of the w_i's that sum to t modulo q, *i.e.* to find bits $x_1, \ldots, x_n \in \{0, 1\}$ such that

$$\sum_{i=1}^{n} x_i w_i = t \bmod q. \tag{6.1}$$

More formally, an instance of this problem is built as explained in the below definition.

Definition 6.1.1 (Subset Sum Problem)**.** *Let q and n be positive integers. The subset sum problem with parameters (q, n) is the following problem:*

Let w, x and t be such that:

1. *w is uniformly sampled from $\mathbb{F}_q^n$,*
2. *x is uniformly sampled from $\{0, 1\}^n$,*
3. *t is defined as $t := \sum_{i=1}^{n} w_i \cdot x_i \bmod q$.*

From (w, t), find x.

This problem was shown to be NP-complete (in its natural decision variant) in 1972 by Karp [Kar72] and was considered in cryptography as an interesting alternative to hardness assumptions based on number theory. Due to its simplicity, it was notably used in the 1980s, following [MH78], for the construction of several public-key encryption schemes.

Most of these proposals (if not all) were swiftly broken using lattice-based techniques (see [Odl90]), but the problem itself remains intractable for appropriate parameters and is even believed to be so for quantum computers. For instance, when the so-called density $d = n/\log_2(q)$ of the subset sum instance is close to 1 (i.e. $q \simeq 2^n$), the fastest known (classical and quantum) algorithms have complexity $2^{O(n)}$ (see [BBSS20] and references therein) and one can reach an alleged security level of λ bits with $n = \Theta(\lambda)$. Many cryptographic constructions were proposed whose security relies on the hardness of the subset sum problem: pseudo-random generators [IN96], bit commitments [IN96], public-key encryption [AD97; LPS10]

6.1.1. Prior Work

Given integers $w_1, w_2, \ldots, w_n$, t and q, an elegant zero-knowledge proof system due to Shamir [Sha86] (see also [BGKW90; Sim91; Blo09]) allows a prover to convince a verifier that she knows $x_1, \ldots, x_n \in \{0, 1\}$ such that Equation (6.1) holds. The proof system is combinatorial in nature and it requires $\Theta(\lambda)$ rounds of communication to achieve soundness error $2^{-\lambda}$ where each round requires $\Theta(n^2)$ bits of communication. For an alleged security level of λ bits, the overall communication complexity of Shamir's proof system is thus of $\Theta(\lambda^3)$. In [LNSW13], Ling, Nguyen, Stehlé, and Wang proposed a proof of knowledge of a solution for the infinity norm *inhomogeneous small integer solution* (ISIS) problem which is a vectorial variant of the subset sum problem. It is based on Stern's zero-knowledge proof of knowledge for the *syndrome decoding* problem [Ste94] and is also combinatorial. It thus requires a large number of rounds of communication and when specialized to the subset sum

problem it also yields proofs with $\Theta(\lambda^3)$-bit communication complexity. When instantiated with parameters for 128-bit security, these two proofs of knowledge have a communication of few thousands of kilobytes.

Using the MPC-in-the-head paradigm, Baum and Nof [BN20] proposed an efficient zero-knowledge argument of knowledge of the *short integer solution* (SIS) problem. Beullens also recently proposed such arguments obtained from sigma protocols *with helper* [Beu20]. When applied to the subset sum problem itself, all (variants of) these protocols yield proofs with $\Theta(\lambda^3)$-bit communication complexity for an alleged security level of λ bits. When instantiated with parameters for 128-bit security, they have a communication of at least few hundreds of kilobytes.

6.1.2. Contributions

In the MPC-in-the-head paradigm, the prover wants to convince a verifier that she knows a pre-image x of $y = f(x)$ for some one-way function f where the function f is represented as an arithmetic circuit. For the subset sum problem, the function f is defined *via* Equation (6.1) and it is thus natural to consider the simple inner-product arithmetic circuit defined over $\mathbb{Z}_q$. The prover's secret input is the binary vector $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ and she has to perform some secret-sharing of x in $\mathbb{Z}_q$ in such a way that the shares of any unauthorized set of parties should reveal no information about the secret. This approach has the major disadvantage that sharing a single bit requires several elements of $\mathbb{Z}_q$ each of size $\Theta(\lambda)$ bits.

We adapt the paradigm using a secret sharing scheme done directly over the integers. This approach was already used in cryptography (e.g. for multi-party computation modulo a shared secret modulus [CGH00]). To additively share a secret t in a given interval $[-T, T]$ for $T \in \mathbb{N}$, among $N \geq 2$ parties, a dealer may pick uniformly at random $t_1, \ldots, t_N \in [-T2^\rho, T2^\rho]$ under the constraint that $t = t_1 + \cdots + t_N$ (over the integers), for some parameter ρ. However, given $(N-1)$ shares, $t_2, \ldots, t_N$ for instance, the value $t_1 = t - (t_2 + \cdots t_N)$ is not randomly distributed in $[-T2^\rho, T2^\rho]$ and this may reveal information on the secret t. It is thus necessary to sample the shares in an interval sufficiently large in such a way that their distributions for distinct secrets are statistically indistinguishable. For a security level λ, this requires $\rho = \Omega(\lambda)$ and thus the additive sharing of bits involves shares of size $\Omega(\lambda)$. To overcome this limitation and use additive secret sharing over *small* integers, we will rely on *rejection*. The computation being actually simulated by the prover, they can abort the protocol whenever the sharing leaks information on the secret vector $x = (x_1, \ldots, x_n) \in \{0,1\}^n$. In some cases, the prover cannot respond to the challenge from the verifier and must abort the protocol. A similar idea was used for lattice-based signatures by Lyubashevsky [Lyu08; Lyu09] but using different methods.

Our technique also allows us to overcome the second disadvantage of the previous attempts to use the MPC-in-the-head paradigm for lattice-based problems. Using our additive secret sharing over the integers, we can prove the knowledge of some integer vector $x = (x_1, \ldots, x_n)$ satisfying Equation (6.1) (for any q) and further prove that $x_i \in \{0,1\}$ for $i \in \{1, \ldots, n\}$. This is achieved by emulating a non-linear MPC operation modulo some arbitrary prime number q' (independent from q and much smaller than q). We also introduce another technique to prove that the solution $x = (x_1, \ldots, x_n)$ indeed lies in $\{0,1\}^n$ using some masking and a *cut-and-choose* strategy. Both methods yield zero-knowledge proofs with $\Theta(\lambda^2)$-bit communication complexity for an alleged security level of λ bits. This improvement is not only of theoretical interest since for $q \simeq 2^{256}$, our protocol can produce proof of size

13KB where Shamir's protocol [Sha86] (updated with modern tips) produces proof of size 1186KB and [LNSW13] produces proofs of size 2350KB.

Our method with the sharing over small integers is not specific to the subset sum problem and can be interesting as soon as we must deal with small secret values in a context where the modulus is large. As another illustration, we use our technique to construct an efficient digital signature scheme based on a pseudo-random function due to Boneh, Halevi, and Howgrave-Graham [BHH01].

Remark 6.1.2. *In this introduction, we insisted on the fact that former zero-knowledge proofs for the subset sum problem achieve communication in $\Theta(\lambda^3)$ bits and, in this chapter, we present a scheme with $\Theta(\lambda^2)$-bit communication. Searching to achieve communication in $\Theta(\lambda^2)$ is very natural. For other hard problems (syndrome decoding, multivariate quadratic, ...), all the zero-knowledge proofs with non-negligible soundness error have communication in $\Theta(\lambda^2)$ bits because*

- *the communication of a single repetition of these zero-knowledge proofs scales linearly with the size of the problem instance, which scales itself in $\Theta(\lambda)$ bits,*
- *we need to repeat the protocol $\Theta(\lambda)$ times to have a λ-bit security.*

6.1.3. Preliminaries

In Section 2.4.1, we defined the additive sharing scheme such that, to share a secret s, we choose $[\![s]\!]_1, \ldots, [\![s]\!]_N$ satisfying

$$s = [\![s]\!]_1 + \ldots + [\![s]\!]_N.$$

In this chapter, we will use a slightly different definition. To share a secret s among N parties, we choose uniformly at random $[\![s]\!]_1, \ldots, [\![s]\!]_N$ and we set a sharing offset Δs such that

$$s = [\![s]\!]_1 + \ldots + [\![s]\!]_N + \Delta s.$$

The sharing offset is a public part of the sharing that every party of the MPC protocol knows, while the ith share $[\![s]\!]_i$ is known only by the ith party. We use this slightly different definition since Δs will not have the same definition domain as the shares $[\![s]\!]_1, \ldots, [\![s]\!]_N$ in this chapter.

6.2. General Idea

We consider an instance $(w, t) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ of the subset sum problem (SSP) and denote x one solution. We have $x \in \{0, 1\}^n$ and $\sum_{j=1}^n x_j \cdot w_j = t \bmod q$.

We want to use the MPCitH paradigm to build a zero-knowledge protocol that proves the knowledge of a solution for the instance (w, t). To proceed, we need to build an MPC protocol taking as inputs shares of the secret x, and possibly shares of other data, and which outputs Accept when x is a valid solution of the SSP instance. As a first ingredient, we need a method to share the secret x between the different parties.

6.2.1. The Naive Approach

The SSP instance is defined on $\mathbb{Z}_q$, so a natural sharing of x would be defined as:

$$\begin{cases} [\![x]\!]_i \leftarrow (\mathbb{Z}_q)^n \text{ for all } i \in [1:N], \\ \Delta x \leftarrow x - \sum_{i=1}^{N} [\![x]\!]_i \bmod q \end{cases} .$$

In the MPCitH paradigm with additive sharings (*c.f.* Section 3.1.2.2), the communication cost induced by a sharing is the cost to send the sharing offset, *i.e.* the vector Δx. Here, the natural sharing of x costs

$$n \cdot \log_2(q) \text{ bits.}$$

If we take $n = 256$ and $q = 2^{256}$, the cost is about 2^{16} bits = 8 KB. To achieve a soundness error of 2^{-128} with $N = 256$, we need to repeat the protocol at least 16 times, so the communication cost of the protocol would be already more than 128 KB for the sole sharing of x (some communication being further required for the MPCitH protocol). Asymptotically, the parameters for the subset sum problem are chosen such that $n = \Theta(\lambda)$ and $\log_2 q = \Theta(\lambda)$, the communication cost of this sharing is thus about $\Theta(\lambda^2)$ bytes per protocol repetition. Since we need to repeat the protocol about $\Theta(\lambda)$ times to achieve a $2^{-\lambda}$ soundness error the global communication cost is then of at least $\Theta(\lambda^3)$ (for the sharing only).

We present hereafter an alternative strategy for the sharing of x, which achieves better practical and asymptotic communication costs.

6.2.2. Sharing on the Integers and Opening with Abort

We propose another way to share the secret x to achieve lower communication. We know that x is a binary vector (*i.e.* $x \in \{0,1\}^n$), so instead of the natural sharing, we suggest to use a sharing defined on the integers, that is

$$\begin{cases} [\![x]\!]_i \leftarrow \{0, \ldots, A-1\}^n \text{ for all } i \in [1:N], \\ \Delta x \leftarrow x - \sum_{i=1}^{N} [\![x]\!]_i \end{cases}$$

with A some positive constant. However, this sharing leaks information about the secret x. The distribution Δx_j is not the same depending on whether $x_j = 0$ or $x_j = 1$ as illustrated on Figure 6.1. To solve this issue, the prover must abort the protocol in some cases.

To see how this leakage can be effectively exploited to (partly) recover x, let us recall that at the end of the protocol, the verifier shall ask the prover to open the views of all parties except one. Let us denote i^* the index of the unopened party. It means the verifier will have access to

$$\{[\![x]\!]_i\}_{i \neq i^*} \quad \text{and} \quad \Delta x \ .$$

For the sake of simplicity, let us first consider the case $n = 1$, *i.e.* $x \in \{0,1\}$ and $[\![x]\!]$ is the sharing of a single integer. With the opened values, the verifier can compute

$$x - [\![x]\!]_{i^*} \quad \text{as} \quad \Delta x + \sum_{i \neq i^*} [\![x]\!]_i \ .$$

Now let us denote $Y = x - [\![x]\!]_{i^*}$ the underlying random variable over the uniform random sampling of $[\![x]\!]_{i^*}$. We have

$$\Pr(Y = -A+1) = \begin{cases} \frac{1}{A} & \text{if } x = 0 \\ 0 & \text{if } x = 1 \end{cases} \quad \text{and} \quad \Pr(Y = 1) = \begin{cases} 0 & \text{if } x = 0 \\ \frac{1}{A} & \text{if } x = 1 \end{cases}$$

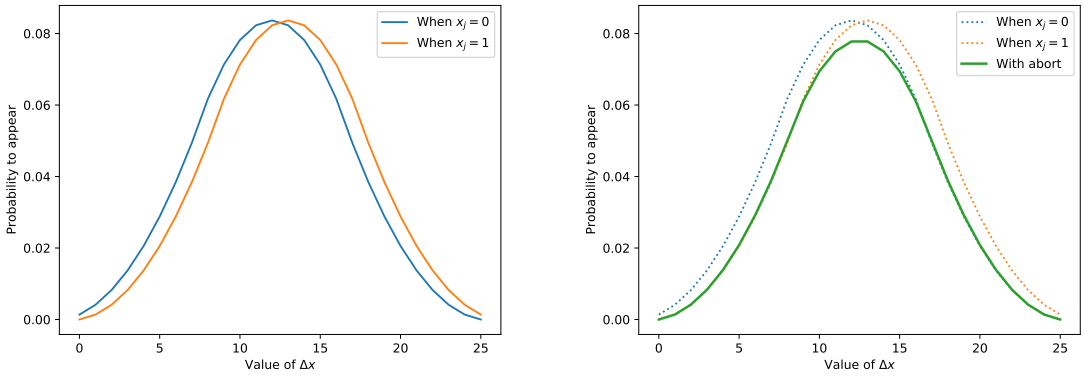


Figure 6.1.: Probability mass function of Δx_j when $x_j = 0$ and when $x_j = 1$ (on the left) and of Δx_j with abort (on the right), for $N = 3$ and $A = 9$.

while

$$\Pr(Y = y) = \frac{1}{A} \quad \text{for every } y \in \{-A+2, \ldots, 0\} \ .$$

So by observing $x - [\![x]\!]_{i^*} = -A+1$ one learns $(x, [\![x]\!]_{i^*}) = (0, -A+1)$. Similarly, by observing $x - [\![x]\!]_{i^*} = 1$ one learns $(x, [\![x]\!]_{i^*}) = (1, 0)$. To avoid this flaw, the prover must abort the protocol before revealing $\{[\![x]\!]_i\}_{i \neq i^*}$ and Δx whenever one of these two cases occurs. This notably implies that Δx must not be revealed before receiving the challenge i^*, but it should still be committed beforehand in order to ensure the soundness of the protocol. Doing so, we modify the distribution of the revealed sharing offset which does not leak any information about x anymore as illustrated in Figure 6.1, and the probability to abort does not leak information about x since it is $1/A$ in the both cases ($x = 0$ and $x = 1$).

Let us now come back to the general case of $n \geq 1$. The prover applies the above abort strategy for all the coordinates of x, namely

- if there exists $j \in [n]$ such that $x_j = 0$ and $[\![x_j]\!]_{i^*} = A - 1$, the prover aborts;
- if there exists $j \in [n]$ such that $x_j = 1$ and $[\![x_j]\!]_{i^*} = 0$, the prover aborts;
- otherwise the prover proceeds.

The probability to abort, which we call *rejection rate*, is

$$1 - \left(1 - \frac{1}{A}\right)^n \leq \frac{n}{A} \ .$$

We note that the rejection rate can be tightly approximated by the n/A upper bound when A is sufficiently large. In order to achieve a small (constant) rejection rate, we should hence choose A greater than n. Asymptotically, we then have $A = \Theta(n) = \Theta(\lambda)$, which represents an exponential improvement compared to $q = 2^{\Theta(\lambda)}$.

Let us now analyze the computation cost of our strategy for sharing x. Let us assume that the verifier chose i^* and there was no rejection. Then, the prover needs to reveal Δx (to enable the verifier to re-emulate the MPC protocol). Δx_j belongs to $\{-N \cdot (A-1)+1, \ldots, 0\}$, therefore sending the sharing offset Δx would cost $n \cdot \log_2(N \cdot (A-1))$ bits. However, the

prover can save communication by sending $x - [\![x]\!]_{i^*}$ instead, which is strictly equivalent in terms of revealed information by the relation $x - [\![x]\!]_{i^*} = \Delta x + \sum_{i \neq i^*} [\![x]\!]_i$. Since each coordinate of $x - [\![x]\!]_{i^*}$ is uniformly distributed over $\{-A+2, \ldots, 0\}$, sending it only costs

$$n \cdot \log_2(A-1) \text{ bits.}$$

With $x - [\![x]\!]_{i^*}$, the verifier can recover Δx by computing $\Delta x = (x - [\![x]\!]_{i^*}) - \sum_{i \neq i^*} [\![x]\!]_i$. The cost of this sharing has the advantage of being independent of the modulus q on which the SSP instance is defined. The value of A will be chosen according to the desired trade-off between communication cost and rejection rate. If $n = 256$ and $A = 2^{16}$, we have a cost of 0.5 KB for a rejection rate of 0.0038, which is much better than the 8 KB of the naive approach.

Let us remark that adding an abort event does not impact the soundness of the protocol. A malicious prover can abort as many times she wants claiming that it would leak information, but an abort does not help to convince the verifier. The soundness theorem will state that someone who does not know the secret can only answer with a probability smaller than the constant value called soundness error, and adding an abort event cannot increase this probability. The prover could sample a random party i' and give to i' a wrong share and she may indeed decide to abort if the verifier challenge is not i', but this does not change the fact that the probability for the prover to convince the verifier is the probability that the prover guesses the verifier's challenge a priori.

Now that we have defined the sharing of x, we need to demonstrate two properties of the shared SSP instance through multi-party computation. The first one is the SSP relation which in the shared setting translates to

$$\sum_{j=1}^{n} [\![x_j]\!] \cdot w_j = [\![t]\!] \bmod q$$

for a sharing $[\![t]\!]$ of t. The linearity of this relation makes it easy to deal with: the share $[\![t]\!]_i$ can simply be computed as $[\![t]\!]_i := \sum_{j=1}^{n} [\![x_j]\!]_i \cdot w_j \bmod q$ and committed to the verifier by the ith party. The verifier can then check that the opened parties have correctly computed their shares $[\![t]\!]_i$ and that the relation $\sum_{i=1}^{N} [\![t]\!]_i = [\![t]\!] \bmod q$ well holds. The second property which must be demonstrated through multi-party computation is that the solution x corresponding to the sharing $[\![x]\!]$ is a binary vector. This is not a priori guaranteed to the verifier since the shares of the coordinate of x are defined over $\{0, \ldots, A-1\}$ and the correctness of the linear relation does not imply that x is indeed binary. We present two different solutions to this issue in the following.

6.2.3. Binarity Proof from Batch Product Verification

Our first solution to prove that the vector x is binary relies on standard MPC-in-the-Head techniques by checking the relation

$$x \circ (x - 1) = 0$$

where $\circ$ denotes the coordinate-wise product, 0 and 1 are to be interpreted as the all-0 and all-1 vectors. It consists in checking n multiplication triples, and so we can use one of the two multiplication verification protocols described in Section 3.4.1:

- the sacrificing-based verification [LN17; BN20; KZ22], or
- the polynomial-based verification [BDK+21b; FJR22b; KZ22].

However, we can do better than a straight application of those techniques.

The relation $x \circ (x-1) = 0$ is defined in $\mathbb{Z}_q$ and the above protocols imply to send at least one field element per product[1], that is n elements from $\mathbb{Z}_q$. To save communication and since the sharing $[\![x]\!]$ is defined on the integers, we can work on a *smaller field*. We previously explained that the verifier receives $\{[\![x]\!]_i\}_{i \neq i^*}$ and Δx from the prover, so she can check that, for all $j \in [n]$,

$$-A + 2 \leq x_j - [\![x_j]\!]_{i^*} \leq 0 \ .$$

She further trusts $[\![x_j]\!]_{i^*} \in \{0, \ldots, A-1\}$ (which is verified for the open parties). Thus the verifier can deduce that, for all $j \in [n]$,

$$-A + 2 \leq x_j \leq A - 1 \ . \tag{6.2}$$

Let q' be a prime such that $q' \geq A$. If the prover convinces the verifier that $x_j(x_j - 1) = 0 \bmod q'$, then the latter deduces that $x_j \in \{0, 1\}$ because

$$\begin{aligned} q' | x_j(x_j - 1) \ &\Rightarrow \ (q' | x_j) \text{ or } (q' | x_j - 1) \\ &\Rightarrow \ (x_j = 0) \text{ or } (x_j = 1) \quad \text{by Equation (6.2).} \end{aligned}$$

So, the prover just needs to prove $x \circ (x-1) = 0 \bmod q'$ for some prime q' such that $q' \geq A$. To this purpose, we apply one of the two batch product verifications.

Using sacrificing-based verification. The prover first samples $a \in (\mathbb{Z}_{q'})^n$ with its sharing

$$[\![a]\!]_i \leftarrow (\mathbb{Z}_{q'})^n \text{ for } i \in [1:N] \ .$$

The value a is hence defined as a uniform random element of $(\mathbb{Z}_{q'})^n$ and no sharing offset Δa is necessary ($\Delta a := 0$). The prover then computes $c = \langle a, x \rangle$ and its sharing as

$$\begin{cases} [\![c]\!]_i \leftarrow \mathbb{Z}_{q'} \text{ for all } i \in [1:N], \\ \Delta c \leftarrow c - \sum_{i=1}^{N} [\![c]\!]_i \bmod q'. \end{cases}$$

The prover gives the shares of x, a and c as input to the parties and runs the following MPC protocol:

1. the parties get a random challenge $\varepsilon \in (\mathbb{Z}_{q'})^n$ from the verifier;
2. the parties locally set $[\![\alpha]\!] = \varepsilon \circ (1 - [\![x]\!]) + [\![a]\!]$, where $\circ$ denotes the coordinate-wise product;
3. the parties open $[\![\alpha]\!]$ to get α;
4. the parties locally set $[\![v]\!] = \langle \alpha, [\![x]\!] \rangle - [\![c]\!]$;
5. the parties open $[\![v]\!]$ to get v;
6. the parties accept iff $v = 0$.

[1] assuming that the MPC protocol asks the verifier for randomness only once

Besides the input shares and commitments, the prover-to-verifier communication cost of the corresponding MPCitH zero-knowledge protocol only results from the size of $[\![\alpha]\!]_{i^*}$ (the broadcasted vector of the unopened party i^*), which is of around

$$n \cdot \log_2(q') \text{ bits.}$$

We stress that the prover does not need to send $[\![v]\!]_{i^*}$ because the verifier knows that v must be zero and will deduce $[\![v]\!]_{i^*} = -\Delta v - \sum_{i \neq i^*} [\![v]\!]_i$.

The sacrificing-based multiplication verification produces false positives with probability $1/q'$. Thus the soundness error of the obtained zero-knowledge protocol is

$$1 - \Big(1 - \frac{1}{N}\Big)\Big(1 - \frac{1}{q'}\Big) \approx \frac{1}{N} + \frac{1}{q'} \ .$$

On the other hand, the resulting protocol has a rejection rate of $1 - (1 - \frac{1}{A})^n$ and a prover-to-verifier communication cost (in bits) of

$$2 \cdot (2\lambda) + \underbrace{n \cdot \log_2(A-1)}_{x - [\![x]\!]_{i^*}} + \underbrace{n \cdot \log_2(q')}_{\Delta\alpha} + \underbrace{\log_2(q')}_{\Delta c} + \lambda \log_2 N + 2\lambda \ .$$

Using polynomial-based verification. Let us consider some distinct public points $\gamma_1, \ldots, \gamma_n \in \mathbb{F}_{q'}$ and an additional scheme parameter ν. The prover first computes the degree-$(n-1)$ polynomial S by interpolation such that $S(\gamma_i) = x_i$ for all i. Since $x \circ (x-1) = 0$, we have that there exists a degree-$(n-2)$ polynomial P such that

$$S(X) \cdot (S(X) - 1) = P(X) \cdot \prod_{i=1}^{n} (X - \gamma_i). \tag{6.3}$$

The prover computes this polynomial P with its sharing $[\![P]\!]$. The prover also samples $a \in (\mathbb{Z}_{q'^\nu})$ with its sharing $[\![a]\!]$. The value is defined as a uniform random element of $(\mathbb{Z}_{q'^\nu})$, so no sharing offset Δa is necessary. The prover then computes $c = a^2$ and its sharing $[\![c]\!]$.

The prover gives the shares of x, P, a and c as input to the parties and runs the following MPC protocol:

1. the parties get random challenges $r, \varepsilon \in (\mathbb{Z}_{q'^\nu})^n$ from the verifier;
2. the parties locally compute $[\![S]\!]$ by interpolation:

$$[\![S]\!] = \sum_{i=1}^{n} [\![x_i]\!] \prod_{j=1, j \neq i}^{n} \left(\frac{X - \gamma_j}{\gamma_i - \gamma_j} \right)$$

3. the parties locally set $[\![S(r)]\!] = [\![S]\!](r)$ and $[\![P(r)]\!] = [\![P]\!](r)$
4. the parties locally set $[\![\alpha]\!] = \varepsilon \cdot [\![S(r)]\!] + [\![a]\!]$;
5. the parties open $[\![\alpha]\!]$ to get α;
6. the parties locally set

$$[\![v]\!] = \varepsilon^2 \cdot \left([\![P(r)]\!] \cdot \prod_{i=1}^{n} (r - \gamma_i) + [\![S(r)]\!] \right) - \alpha \cdot (\varepsilon \cdot [\![S(r)]\!] - [\![a]\!]) - [\![c]\!];$$

7. the parties open $[\![v]\!]$ to get v;
8. the parties accept iff $v = 0$.

The polynomial-based multiplication verification produces false positives with probability

$$\underbrace{\frac{2n-1}{q'^{\nu}}}_{\substack{\text{false positive}\\ \text{from Schwartz-Zippel}}} + \left(1 - \frac{2n-1}{q'^{\nu}}\right) \cdot \underbrace{\frac{2}{q'^{\nu}}}_{\substack{\text{false positive}\\ \text{from [BN20]}}} .$$

Thus the soundness error of the obtained zero-knowledge protocol is

$$1 - \left(1 - \frac{1}{N}\right)\left(1 - \frac{2n-1}{q'^{\nu}}\right)\left(1 - \frac{2}{q'^{\nu}}\right) \approx \frac{1}{N} + \frac{2n+1}{q'^{\nu}} .$$

On the other hand, the protocol has a rejection rate of $1 - (1 - \frac{1}{A})^n$ and a prover-to-verifier communication cost (in bits) of

$$2 \cdot (2\lambda) + \underbrace{n \cdot \log_2(A-1)}_{x - [\![x]\!]_{i^*}} + \underbrace{(n-1) \cdot \log_2(q')}_{\Delta P} + \underbrace{\nu \cdot \log_2(q')}_{\Delta\alpha} + \underbrace{\nu \cdot \log_2(q')}_{\Delta c} + \lambda \log_2 N + 2\lambda .$$

Comparison between both multiplication checking MPC protocols. In a context where the false positive rate just needs to be negligible compared to $\frac{1}{N}$, the sacrificing-based approach will provide better results: it leads to slighly smaller sizes and a much simpler implementation (it does not require interpolation). In a context where the false positive rate must be close to the security level, the polynomial-based approach will be better: it can achieve very small false positive rate by increasing ν without degrading too much the communication (with the sacrificing-based approach, it would require to take a larger q' but it directly impacts the obtained sizes). So in practice,

- when we want to build an interactive proof of knowledge, we should consider the sacrificing-based verification;
- when we want to build a non-interactive argument (e.g. a signature scheme), we should consider the polynomial-based verification.

6.2.4. Binarity Proof from Masking and Cut-and-Choose Strategy

Our second solution to prove that $[\![x]\!]$ encodes a binary vector relies on a masking of x and a cut-and-choose strategy (see Section 3.3.1). The idea is to generate a random vector r from $\{0,1\}^n$ and to apply the sharing described in Section 6.2.2 to r. In addition, the prover computes (and commits) $\tilde{x} := x \oplus r \in \{0,1\}^n$ where $\oplus$ represents the XOR operation. Instead of giving the shares $[\![x]\!]$ of x as inputs of the MPC protocol, the idea is now to send the shares $[\![r]\!]$ of r. Then using $\tilde{x}$, the parties can locally deduce a sharing of x as

$$[\![x]\!] = (1 - \tilde{x}) \circ [\![r]\!] + \tilde{x} \circ (1 - [\![r]\!])$$

which is a linear relation in $[\![r]\!]$, and the verifier can further deduce the sharing offset Δx from Δr as

$$\Delta x = (1 - \tilde{x}) \circ \Delta r + \tilde{x} \circ (1 - \Delta r) .$$

By replacing $[\![x]\!]$ with $[\![r]\!]$ the parties' input is made independent of the secret. The interest of doing so is to enable a cut-and-choose strategy to prove that $[\![r]\!]$ encodes a binary vector, which in turns implies that $x = \tilde{x} \oplus r$ is a binary vector. More precisely, at the beginning of the zero-knowledge protocol, the prover produces M binary vectors $r^{[\ell]}$ and their corresponding shares $[\![r^{[\ell]}]\!]$ (in practice these vectors and their sharings are pseudo-randomly derived from some seeds). Then the prover commits those sharings $[\![r^{[\ell]}]\!]$ as well as the corresponding masked vectors $\tilde{x}^{[\ell]} := x \oplus r^{[\ell]}$. Then the verifier asks to open all the sharings $r^{[\ell]}$ except one and checks that they correspond to binary vectors. The verifier will hence trust that the unopened sharing encodes also a binary vector with a soundness error of $1/M$. We stress that all the values $\tilde{x}^{[\ell]}$ for which $r^{[\ell]}$ is opened must remain hidden (otherwise x could be readily recovered). The obtained zero-knowledge protocol has a soundness error of

$$\max\left\{\frac{1}{M}, \frac{1}{N}\right\} ,$$

a rejection rate of $1 - (1 - \frac{1}{A})^n$ and a prover-to-verifier communication cost (in bits) of

$$2 \cdot (2\lambda) + \underbrace{\lambda \log_2 M}_{\text{Cost of C\&C}} + \underbrace{n \cdot \log_2(A-1)}_{r - [\![r]\!]_{i^*}} + \underbrace{n}_{\tilde{x}} + \lambda \log_2 N + 2\lambda \ .$$

6.2.5. Asymptotic Analysis

We analyze hereafter the asymptotic complexity of the two variants of our protocol. We show that for a security parameter λ both variants have an asymptotic communication cost of $\Theta(\lambda^2)$ and an asymptotic computation time of $\Theta(\lambda^4)$.

For the binarity proof based on masking and cut-and-choose, we assume $M = N$ (which is optimal for the communication cost given the soundness error). For the other parameters, let us recall that

- for a security parameter λ, one must take $n \approx \log_2 q = \Theta(\lambda)$,
- the prime q' can be chosen as the smallest prime greater than A, which implies $q' \approx A$.

For both variants, the asymptotic communication cost for one repetition of the protocol is then of

$$\Theta(\lambda \log_2 A + \lambda \log_2 N) \ .$$

Since each repetition has a soundness error of $\Theta(1/N)$, the protocol must be repeated $\tau = \Theta(\lambda/\log_2 N)$ times to reach a global soundness error of $2^{-\lambda}$. The probability that any of these τ repetitions aborts is given by

$$1 - \left(1 - \frac{1}{A}\right)^{n \cdot \tau} \approx \frac{n \cdot \tau}{A}$$

where the approximation is tight when A is sufficiently large. Thus for a small constant rejection probability, one must take $A = \Theta(n \cdot \tau) = \Theta(\lambda^2/\log_2 N)$. We have a communication cost for the τ iterations in

$$\Theta\left(\lambda^2 \frac{\log_2 A}{\log_2 N} + \lambda^2\right) = \Theta\left(\frac{\lambda^2}{\log_2 N} \log_2\left(\frac{\lambda^2}{\log_2 N}\right) + \lambda^2\right)$$

and we hence obtain a minimal asymptotic communication cost of $\Theta(\lambda^2)$ by taking $N = \Theta(\lambda)$.

The asymptotic computation time for one repetition of the protocol is of $\Theta(Nn(\log_2 q)(\log_2 A))$, where the term $(\log_2 q)(\log_2 A)$ arises from the complexity of the multiplication between an element of $\mathbb{Z}_q$ and a value smaller than A. We hence get a computation time of $\Theta(\lambda^3 \log_2 \lambda)$ per repetition which makes $\Theta(\lambda^4)$ for τ repetitions.

6.3. Protocols and Security Proofs

In this section, we formally describe our two protocols and state their security. We further introduce a method to decrease the rejection rate.

6.3.1. Protocol with Batch Product Verification

Protocol description. In Section 6.2.3, we proposed two MPC protocols that proves that the sharing $[\![x]\!]$ encodes a binary vector. We then add the checking of the linear relation as described in Section 6.2.2 and we transform the multi-party computation into a zero-knowledge protocol which proves the knowledge of a solution of an SSP instance. We give the formal description of our protocol in Protocol 7: the instructions in blue correspond to those of the sacrificing-based verification, while the instructions in orange correspond to those of the polynomial-based verification. The protocol makes use of a pseudo-random generator PRG, a GGM tree as puncturable PRF, two collision-resistant hash functions Hash_i for $i \in \{1, 2\}$ and a commitment scheme $(\mathsf{Com}, \mathsf{Verif})$.

Security proofs. The following theorems state the completeness, zero-knowledge and soundness of Protocol 7. We refer the reader to [FMRV22a] for the proofs of Theorems 6.3.1, 6.3.2 and 6.3.3.

Theorem 6.3.1 (Completeness)**.** *A prover $\mathcal{P}$ who knows a solution x to the subset sum instance $(w, t) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ and who follows the steps of Protocol 7 convinces the verifier $\mathcal{V}$ with probability*

$$\left(1 - \frac{1}{A}\right)^n .$$

Theorem 6.3.2 (Zero-Knowledge)**.** *Let the PRG used in Protocol 7 be (t, ε_{PRG})-secure and the commitment scheme Com be (t, ε_{Com})-hiding. There exists an efficient simulator $\mathcal{S}$ which outputs a transcript which is $(t, \varepsilon_{PRG} + \varepsilon_{Com})$-indistinguishable from a real transcript of Protocol 7.*

Theorem 6.3.3 (Soundness)**.** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input (w, t), convinces the honest verifier $\mathcal{V}$ on input H, y to accept with probability*

$$\tilde{\varepsilon} := \Pr[\langle \tilde{\mathcal{P}}(w, t), \mathcal{V}(w, t) \rangle = 1] > \varepsilon$$

for a soundness error ε equal to

$$p + \frac{1}{N} - p \cdot \frac{1}{N} ,$$

with

- *p is equal to $\frac{1}{q'}$ when relying on the sacrificing-based verification,*

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$x \in \{0,1\}^n$		
$w \in \mathbb{Z}_q^n, t = \langle w, x\rangle$		w, t
$\mathsf{mseed} \xleftarrow{\$} \{0,1\}^\lambda$		
Compute parties' seeds		
$(\mathsf{seed}_1, \rho_1), \ldots, (\mathsf{seed}_N, \rho_N)$		
with TreePRG(mseed)		
For each party $i \in \{1, \ldots, N\}$:		
$[\![a]\!]_i, [\![x]\!]_i, [\![c]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$		$\triangleright\ a \in \mathbb{Z}_{q'}^n,\ c \in \mathbb{Z}_{q'},\ [\![x]\!]_i \in \{0, \ldots, A-1\}^n$
$[\![a]\!]_i, [\![P]\!]_i, [\![x]\!]_i, [\![c]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$		$\triangleright\ a \in \mathbb{Z}_{q'^\nu},\ P \in \mathbb{Z}_{q'}[X]_{<n-1},$ $c \in \mathbb{Z}_{q'},\ [\![x]\!]_i \in \{0, \ldots, A-1\}^n$
$\mathsf{com}_i = \mathsf{Com}(\mathsf{seed}_i; \rho_i)$		
$\Delta x = x - \sum_i [\![x]\!]_i$		
$\Delta c = \langle a, x\rangle - \sum_i [\![c]\!]_i$		
$\Delta c = a^2 - \sum_i [\![c]\!]_i$		
Compute P as in Equation (6.3)		
$\Delta P = P - \sum_i [\![P]\!]_i$		
$h = \mathrm{Hash}_1(\Delta x, \Delta P, \Delta c, \mathsf{com}_1, \ldots, \mathsf{com}_N)$		
	$\xrightarrow{h}$	
		$\varepsilon \xleftarrow{\$} \mathbb{Z}_{q'}^n$
	$\xleftarrow{\varepsilon}$	
The parties locally set $[\![t]\!] = \langle w, [\![x]\!]\rangle$.		$\triangleright\ t \in \mathbb{Z}_q$
The parties run the MPC protocol.		$\triangleright$ computation in $\mathbb{Z}_{q'}$
$h' = \mathrm{Hash}_2([\![t]\!], [\![\alpha]\!], [\![v]\!])$		
	$\xrightarrow{h'}$	
		$i^* \xleftarrow{\$} \{1, \ldots, N\}$
	$\xleftarrow{i^*}$	
If there exists $j \in [n]$ such that:		
- either $[\![x_j]\!]_{i^*} = 0$ with $x_j = 1$		
- or $[\![x_j]\!]_{i^*} = A - 1$ with $x_j = 0$,		
then abort.		
$y = x - [\![x]\!]_{i^*}$		
	$(\mathsf{seed}_i, \rho_i)_{i \neq i^*},\ \mathsf{com}_{i^*},$ $y,\ \Delta P,\ \Delta c,\ \alpha$ $\longrightarrow$	
		For all $i \neq i^*$,
		$[\![a]\!]_i, [\![x]\!]_i, [\![c]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$
		$[\![a]\!]_i, [\![P]\!]_i, [\![x]\!]_i, [\![c]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$
		$\Delta x = y - \sum_{i \neq i^*} [\![x]\!]_i$
		$\Delta\alpha = \varepsilon \cdot (1 - \Delta x)$
		$\Delta S(r) = \sum_i \Delta x_i \prod_{j \neq i} \left(\frac{r - \gamma_j}{\gamma_i - \gamma_j}\right)$
		$\Delta\alpha = \varepsilon \cdot \Delta S(r)$
		For all $i \neq i^*$,
		Rerun the party i as the prover
		and compute the commitment com_i.
		$\Delta t = \langle w, \Delta x\rangle$
		$\Delta v = \langle \alpha, \Delta x\rangle - \Delta c$
		$\Delta v = \varepsilon^2((\Delta P)(r) \prod_i (r - \gamma_i) + \Delta S(r))$ $-\alpha \cdot \varepsilon \cdot \Delta S(r) - \Delta c$
		$[\![\alpha]\!]_{i^*} = \alpha - \sum_{i \neq i^*} [\![\alpha]\!]_i$
		$[\![t]\!]_{i^*} = t - \Delta t - \sum_{i \neq i^*} [\![t]\!]_i$
		$[\![v]\!]_{i^*} = -\Delta v - \sum_{i \neq i^*} [\![v]\!]_i$
		Check $h = \mathrm{Hash}_1(\Delta x, \Delta c, \mathsf{com}_1, \ldots, \mathsf{com}_N)$
		Check $h' = \mathrm{Hash}_2([\![t]\!], [\![\alpha]\!], [\![v]\!])$
		Return Success

Protocol 7: Zero-knowledge argument for Subset Sum Problem via MPC-in-the-head with rejection, using batch product verification to prove binarity (using sacrificing-based verification in blue, using polynomial-based verification in orange).

- *p is equal to $\frac{2n-1}{q'^{\nu}} + (1 - \frac{2n-1}{q'^{\nu}})\frac{2}{q'^{\nu}}$ when relying on the polynomial-based verification.*

Then, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given rewindable black-box access to $\tilde{\mathcal{P}}$, produces either a witness x such that $t = \langle w, x\rangle$ and $x \in \{0,1\}^n$, or a commitment collision, by making an average number of calls to $\tilde{\mathcal{P}}$ which is upper bounded by

$$\frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{2 \cdot \ln(2)}{\tilde{\varepsilon} - \varepsilon}\right) .$$

Proof size. To achieve a targeted soundness error $2^{-\lambda}$, we can perform τ parallel executions of the protocol such that $\varepsilon^{\tau} \leq 2^{-\lambda}$. Such parallel repetition does not preserve (general) zero-knowledge and the resulting scheme achieves *honest verifier* zero knowledge. And instead of sending τ values for h and h', the prover can merge them together to send a single h and a single h'. Moreover, instead of sending the $N-1$ seeds and commitment randomness of $(\mathsf{seed}_i, \rho_i)_{i \neq i^*}$ for each execution, we can send the sibling path from $(\mathsf{seed}_{i^*}, \rho_{i^*})$ to the tree root in the GGM tree, it costs at most $\lambda \cdot \log_2(N)$ bits (we need to reveal $\log_2(N)$ nodes of the tree) by execution. The communication cost (in bits) of the protocol with τ repetitions is (with $\nu = 1$ when considering the sacrificing-based polynomial)

$$\textsc{Size} = 4\lambda + \tau \cdot \left[n \cdot (\log_2(A-1) + \log_2(q')) + (2\nu - 1)\log_2(q') + \lambda \log_2 N + 2\lambda\right]$$

while the soundness error and rejection rate scale as

$$\left(p + \frac{1}{N} - p \cdot \frac{1}{N}\right)^{\tau} \quad \text{and} \quad 1 - \left(1 - \frac{1}{A}\right)^{\tau \cdot n}$$

respectively, with p defined as in Theorem 6.3.3. Let us stress that the obtained size is independent of the modulus q (and of the size of the integers $\{w_j\}, t$).

6.3.2. Protocol with Cut-and-Choose Strategy

Protocol description. As described in Section 6.2.4, we can also use a cut-and-choose strategy to prove that the vector $[\![x]\!]$ is binary. It is possible since we can remplace the input $[\![x]\!]$ of the multi-party computation by a sharing $[\![r]\!]$ *independent* of the secret, where r is a mask uniformly sampled in $\{0,1\}^n$. To achieve a targeted soundness error $2^{-\lambda}$, we can perform τ parallel executions of the protocol such that $\varepsilon^{\tau} \leq 2^{-\lambda}$. Like [KKW18], instead of performing τ independent cut-and-choose phases each resulting in trusting one sharing $[\![r]\!]$ among M, we can perform a global cut-and-choose phase resulting in τ trusted sharings $[\![r]\!]$ among a larger M (see [KKW18] for more details). We give the formal description of this zero-knowledge protocol in Protocol 8. The protocol makes use of a pseudo-random generator PRG, a GGM tree as puncturable PRF, four collision-resistant hash functions Hash_i for $i \in \{1,2,3,4\}$ and a commitment scheme $(\mathsf{Com}, \mathsf{Verif})$.

Security proofs. The following theorems state the completeness, zero-knowledge and soundness of Protocol 8. We refer the reader to [FMRV22a] for the proofs of Theorems 6.3.4, 6.3.5 and 6.3.6.

Theorem 6.3.4 (Completeness)**.** *A prover $\mathcal{P}$ who knows a solution x to the subset sum instance $(w,t) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ and who follows the steps of Protocol 8 convinces the verifier $\mathcal{V}$ with probability*

$$\left(1 - \frac{1}{A}\right)^{\tau \cdot n} .$$

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$x \in \{0,1\}^n$		
$w \in \mathbb{Z}_q^n, t = \langle w, x \rangle$		w, t
$\mathsf{mseed}^{[0]} \leftarrow \{0,1\}^\lambda$		
$(\mathsf{mseed}^{[e]})_{e\in[M]} \leftarrow \mathrm{TreePRG}(\mathsf{mseed}^{[0]})$		
For each $e \in \{1,\dots,M\}$:		
$r^{[e]} \leftarrow \mathrm{PRG}(\mathsf{mseed}^{[e]})$		$\triangleright\ r^{[e]} \in \{0,1\}^n$
$(\mathsf{seed}_i^{[e]}, \rho_i^{[e]})_{i\in[1:N]} \leftarrow \mathrm{TreePRG}(\mathsf{mseed}^{[e]})$		
For each $i \in \{1,\dots,N\}$:		
$[\![r^{[e]}]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i^{[e]})$		$\triangleright\ [\![r^{[e]}]\!]_i \in \{0,\dots,A-1\}^n$
$\mathsf{com}_i^{[e]} = \mathsf{Com}(\mathsf{seed}_i^{[e]}; \rho_i^{[e]})$		
$\Delta r^{[e]} = r^{[e]} - \sum_i [\![r^{[e]}]\!]_i$		
$h_e = \mathrm{Hash}_1(\Delta r^{[e]}, \mathsf{com}_1^{[e]}, \dots, \mathsf{com}_n^{[e]})$		
$h = \mathrm{Hash}_2(h_1, \dots, h_M)$		
	$\xrightarrow{h}$	
		$J \leftarrow \{J \subset [M]\ ;\ \lvert J\rvert = \tau\}$
	$\xleftarrow{J}$	
For each $e \in J$:		
$\tilde{x}^{[e]} = x \oplus r^{[e]}$		$\triangleright\ \oplus$ is the XOR operation ($\tilde{x} \in \{0,1\}^n$)
The parties locally set		
$[\![x^{[e]}]\!] = (1-\tilde{x}^{[e]}) \circ [\![r^{[e]}]\!] + \tilde{x}^{[e]} \circ (1 - [\![r^{[e]}]\!])$		
and they set $[\![t^{[e]}]\!] = \langle w, [\![x^{[e]}]\!] \rangle$.		
$h'_e = \mathrm{Hash}_3(\tilde{x}^{[e]}, [\![t^{[e]}]\!])$		
$h' = \mathrm{Hash}_4((h'_e)_{e\in J})$		
	$\xrightarrow{h',\ (\mathsf{mseed}^{[e]})_{e\in[M]\setminus J}}$	
		$L = \{\ell_e\}_{e\in J} \leftarrow \{1,\dots,N\}^\tau$
	$\xleftarrow{L}$	
If there exists $(e,j) \in J \times [n]$ such that:		
- either $[\![r_j^{[e]}]\!]_{\ell_e} = 0$ with $r_j^{[e]} = 1$		
- or $[\![r_j^{[e]}]\!]_{\ell_e} = A-1$ with $r_j^{[e]} = 0$,		
then abort.		
$y = r^{[e]} - [\![r^{[e]}]\!]_{\ell_e}$		
	$\xrightarrow{\left(\begin{array}{c}(\mathsf{seed}_i^{[e]}, \rho_i^{[e]})_{i\neq\ell_e} \\ y, \tilde{x}^{[e]},\ \mathsf{com}_{\ell_e}^{[e]}\end{array}\right)_{e\in J}}$	
		For each $e \notin J$:
		Compute h_e using $\mathsf{mseed}^{[e]}$
		For each $e \in J$:
		For all $i \neq \ell_e$
		$\mathsf{com}_i^{[e]} = \mathsf{Com}(\mathsf{seed}_i^{[e]}; \rho_i^{[e]})$
		Rerun the party i as the prover to get $[\![t^{[e]}]\!]_i$
		$\Delta r^{[e]} = y - \sum_{i\neq\ell_e} [\![r^{[e]}]\!]$
		$h_e = \mathrm{Hash}_1(\Delta r^{[e]}, \mathsf{com}_1^{[e]}, \dots, \mathsf{com}_n^{[e]})$
		From $\Delta r^{[e]}$, deduce $\Delta t^{[e]}$.
		$[\![t^{[e]}]\!] = t - \Delta t^{[e]} - \sum_{i\neq\ell_e} [\![t^{[e]}]\!]_i$
		$h'_e = \mathrm{Hash}_3(\tilde{x}^{[e]}, [\![t^{[e]}]\!])$
		Check $h = \mathrm{Hash}_2(h_1, \dots, h_M)$
		Check $h' = \mathrm{Hash}_4((h'_e)_{e\in J})$
		Return Success

Protocol 8: Zero-knowledge argument for Subset Sum Problem via MPC-in-the-head with rejection, using cut-and-choose strategy to prove binarity.

Theorem 6.3.5 (Honest-Verifier Zero-Knowledge)**.** *Let the PRG used in Protocol 8 be (t, ε_{PRG})-secure and the commitment scheme* Com *be (t, ε_{Com})-hiding. There exists an efficient simulator $\mathcal{S}$ which, given random challenges J and L outputs a transcript which is $(t, \tau \cdot \varepsilon_{PRG} + \tau \cdot \varepsilon_{Com})$-indistinguishable from a real transcript of Protocol 8.*

Theorem 6.3.6 (Soundness)**.** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input (w, t), convinces the honest verifier $\mathcal{V}$ on input H, y to accept with probability*

$$\tilde{\varepsilon} := \Pr[\langle \tilde{\mathcal{P}}(w,t), \mathcal{V}(w,t) \rangle = 1] > \varepsilon$$

for a soundness error ε equal to

$$\max_{M-\tau \leq k \leq M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau} \cdot N^{k-M+\tau}} \right\} .$$

Then, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given rewindable black-box access to $\tilde{\mathcal{P}}$, produces either a witness x such that $t = \langle w, x \rangle$ and $x \in \{0,1\}^n$, or a commitment collision, by making an average number of calls to $\tilde{\mathcal{P}}$ which is upper bounded by

$$\frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{8 \cdot M}{\tilde{\varepsilon} - \varepsilon}\right) .$$

Proof size. Let us recall that the couples $(\mathsf{seed}_i, \rho_i)$ are sampled using a GGM seed tree, sending $(\mathsf{seed}_i^{[e]}, \rho_i^{[e]})_{i \neq \ell_e}$ costs at most $\lambda \cdot \log_2(N)$ bits by iteration. The communication cost (in bits) of the protocol is then

$$\textsc{Size} = 4\lambda + \lambda \cdot \tau \cdot \log_2 \frac{M}{\tau} + \tau \cdot [n \cdot \log_2(A-1) + n + \lambda \log_2 N + 2\lambda] .$$

Here again, the obtained size is independent of the modulus q (and of the size of the integers $\{w_j\}, t$).

6.3.3. Decreasing the Rejection Rate

The two above protocols have a rejection rate around $\tau n / A$ which implies that we must take $A = \Theta(\tau n)$ to obtain a constant (small) rejection rate. In practice, this results in a significant increase in the communication cost. Let us for instance consider Protocol 7 with $(\tau, N, A) = (16, 280, 2^{13})$ using $(n, q) = (256, q^{256})$. For this setting, the proof size is about 15.6 KB for a rejection rate of 0.394. If we want a rejection rate below 0.003 (by increasing A), we should take $A = 2^{21}$ and the proof size would be 23.6 KB.

A better strategy consists in allowing the prover to abort a few of the τ iterations. Let us assume that the verifier accepts the proof if the prover can answer to $\tau - \eta$ challenges among the τ iterations. This slightly increases the soundness error, but it can also significantly decrease the global rejection rate. If we denote p_{rej} the probability that an iteration aborts, then the global rejection rate of this strategy is given by

$$1 - \sum_{i=0}^{\eta} \binom{\tau}{i} \cdot (1 - p_{\text{rej}})^{\tau - i} \cdot p_{\text{rej}}^{i} . \tag{6.4}$$

At the same time, the soundness error for Protocol 7 becomes

$$\sum_{i=0}^{\eta} \binom{\tau}{i} \cdot (1-\varepsilon)^i \cdot \varepsilon^{\tau-i}$$

where $\varepsilon = \frac{1}{N} + p - p \cdot \frac{1}{N}$ is the soundness error of a single iteration. Using this strategy with $\tau = 20$ and $\eta = 3$, the proof size is of 16.7 KB for a rejection rate of 0.003 (instead of 23.6 KB with the naive strategy).

The same strategy also applies to Protocol 8. The rejection rate is also given by Equation Equation (6.4) while the soundness error becomes

$$\max_{M-\tau \le k \le M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau}} \cdot \sum_{i=0}^{\eta} \left[\binom{k-M+\tau}{i} \left(1-\frac{1}{N}\right)^i \left(\frac{1}{N}\right)^{k-M+\tau-i} \right] \right\}.$$

In any case, the prover always answers to at most $\tau - \eta$ challenges of the verifier (even if the prover aborts less than η among the τ iterations) so that the communication cost is roughly that of $\tau - \eta$ iterations. Additionally, for each unanswered challenge, the prover must further send two hash digests to enable the verifier to recompute and check h and h'. Thus the new proof size (in bits) for Protocol 7 is

$$\textsc{Size}_\eta = 4\lambda + \eta \cdot 4\lambda + (\tau - \eta) \cdot \left[n \cdot (\log_2(A-1) + \log_2(q')) + (2\nu - 1)\log_2(q') + \lambda \log_2 N + 2\lambda \right],$$

while the new proof size (in bits) for Protocol 8 is

$$\textsc{Size}_\eta = 4\lambda + \eta \cdot 4\lambda + \lambda \cdot \tau \cdot \log_2 \frac{M}{\tau} + (\tau - \eta) \cdot \left[n \cdot \log_2(A-1) + n + \lambda \log_2 N + 2\lambda \right].$$

We note that in practice, given a target security level and a target rejection probability, one needs to use a slightly increased τ (or N) to compensate for the loss in terms of soundness. While this shall slightly increase the proof size, the above approach (with $\eta > 0$) still provides better trade-offs than the original approach ($\eta = 0$).

6.4. Instantiations and Performance

6.4.1. Subset Sum Instances

We recall in this section known techniques to solve the modular subset sum problem (SSP) defined by Equation (6.1). It is well-known that the hardness of an SSP instance depends greatly on its *density* defined as $d = n/\log_2 q$. If the SSP instance is too sparse (e.g. $d < 1/n$) or too dense (e.g. $d > n/\log^2 n$) then the problem can be solved in polynomial time (see e.g. [CJL+92] and references therein). We shall therefore only consider SSP instances with density $d \simeq 1$ (i.e. $q \simeq 2^n$) which are arguably the hardest ones [IN96].

In this case, simple algorithms exist based on brute force enumeration at $O(2^n)$ time and constant space, or time-space tradeoff [HS74] with $O(2^{n/2})$ time and space complexities. The first non-trivial algorithm was published by Schroeppel and Shamir [SS81] with time complexity $O(2^{n/2})$ and space complexity $O(2^{n/4})$. Later, faster algorithms were proposed with similar time and space complexities, e.g. $\tilde{O}(2^{0.337n})$ by Howgrave-Graham and Joux [HJ10] and $\tilde{O}(2^{0.283n})$ by Bonnetain, Bricout, Schrottenloher and Shen [BBSS20]. The latter algorithms neglect the cost to access an exponential memory but even with this optimistic

assumption, for $n = 256$, all known algorithms require at least a time complexity lower-bounded by 2^{128} operations or memory of size at least 2^{72} bits. There also exists a vast literature on quantum algorithms for solving the SSP (see [BBSS20] and references therein). The best (heuristic) quantum complexity from [BBSS20] has time complexity $\tilde{O}(2^{0.216n})$ and thus requires about 2^{64} quantum operations and quantum memory for $n = 256$. In the following, we, therefore, consider the efficiency of our protocols for $n = 256$.

6.4.2. Zero Knowledge Protocols

Let us consider the subset sum problem with $n = 256$. We propose in Table 6.1 several sets of parameters for our two protocols which target a security of 128 bits. We provide two kinds of instantiations to give the reader an idea of the obtained performance while changing the number of parties. The first ones correspond to instantiations with fast computation. The second ones correspond to instantiations that achieve smaller communication costs but slower computation. For each setting, we suggest two parameter sets: one achieving a rejection rate around 0.4 and the other one achieving a rejection rate between 0.001 and 0.004.

Table 6.1.: Comparison of state-of-the-art zero-knowledge protocols for proving the knowledge of an SSP instance (with $n = 256$ and $q \approx 2^{256}$).

Protocol	Parameters						Proof size	Rej. rate
	τ	η	N	A	M	ν		
Shamir [Sha86]	219	-	-	-	-	-	1186 KB	-
[LNSW13]	219	-	-	-	-	-	2350 KB	-
Beullens [Beu20]	14	-	1024	-	4040	-	122 KB	-
Prot. 7 (batching sacr.)	26	0	32	2^{14}	-	-	25.7 KB	0.334
Prot. 7 (batching sacr.)	31	3	32	2^{14}	-	-	27.9 KB	0.001
Prot. 7 (batching poly.)	26	0	32	2^{14}	-	2	25.8 KB	0.334
Prot. 7 (batching poly.)	31	3	32	2^{14}	-	2	28.0 KB	0.001
Prot. 8 (C&C)	27	0	32	2^{14}	462	-	17.4 KB	0.344
Prot. 8 (C&C)	33	3	32	2^{14}	470	-	19.6 KB	0.002
Prot. 7 (batching sacr.)	17	0	256	2^{13}	-	-	16.6 KB	0.412
Prot. 7 (batching sacr.)	21	3	256	2^{13}	-	-	17.7 KB	0.004
Prot. 7 (batching poly.)	17	0	256	2^{13}	-	2	16.6 KB	0.412
Prot. 7 (batching poly.)	21	3	256	2^{13}	-	2	17.8 KB	0.004
Prot. 8 (C&C)	19	0	256	2^{13}	954	-	13.0 KB	0.448
Prot. 8 (C&C)	24	3	256	2^{14}	952	-	15.4 KB	0.001

We provide in Table 6.1 the performance of the other zero-knowledge protocols proving the knowledge of an SSP solution. The only other protocol designed for the subset sum problem is Shamir's one [Sha86]. We can also compare these protocols with [LNSW13] which is an adaptation of Stern's protocol to the ISIS (inhomogeneous short integer solution) problem. The remaining articles in the literature about proofs for the ISIS problem are restricted to the case where the modulus q is prime. We add Beullens' protocol [Beu20] for ISIS with prime q to the comparison.

6.4.3. Signature Schemes with Subset Sum Problem

For each of our two protocols, we explain how to apply the Fiat-Shamir transform [FS87] to get signature schemes.

Signature from Protocol 7. We compute the challenges $\{\varepsilon^{[e]}\}_{e\in[\tau]}$ and $\{i^{*[e]}\}_{e\in[\tau]}$ for τ executions as:

$$\{\varepsilon^{[e]}\}_{e\in[\tau]} := \mathrm{Hash}'_1(m,h)$$

and

$$\{i^{*[e]}\}_{e\in[\tau]} := \mathrm{Hash}'_2(m,h,h')$$

where m is the message to sign, Hash'_1 and Hash'_2 are some hash functions, and h (resp. h') is the hash value corresponding to the merged inputs of Hash_1 (resp. Hash_2) from the τ executions.

Since the protocol has 5 rounds, we must take into account the forgery attack described in [KZ20a] to estimate the security of the resulting signature. When we adapt the attack for Protocol 7, its cost is given by

$$\mathrm{cost}_{\mathrm{forge}} = \min_{\tau_1,\tau_2:\tau_1+\tau_2=\tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \mathrm{PMF}(i,\tau,\frac{1}{q'})} + \frac{1}{\sum_{i=0}^{\eta} \mathrm{PMF}(i,\tau_2,1-\frac{1}{N})} \right\},$$

with $\mathrm{PMF}(i,\tau,p) := \binom{\tau}{i} p^i (1-p)^{\tau-i}$. When selecting the signature parameters, we must choose τ such that $\mathrm{cost}_{\mathrm{forge}} \geq 2^\lambda$.

Signature from Protocol 8. The challenges J and L are computed as

$$J := \mathrm{Hash}'_1(m,h)$$

and

$$L := \mathrm{Hash}'_2(m,h,h',(\mathsf{mseed}^{[j]})_{j\in[M]\setminus J})$$

where m is the message to sign and where Hash'_1 and Hash'_2 are some hash functions.

Since the protocol has 5 rounds, the security of the resulting signature scheme is given by the attack of [KZ20a] which has, in the context of the Protocol 8, a forgery cost of

$$\mathrm{cost}_{\mathrm{forge}} = \min_{M-\tau\leq k\leq M} \left\{ \frac{\binom{M}{M-\tau}}{\binom{k}{M-\tau}} + \frac{1}{\sum_{i=0}^{\eta} \mathrm{PMF}(i,k-M+\tau,1-\frac{1}{N})} \right\}.$$

Another approach consists in turning the 5-round protocol into a 3-round protocol (before applying the Fiat-Shamir). We refer to [KKW18; FJR23] for the details of such an approach. The soundness error of this variant is the same as for the original protocol (see Theorem 6.3.6). When we apply the Fiat-Shamir to this variant, the security of the obtained signature scheme is equal to the soundness security of the protocol (since the protocol has now only 3 rounds) and its size (in bits) is

$$\textsc{Size}_\eta = 4\lambda + \eta\cdot 4\lambda + 3\lambda\cdot\tau\cdot\log_2\frac{M}{\tau} + (\tau-\eta)\cdot[n\cdot\log_2(A-1)+n+\lambda\log_2 N+2\lambda].$$

Performance. We selected some parameter sets to instantiate the resulting signature schemes while targeting a security of 128 bits and a rejection rate of 0.01. We obtained the performance of Table 6.2.

Table 6.2.: Performance of the obtained signatures. The public keys (w, t) of those schemes have a size of $\frac{\lambda+n}{8} = 48$ bytes, since w can be derived from a λ-bit seed and t is an integer of $\log_2 q \approx n$ bits.

Signature	Parameters						Sig. size	Rej. rate
	τ	η	N	A	M	ν		
Protocol 7 (batching sacr.)	29	2	256	2^{14}	-	-	28.1 KB	0.010
Protocol 7 (batching sacr.)	42	3	32	2^{14}	-	-	38.7 KB	0.004
Protocol 7 (batching poly.)	20	2	256	2^{14}	-	6	19.1 KB	0.004
Protocol 7 (batching poly.)	32	3	32	2^{14}	-	6	29.3 KB	0.002
Protocol 8 (C&C), 5 rounds	46	3	256	2^{14}	993	-	30.3 KB	0.006
Protocol 8 (C&C), 5 rounds	71	3	32	2^{14}	452	-	42.5 KB	0.025
Protocol 8 (C&C), 3 rounds	28	2	64	2^{14}	514	-	21.1 KB	0.009
Protocol 8 (C&C), 3 rounds	53	3	8	2^{14}	253	-	33.2 KB	0.009

6.5. Digital Signatures from Boneh-Halevi-Howgrave-Graham PRF

As illustrated on the subset sum problem, our technique of sharing over the integers with rejection is – more generally – instrumental to a context of a secret vector $s \in \mathbb{Z}_q^n$ with small coefficients. Since the communication cost of our protocols is independent of the size q of the ring $\mathbb{Z}_q$, the gain in communication is higher when the modulus q is high. But it does not need to have a modulus as high as in the subset sum problem to be interesting. In what follows, we present another application of our technique: a short and efficient candidate post-quantum signature scheme based on an elegant pseudo-random function (PRF) proposed by Boneh, Halevi, and Howgrave-Graham in 2001 [BHH01].

Let p be a public m-bit prime number that defines the PRF message space as $\mathbb{Z}_p$. A secret key for the PRF is an element $x \in \mathbb{Z}_p$ picked uniformly at random. We denote $\mathrm{MSB}_\delta(t)$ the δm most significant bits of an m-bit element $t \in \mathbb{Z}_p$.[2] The value of the PRF on the message $m \in \mathbb{Z}_p$ for the secret-key $x \in \mathbb{Z}_p$ is $F_x(m) = \mathrm{MSB}_\delta((x+m)^{-1} \bmod p)$.

Our signature scheme follows the blueprint of most signatures based on the MPCitH paradigm since the proposal of Picnic [CDG+17]: the public key is made of the outputs of Boneh *et al.*'s PRF on t public messages in $\{1, \ldots, t\}$, i.e. the δm-bit elements $y_1, \ldots, y_t$ such that

$$y_i := \mathrm{MSB}_\delta((x+i)^{-1} \bmod p) \text{ for } i \in \{1, \ldots, t\}$$

and the signature consists of a non-interactive proof of knowledge of $x, z_1, \ldots, z_t$ (parametrized by the signed message using the Fiat-Shamir heuristic) such that

$$(x+1)(2^{(1-\delta)m} y_1 + z_1) \equiv \cdots \equiv (x+t)(2^{(1-\delta)m} y_t + z_t) \equiv 1 \bmod p \tag{6.5}$$

$$\text{and} \qquad z_1, \ldots, z_t \in \{0, \ldots, 2^{(1-\delta)m} - 1\} \tag{6.6}$$

[2] We assume hereafter that $\delta m \in \mathbb{Z}$. Otherwise, one should take the nearest integer $\lfloor \delta m \rceil$ instead.

where $z_1, \ldots, z_t$ are the $(1-\delta)m$ least significant bits of $(x+1)^{-1} \bmod p, \ldots, (x+t)^{-1} \bmod p$. Note that the condition (6.6) on the size of the z_i's is fundamental since otherwise, it is easy for an attacker to find a witness.

In our applications, the values of t and δ are chosen to prevent all known classical attacks and target a 128-bit security level.

Let us fix t, the number of outputs of the PRF. Then, to ensure that the equations (6.5) and (6.6) have a unique witness, we add the constraint $\delta \geq 1/t$ so that the t PRF outputs define (heuristically) the secret x uniquely. To avoid brute-force attacks from a single output of the PRF, at least 128 bits should remain hidden for each output, thus $m := \log p \geq \frac{128}{1-\delta}$. Otherwise, an attacker could reconstruct a possible PRF key matching with the first output and then test it by evaluating the other outputs with this candidate.

It is possible to apply generically the MPCitH paradigm to prove (6.5) and (6.6), but proving (6.6) seems inefficient (e.g. by using a binary decomposition and proving consistency). Instead, we can use our secret sharing over the integers for proving the knowledge of small z_i's by sharing them as a sum of "small" integers which directly proves that the z_i's are indeed small.

Proving Equation (6.5). Instead of proving the t products of (6.5) separately, the prover can batch them into a linear combination where coefficients $\gamma_1, ..., \gamma_t$ are provided by the verifier, *i.e.* the prover proves the equation

$$\sum_{i=1}^{t} \gamma_i \cdot \Big((x+i)(2^{(1-\delta)m} y_i + z_i) - 1\Big) = 0 \bmod p,$$

or equivalently,

$$x \cdot \left(\sum_{i=1}^{t} \gamma_i z_i\right) = -\sum_{i=1}^{t} \gamma_i \left(x \cdot 2^{(1-\delta)m} y_i + i \cdot 2^{(1-\delta)m} y_i + i \cdot z_i - 1\right) \bmod p. \tag{6.7}$$

If one of the products is not equal to 1 in (6.5), then (6.7) is satisfied only with a probability of $\frac{1}{p}$. And to prove (6.7), one can use the [BN20] protocol with a single multiplication on $\mathbb{Z}_p$ (for the left-hand side of (6.7), the right-hand side being a linear combination of the witness). The resulting MPC protocol produces false positives with probability $1/p + (1-1/p) \cdot 1/p := 2/p - 1/p^2$, and thus the obtained zero-knowledge argument has a soundness error of

$$\varepsilon = \frac{1}{N} + \left(1 - \frac{1}{N}\right)\left(\frac{2}{p} - \frac{1}{p^2}\right).$$

Proving Equation (6.6). It remains to prove that z_i is in $\{0, \ldots, B-1\}$ with $B = 2^{(1-\delta)m}$ in (6.6) for $i \in \{1, \ldots, t\}$. To share z_i, we use our secret sharing over the integers of Section 6.2.2. Since the z_i are not binary but in a larger range, we need to adapt the rejection rules. Following exactly the same reasoning as in Section 6.2.2, we get that the prover must abort if there exists an index $j \in [t]$ for which $z_j - [\![z_j]\!]_{i^*} \geq 1$ or $z_j - [\![z_j]\!]_{i^*} \leq -A + B - 1$. The resulting rejection rate is given by

$$p_{\text{rej}} = 1 - \left(1 - \frac{B-1}{A}\right)^{t \cdot \tau} \approx t \cdot \tau \cdot \frac{B-1}{A}.$$

Even without proving anything on the range of z_j, the verifier knows that

$$\forall j \in [t], -A + B \leq z_j \leq A - 1$$

thanks to Equation (6.2) (generalized). In practice, we settle for this range, implying that there is a slack between the underlying hard problem and the proven statement. A malicious prover can use bigger values for z_i, and this is equivalent to ignoring some bits of y_i. A malicious prover can ignore up to $\log_2 \frac{A}{B} \approx \log_2 \frac{t \cdot \tau}{p_{\text{rej}}}$ bits for each PRF output, and thus it reduces the security of $t \cdot \log_2 \frac{t \cdot \tau}{p_{\text{rej}}}$ bits.

A way to fix this security loss without increasing the size of p (and of the key) is to reveal a few more PRF outputs to guarantee that the key is still heuristically unique. In theory, this decreases the security but for state-of-the-art algorithms, this stays beyond the capacity of the best-known algorithms for small t. In fact, we need to reveal $\tilde{t} \geq t$ outputs of the PRF such that

$$\tilde{t} \cdot \delta \cdot m - \tilde{t} \cdot \log \left(\frac{\tilde{t} \cdot \tau}{p_{\text{rej}}} \right) > m.$$

In other words, we replace the constraint $\delta \geq \frac{1}{t}$ by

$$\delta \geq \frac{1}{\tilde{t}} + \frac{1}{m} \log_2 \left(\frac{\tilde{t} \cdot \tau}{p_{\text{rej}}} \right).$$

This leads to the scheme described as Protocol 9 with the communication cost (in bits):

$$4\lambda + \tau \cdot (\underbrace{\log_2 p}_{\Delta x} + \underbrace{\tilde{t} \cdot \log_2 A}_{\Delta a_i} + \underbrace{\log_2 p}_{\Delta c} + \underbrace{\log_2 p}_{\Delta \alpha} + \lambda \cdot \log_2 N + 2\lambda),$$

with soundness error (if interactive)

$$\varepsilon = \frac{1}{N} + p' \cdot \left(1 - \frac{1}{N}\right),$$

and with forgery security (if non-interactive)

$$\text{cost}_{\text{forge}} = \min_{\tau_1, \tau_2 : \tau_1 + \tau_2 = \tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} p'^i (1-p')^{\tau - i}} + N^{\tau_2} \right\},$$

with $p' := 2/p + 1/p^2$.

We propose in Table 6.3 some parameters which target 128-bit security (based on the hardness of the so-called *modular inverse hidden number problem*) according to the current cryptanalysis state-of-the-art for Boneh *et al.*'s PRF. We can remark that the achieved signature sizes are competitive with Rainier scheme [DKR+21] (which can produce signatures that are around 5 KB in size too) and outperform all the other signatures based on MPC-in-the-Head paradigm (Picnic4 [KZ22], PorcRoast [BD20], SDitH [FJR22b], ...).

Regarding the cryptanalysis, the security of Boneh *et al.*'s PRF has been extensively analyzed since 20 years [BHH01; LSSW12; BVZ12; XSH+19] and relies strongly on δ and the number of known PRF outputs. The first natural attack is the brute-force search on one output of the PRF as explained previously. We choose our parameter sets such that

$$(1 - \delta)m > 128 \tag{6.8}$$

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
$x \in \{0,1\}^n$		
$(z_1, y_1), \ldots, (z_t, y_t)$		$y_1, \ldots, y_t$
$\mathsf{mseed} \xleftarrow{\$} \{0,1\}^\lambda$		
Compute parties' seeds		
$(\mathsf{seed}_1, \rho_1), \ldots, (\mathsf{seed}_N, \rho_N)$		
with TreePRG(mseed)		
For each party $i \in \{1, \ldots, N\}$:		
$[\![x]\!]_i, [\![a]\!]_i, [\![c]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$		$\triangleright$ $[\![x]\!], [\![a]\!]_i, [\![c]\!]_i \in \mathbb{Z}_p$
$[\![\vec{z}]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$		$\triangleright$ $[\![\vec{z}]\!]_i \in \{0, \ldots, A-1\}^t$
$\mathsf{com}_i = \mathsf{Com}(\mathsf{seed}_i; \rho_i)$		
$\Delta x = x - \sum_i [\![x]\!]_i$		
$\Delta c = a \cdot x - \sum_i [\![c]\!]_i$		
$\Delta \vec{z} = \vec{z} - \sum_i [\![\vec{z}]\!]_i$		
$h = \mathrm{Hash}_1(\Delta x, \Delta c, \Delta \vec{z}, \mathsf{com}_1, \ldots, \mathsf{com}_N)$		
	$\xrightarrow{h}$	
		$\gamma_1, \ldots, \gamma_t, \varepsilon \xleftarrow{\$} \mathbb{Z}_p$
	$\xleftarrow{\vec{\gamma}, \varepsilon}$	
The parties locally set		
- $[\![\alpha]\!] = \varepsilon \cdot \langle \vec{\gamma}, [\![\vec{z}]\!] \rangle + [\![a]\!] \bmod p$		
- $[\![r]\!] =$ "right part of Equation 6.7"		
The parties open $[\![\alpha]\!]$ to get α.		
The parties locally set		
$[\![v]\!] = \varepsilon \cdot [\![r]\!] - \alpha \cdot [\![x]\!] + [\![c]\!] \bmod p$		
$h' = \mathrm{Hash}_2([\![\alpha]\!], [\![v]\!])$		
	$\xrightarrow{h'}$	
		$i^* \xleftarrow{\$} \{1, \ldots, N\}$
	$\xleftarrow{i^*}$	
$\vec{\mu} = \vec{z} - [\![\vec{z}]\!]_{i^*}$		
If there exists $j \in [t]$ such that:		
- either $\mu_j \geq 1$		
- or $\mu_j \leq -A + B - 1$,		
then abort.		
	$\xrightarrow{(\mathsf{seed}_i, \rho_i)_{i \neq i^*},\ \mathsf{com}_{i^*},\ \vec{\mu},\ \Delta c,\ [\![\alpha]\!]_{i^*}}$	
		For all $i \neq i^*$,
		$[\![x]\!]_i, [\![a]\!]_i, [\![c]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$
		$[\![\vec{z}]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i)$
		$\Delta \vec{z} = \vec{\mu} - \sum_{i \neq i^*} [\![\vec{z}]\!]_i$
		$\Delta \alpha = \varepsilon \cdot \langle \vec{\gamma}, \Delta \vec{z} \rangle$
		For all $i \neq i^*$,
		Rerun the party i as the prover
		and compute the commitment com_i.
		$\Delta r =$ deduces from the right part of Eq 6.7
		$\Delta v = \varepsilon \cdot \Delta r - \alpha \cdot \Delta x - \Delta c$
		$[\![v]\!]_{i^*} = -\Delta v - \sum_{i \neq i^*} [\![v]\!]_i$
		Check $h = \mathrm{Hash}_1(\Delta x, \Delta c, \Delta \vec{z}, \mathsf{com}_1, \ldots, \mathsf{com}_N)$
		Check $h' = \mathrm{Hash}_2([\![\alpha]\!], [\![v]\!])$
		Return Success

Protocol 9: Relaxed zero-knowledge argument for Boneh et al's PRF.

Table 6.3.: Parameter sets and achieved performance of the signature based on Boneh *et al.*'s PRF, for a 128-bit security.

Parameters							Size	p_{rej}
$p \approx 2^m$	$\tilde{t}$	δ	B	A	N	τ		
$\approx 2^{229}$	3	88/229	2^{141}	2^{141+12}	256	16	4 916 B	0.012
$\approx 2^{186}$	4	58/186	2^{128}	2^{128+12}	256	16	4 860 B	0.016
$\approx 2^{175}$	5	47/175	2^{128}	2^{128+12}	256	16	5 074 B	0.019

to prevent this attack. [BHH01] and [BVZ12] describe the best known lattice-based attacks with a small number of PRF outputs and require larger δ's than the ones we use. In order to mount them, an adversary has to perform an exhaustive search on the missing bits on several outputs. Let us focus on the attack of [BHH01]. The attacker first chooses $n > 1$ arbitrary ouputs among the $\tilde{t}$ ones. To run the attack, they need to have at least $\frac{2n+1}{3n+1} \cdot m$ bits for each output, thus they can exhaustively search the $n \cdot \left(\frac{2n+1}{3n+1} - \delta\right) \cdot m$ missing bits. For each candidate, the attacker applies the attacks of [BHH01] which consists in reducing a lattice of dimension $O(n)$. To prevent this attack against our parameter sets, we select m, $\tilde{t}$ and δ such that

$$\forall 1 < n \leq \tilde{t},\ n \cdot \left(\frac{2n+1}{3n+1} - \delta\right) \cdot m \geq 128. \tag{6.9}$$

Similarly, we can build an attack based on [BVZ12] with an exhaustive search to get the missing bits. To prevent this attack, we select m, $\tilde{t}$ and δ such that

$$\forall 1 < n \leq \tilde{t},\ n \cdot \left(\frac{2^{n-1}}{2^n - 1} - \delta\right) \cdot m \geq 128. \tag{6.10}$$

Following this discussion, we chose our parameters as follows: by taking $N = 256$ and $\tau = 16$, we first choose $\tilde{t}$, then we take m minimal such that there exists δ which satisfies the constraints (6.8), (6.9) and (6.10) together with the constraint ensuring the uniqueness of the secret (as described previously)

$$\tilde{t} \cdot \delta \cdot m - \tilde{t} \cdot \log\left(\frac{\tilde{t} \cdot \tau}{p_{\text{rej}}}\right) > m.$$

For all parameters provided in Table 6.3 an exhaustive search on (at least) 128 bits has to be performed by the adversary in order to run the attacks from [BHH01; BVZ12].

We should care about another kind of attack based on Coppersmith's method. Indeed, [XSH+19] presented a heuristic attack that breaks Boneh *et al.*'s PRF (for a sufficiently large modulus p) if the number of outputs of the PRF is large enough (depending on δ). However, this polynomial-time attack is not practical and hides *galactic* constant factors. For instance, for $\delta = 2/3$, this attack requires 45 outputs of the PRF and uses a lattice of dimension 209899 in Coppersmith's method. We have checked that for 3 outputs, the attack requires $\delta > 5/6$, for 4 outputs $\delta > 7/10$, and for 5 outputs $\delta > 5/8$. We can observe that our sets of parameters are secure against these values. More generally, for a small number of outputs, the other Coppersmith's style attacks are ineffective if $1-\delta \geq 1/2$. Indeed, [LSSW12]

need δ to be at least 2/3 and [BHH01] proposed a second attack (not described) which needs a large number of outputs to get a δ close to 1/2.

To the best of our knowledge, the quantum security of Boneh *et al.*'s PRF has not been analyzed yet. Our signature protocol is thus a post-quantum candidate and requires further analysis of its security by quantum algorithm specialists.

6.6. Conclusion

In this chapter, we have developed an MPCitH technique to deal with small secret values living modulo a large value. It enables us to propose two new signature schemes:

- the first one relying on the subset sum problem, achieving sizes around 20 KB (for 128-bit security),
- the second one relying on the Boneh-Halevi-Howgrave-Graham PRF, achieving sizes around 5 KB (also for 128-bit security).

The SSP-based scheme could be considered as a conservative post-quantum scheme since the subset sum problem has been extensively studied in the state of the art. Unfortunately the obtained signature sizes are not competitive among the post-quantum schemes. It is the opposite situation for the scheme relying on the [BHH01] PRF. While it is one of the shortest MPCitH-based signature schemes, there exists no literature about the quantum cryptanalysis of this pseudo-random function.

Let us remark that the technique of the sharings over integers is not limited to signature schemes. It is useful for any MPCitH proof of knowledge as soon as we need to deal with small values in large modulus. This manuscript focuses on the context of the post-quantum signatures, but other applications have been analyzed in [FMRV22b]:

- Proving the knowledge of a solution of an *inhomogeneous short integer solution* problem instance. It improves the state of the art when the modulus q is not an NTT-friendly prime.
- Proving the knowledge of a secret key and plaintext(s) matching a (set of) FHE ciphertext(s).

Chapter 7.

Building MPCitH-based Signatures from MQ, MinRank and Rank SD

We have shown that the MPC-in-the-Head paradigm improves the state of the art of signature schemes relying on the syndrome decoding problem and the subset sum problem. A natural follow-up work consists in generalizing this approach to other hard problems. Given a problem, the goal is to design the most efficient MPC protocol which verifies a solution to this problem. In this chapter, we investigate the cases of the multivariate quadratic problem, the MinRank problem and the rank syndrome decoding problem.

The results presented in this chapter were released in November 2022 on the cryptology ePrint archive [Fen22] and will be published in the proceedings of the international conference *ACNS 2024*.

Contents

7.1. Introduction **108**
7.2. Methodology **109**
7.3. Signature Scheme from $\mathcal{MQ}$ **112**
7.4. Signature Scheme from MinRank and Rank SD **116**
7.5. Running times **126**
7.6. Conclusion **128**

7.1. Introduction

The security of the MPCitH-based signature schemes only depends on the security of commitment/hash functions and the security of a one-way function. As explained in Section 3.2, the choice of this one-way function is left to the signature designers. A first research line [ARS+15; DKR+21] has focused on the design of MPC-friendly primitives and their use with the MPC-in-the-Head paradigm to get short signatures. This methodology has the disadvantage of requiring deep cryptanalysis of the introduced primitives. Another strategy would be to use standard symmetric primitives like AES as security assumptions for the MPCitH-based signatures, but it tends to produce larger signatures [DDOS19; BDK+21b; DOT21]. As a last option, we can rely on a hard problem that exists for a long time and thus which is well understood. In the previous chapters, we succeeded in designing efficient signature schemes using the syndrome decoding problem (with the Hamming metric) and the subset sum problem. These two cases have been covered, but a natural question is

Which performance can we get when using
the MPC-in-the-Head paradigm with other hard problems?

Some articles [Wan22; BG22; BESV22] already apply this paradigm to hard problems (multivariate quadratic problem, MinRank problem, ...). One of the drawbacks of almost all the schemes is that, when there is no structure to exploit, they need to rely on *protocols with helper* [Beu20], and thus they suffer from a high computational cost. Recently, [BG22] succeeds in leveraging the structure when considering a structured hard problem (as the ideal rank syndrome decoding problem) and thus achieves smaller sizes by removing the helper from [FJR21].

The present chapter aims to complete the state of the art of the MPC-in-the-Head applied to hard problems. Table 7.1 overviews schemes producing the shortest signatures for some hard problems in 2022 (the year of this work).

Hard Problem	**Best scheme**	**Achieved sizes**
Multivariate Quadratic	Over $\mathbb{F}_4$, [Wan22] Over $\mathbb{F}_{256}$, Section 7.3	8.4 – 9.4 KB 6.9 – 8.3 KB
Min Rank	Section 7.4.2	5.4 – 7.0 KB
Permuted Kernel	[BG22]	8.6 – 9.7 KB
Subset Sum	[FMRV22a]	21.1 – 33.2 KB
Syndrome Decoding *(Hamming)*	[FJR22b]	Over $\mathbb{F}_2$, 10.9 – 15.6 KB Over $\mathbb{F}_{256}$, 8.3 – 11.5 KB
Syndrome Decoding *(Rank)*	Section 7.4.3	5.8 – 7.2 KB

Table 7.1.: State of the art of the MPCiH-based signatures, including this work.

First, we propose a new zero-knowledge proof of knowledge for the *multivariate quadratic* problem. The resulting signature scheme outperforms [Wan22] only when the base field is large enough (*e.g.* $\mathbb{F}_{256}$).

Secondly, we propose two efficient MPC protocols which take as input a matrix $M \in \mathbb{F}_q^{n\times m}$ and which check that the rank of M is upper bounded by r, where r is a public positive integer:

- the first one decomposes M as a product TR where $T \in \mathbb{F}_q^{n\times r}$ and $R \in \mathbb{F}_q^{r\times m}$, and uses an MPC protocol that checks the correctness of a matrix multiplication;
- the second one relies on the fact that the rows of M (represented as elements of $\mathbb{F}_{q^m}$) are roots of a q-polynomial of degree q^r and on the fact that computing a q-polynomial is efficient in MPC while exploiting the linearity of the Frobenius endomorphism $v \mapsto v^q$.

We then use those protocols to build efficient signatures relying on the *MinRank* problem or on the *rank syndrome decoding* problem. Our schemes outperform all the previous proposals, by achieving sizes below 7 KB. They also outperform the [BG22]'s proposals which use structured problems (as the ideal rank syndrome decoding problem) to achieve small sizes.

7.2. Methodology

In each of the following sections, we focus on a specific hard problem which is supposed quantum-resilient:

- Section 7.3: Multivariate Quadratic Problem;
- Section 7.4.2: Min Rank Problem;
- Section 7.4.3: Syndrome Decoding in the *rank* metric;

For each of them, we will use the MPC-in-the-Head paradigm to build a new zero-knowledge protocol. To proceed, we will first describe the MPC protocol we use. This MPC protocol will have the same form as the one from Chapter 5:

- it takes as input an additive sharing of a candidate solution of the studied problem, and eventually an additive sharing of auxiliary data;
- the MPC parties get (only once) a common random value (sampled by the verifier);
- when the tested solution is valid (*i.e.* a solution of the studied hard problem) and when the auxiliary data are genuinely computed, the MPC protocol always outputs Accept; otherwise, it outputs Accept with probability at most p, where p is called the *false positive rate*;
- the views of all the parties except one leak no information about the candidate solution.

By applying the MPC-in-the-Head paradigm to this MPC protocol, we get a 5-round zero-knowledge proof of knowledge of a solution of the studied problem, with soundness error

$$\frac{1}{N} + \left(1 - \frac{1}{N}\right) \cdot p$$

where N is the number of parties involved in the multi-party computation. We do not exhibit the obtained proof of knowledge since the transformation is standard.

To obtain a signature scheme, we apply the Fiat-Shamir transform [FS87] to the previous protocol. Since this protocol has 5 rounds, the security of the resulting scheme should take into account the attack of [KZ20a]. More precisely, the forgery cost of the signature scheme is given by

$$\mathrm{cost}_{\mathrm{forge}} := \min_{\tau_1,\tau_2:\tau_1+\tau_2=\tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau-i}} + N^{\tau_2} \right\}$$

where τ is the number of parallel executions.

Finally, we compare the resulting scheme with all the former schemes which are non-interactive identification schemes based on the same security assumption. To proceed, we first list all these schemes with their formulae of the forgery security and of the communication cost. Since some quantities occur several times, we define some notations to ease the readability. For the forgery cost, we introduce the two following notations:

- $\varepsilon_{\text{helper}}(\tau, M, \varepsilon)$ is the soundness error of a protocol with helper [Beu20] when the helper entity is emulated by a cut-and-choose phase. M is the total number of repetitions in the cut-and-choose phase, ε is the soundness of the unitary protocol relying on the helper, and τ is the number of repetitions of this unitary protocol. We have
$$\varepsilon_{\text{helper}}(\tau, M, \varepsilon) := \max_{M-\tau \leq k \leq M} \left\{ \frac{\binom{k}{M-\tau}}{\binom{M}{M-\tau}} \cdot \varepsilon^{k-(M-\tau)} \right\}.$$
- $\text{KZ}(p_1, p_2)$ is the forgery cost of [KZ20a] for a 5-round protocol[1]. We have
$$\text{KZ}(p_1, p_2) := \min_{\tau_1, \tau_2 : \tau_1 + \tau_2 = \tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} p_1^i (1-p_1)^{\tau - i}} + \frac{1}{p_2^{\tau_2}} \right\}.$$

For the communication cost (*i.e.* the signature size), we introduce the following notations:

- μ_{seed} is the cost of sending a λ-bit seed;
- μ_{dig} is the cost of sending a 2λ-bit commitment/hash digest;
- μ_{helper} is the cost (per repetition) of using the helper technique of [Beu20], this cost satisfies
$$\mu_{\text{helper}} \leq (\mu_{\text{seed}} + \mu_{\text{dig}}) \cdot \log_2 \left(\frac{M}{\tau} \right)$$
where M is the number of repetitions involved in the cut-and-choose phase emulating the helper. It corresponds to the cost of revealing $M - \tau$ leaves among M in a GGM tree, with the cost of sending the authentication paths of τ leaves among M in a Merkle tree.
- μ_{MPCitH} is the fixed cost (per repetition) of using the MPC-in-the-Head paradigm, we have
$$\mu_{\text{MPCitH}} = \mu_{\text{seed}} \cdot \log_2 N + \mu_{\text{dig}}.$$
It corresponds to the cost of revealing all the leaves but one in a seed tree of N leaves (plus a commitment digest).

Then, to get a numerical comparison, we select one or two instances of the studied hard problem and we compare all these schemes for these precise instances.

To proceed, we need to select the parameters of the schemes when relevant. The signature schemes based on the MPC-in-the-Head paradigm have as parameter the number N of parties involved in the multi-party computation. When taking a small N, we get a faster scheme, but when taking a large N, we get shorter signature sizes. To have a fair comparison between the different schemes, we will always take the same N:

[1] in the case where the verifier can not perform some checks after receiving the first response (see [KZ20a] for details).

- when the protocol relies on a helper, we take $N = 8$ to have a fast scheme and $N = 32$ to have short sizes.
- otherwise, we take $N = 32$ to have a fast scheme and $N = 256$ to have short sizes.

7.2.1. Matrix Multiplication Checking Protocol

In our constructions, we need an MPC protocol that checks that three matrices X, Y, Z satisfy $Z = X \cdot Y$. We describe in Figure 7.1 such a protocol Π^{η}_{MM} which has a positive parameter η. This protocol is a matrix variant of the multiplication checking protocol of [BN20] (optimized in [KZ22]).

Inputs: Each party takes a share of the following sharings as inputs: $[\![X]\!]$ where $X \in \mathbb{F}_q^{m\times p}$, $[\![Y]\!]$ where $Y \in \mathbb{F}_q^{p\times n}$, $[\![Z]\!]$ where $\mathbb{F}_q^{m\times n}$, $[\![A]\!]$ where A has been uniformly sampled from $\mathbb{F}_q^{p\times \eta}$, and $[\![C]\!]$ where $C \in \mathbb{F}_q^{m\times \eta}$ satisfies $C = XA$.

MPC Protocol:

1. The parties get a random $\Sigma \in \mathbb{F}_q^{n\times \eta}$.
2. The parties locally set $[\![D]\!] = [\![Y]\!]\Sigma + [\![A]\!]$.
3. The parties broadcast $[\![D]\!]$ to obtain $D \in \mathbb{F}_q^{p\times \eta}$.
4. The parties locally set $[\![V]\!] = [\![X]\!]D - [\![C]\!] - [\![Z]\!]\Sigma$.
5. The parties open $[\![V]\!]$ to obtain $V \in \mathbb{F}_q^{m\times \eta}$.
6. The parties outputs Accept if $V = 0$ and Reject otherwise.

Figure 7.1.: The MPC protocol Π^{η}_{MM} which checks that $Z = X \cdot Y$.

Lemma 7.2.1. *If $Z = X \cdot Y$ and if C are genuinely computed, then Π^{η}_{MM} always outputs* Accept. *If $Z \neq X \cdot Y$, then Π^{η}_{MM} outputs* Accept *with probability at most $\frac{1}{q^{\eta}}$.*

Proof. We have

$$\begin{aligned} V &= XD - C - Z\Sigma \\ &= X(Y\Sigma + A) - C - Z\Sigma \\ &= (XY - Z)\Sigma - (C - XA). \end{aligned}$$

If $Z = XY$ and $C = XA$, V is equal to zero and thus the parties will always output Accept. In contrast, if $Z \neq XY$, then there exists $(i^*, j^*) \in [m] \times [n]$ such that $Z_{i^*,j^*} - (X \cdot Y)_{i^*,j^*} \neq 0$. Given $k \in \{1, \ldots, \eta\}$, $\Sigma_{j^*,k}$ is uniformly sampled in $\mathbb{F}_q$ and then $((Z - X \cdot Y)\Sigma)_{i^*,k}$ is uniformly random in $\mathbb{F}_q$ (because one term of the sum is uniformly random). Thus, the probability that V is zero is at most the probability that $(Z - X \cdot Y)\Sigma$ is equal to $(C - XA)$ on the row i^* whereas the row i^* of $(Z - X \cdot Y)\Sigma$ is uniformly random in $\mathbb{F}_q^{\eta}$, *i.e.* the probability that V is zero (at row i^*) is at most $\frac{1}{q^{\eta}}$. □

7.2.2. MPCitH Optimizations

It is often possible to optimize the communication cost of a scheme relying on the MPC-in-the-Head paradigm. The common optimization tricks are the following:

- Except for the last party, the input share of a party can be derived from a seed using a pseudo-random generator. Thus, when we need to reveal the input share, we just need to reveal a seed. In practice, a prover must reveal the input shares of $N-1$ parties, so it would imply revealing $N-1$ seeds. To save more communication, we can generate the seeds using a GGM tree, decreasing the number of revealed seeds to $\log_2(N)$ (see [KKW18, Sec. 2.3] for details).
- We do not need to reveal shares for shared random values (as A in Figure 7.1) since they can be entirely derived from the seeds of the previous point.
- We do not need to reveal shares for shared publicly-known values (see [KZ22, Sec. 2.4] for details). For example, we do not need to reveal the share of V broadcasted by the hidden party in Figure 7.1. Indeed, this share can be deduced from the shares of the other parties and knowing that V must be equal to zero (otherwise the verification fails).

7.3. Signature Scheme from $\mathcal{MQ}$

We want to build a zero-knowledge proof of knowledge for the *multivariate quadratic problem*:

Definition 7.3.1 (Multivariate Quadratic Problem - Matrix Form)**.** *Let (q, m, n) be positive integers. The multivariate quadratic problem with parameters (q, m, n) is the following problem:*

Let $(A_i)_{i\in[1:m]}$, $(b_i)_{i\in[1:m]}$, x and y be such that:

1. *x is uniformly sampled from $\mathbb{F}_q^n$,*
2. *for all $i \in [1:m]$, A_i is uniformly sampled from $\mathbb{F}_q^{n\times n}$,*
3. *for all $i \in [1:m]$, b_i is uniformly sampled from $\mathbb{F}_q^n$,*
4. *for all $i \in [1:m]$, y_i is defined as $y_i := x^T A_i x + b_i^T x$.*

From $((A_i)_{i\in[1:m]}, (b_i)_{i\in[1:m]}, y)$, find x.

The prover wants to convince the verifier that she knows $x \in \mathbb{F}_q^n$ such that

$$\begin{cases} y_1 &= x^T A_1 x + b_1^T x \\ &\vdots \\ y_m &= x^T A_m x + b_m^T x \end{cases}$$

To proceed, she will rely on the MPC-in-the-Head paradigm: she will first share the secret vector x and then use an MPC protocol which verifies that this vector satisfies the above relations.

MPC Protocol. Instead of checking the m relations separately, we batch them into a linear combination where coefficients $\gamma_1, \ldots, \gamma_m$ are uniformly sampled in the field extension $\mathbb{F}_{q^n}$. The MPC protocol will check that

$$\sum_{i=1}^{m} \gamma_i (y_i - x^T A_i x - b_i^T x) = 0. \tag{7.1}$$

If one of the relations was not satisfied, then Equation (7.1) would be satisfied only with a probability $\frac{1}{q^\eta}$. We can write the equality as

$$\begin{aligned}\sum_{i=1}^{m} \gamma_i (y_i - b_i^T x) &= \sum_{i=1}^{m} \gamma_i (x^T A_i x) \\ &= x^T \left(\sum_{i=1}^{m} \gamma_i A_i \right) x \\ &= \langle x, w \rangle \quad \text{where} \quad w := \left(\sum_{i=1}^{m} \gamma_i A_i \right) x.\end{aligned}$$

By defining $z := \sum_{i=1}^{m} \gamma_i (y_i - b_i^T x)$ and $w := \left(\sum_{i=1}^{m} \gamma_i A_i\right) x$, proving Equation (7.1) is equivalent to proving that

$$z = \langle x, w \rangle.$$

And to prove the above equality, we can rely on the subprotocol Π_{MM} described in Section 7.2.1 (assuming that all the scalars live in $\mathbb{F}_{q^\eta}$). Thus, the MPC protocol proceeds as follows:

1. The parties get random $\gamma_1, \ldots, \gamma_m \in \mathbb{F}_{q^\eta}$.
2. The parties locally set $[\![z]\!] = \sum_{i=1}^{m} \gamma_i (y_i - b_i^T [\![x]\!])$.
3. The parties locally set $[\![w]\!] = \left(\sum_{i=1}^{m} \gamma_i A_i\right) [\![x]\!]$.
4. The parties execute the protocol Π_{MM} to check that $z = \langle w, x \rangle$.

Since this sub-protocol Π_{MM} produces false positive events with a rate of $\frac{1}{q^\eta}$, if x does not satisfy the m $\mathcal{MQ}$ relations, the complete MPC protocol outputs Accept only with a probability of at most

$$\frac{1}{q^\eta} + \left(1 - \frac{1}{q^\eta}\right) \frac{1}{q^\eta} = \frac{2}{q^\eta} - \frac{1}{q^{2\eta}}.$$

The complete MPC protocol is described in Figure 7.2.

Proof of Knowledge. Using the MPC-in-the-Head paradigm, we transform the above MPC protocol into an interactive zero-knowledge proof of knowledge which enables to convince a verifier that a prover knows the solution of a $\mathcal{MQ}$ problem. The soundness error of the resulting protocol is

$$\varepsilon := \frac{1}{N} + \left(1 - \frac{1}{N}\right) \left(\frac{2}{q^\eta} - \frac{1}{q^{2\eta}}\right).$$

By repeating the protocol τ times, we get a soundness error of ε^τ. To obtain a soundness error of λ bits, we can take $\tau = \left\lceil \frac{-\lambda}{\log_2 \varepsilon} \right\rceil$. We can transform the interactive protocol into a non-interactive argument / signature thanks to the Fiat-Shamir transform [FS87]. According to [KZ20a], the security of the resulting scheme is

$$\mathrm{cost}_{\mathrm{forge}} := \min_{\tau_1, \tau_2 : \tau_1 + \tau_2 = \tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau - i}} + N^{\tau_2} \right\}$$

where $p := \frac{2}{q^\eta} - \frac{1}{q^{2\eta}}$.

Public values: The matrices $A_1, \ldots, A_m \in \mathbb{F}_q^{n \times n}$, the vectors $b_1, \ldots, b_m \in \mathbb{F}_q^n$, and the outputs $y_1, \ldots, y_m \in \mathbb{F}_q$.

Inputs: Each party takes a share of the following sharings as inputs: $[\![x]\!]$ where $x \in \mathbb{F}_q^n$, $[\![a]\!]$ where a has been uniformly sampled from $\mathbb{F}_{q^\eta}^n$, and $[\![c]\!]$ where $c \in \mathbb{F}_{q^\eta}$ satisfies $c = -\langle a, x \rangle$.

MPC Protocol:

1. The parties get random $\gamma_1, \ldots, \gamma_m \in \mathbb{F}_{q^\eta}$ and a random $\varepsilon \in \mathbb{F}_{q^\eta}$.
2. The parties locally set $[\![z]\!] = \sum_{i=1}^m \gamma_i (y_i - b_i^T [\![x]\!])$.
3. The parties locally set $[\![w]\!] = \left(\sum_{i=1}^m \gamma_i A_i\right) [\![x]\!]$.
4. The parties locally set $[\![\alpha]\!] = \varepsilon \cdot [\![w]\!] + [\![a]\!]$.
5. The parties open $\alpha \in \mathbb{F}_{q^\eta}^n$.
6. The parties locally set $[\![v]\!] = \varepsilon \cdot [\![z]\!] - \langle \alpha, [\![x]\!] \rangle - [\![c]\!]$.
7. The parties open $v \in \mathbb{F}_{q^\eta}$.
8. The parties outputs Accept if $v = 0$ and Reject otherwise.

Figure 7.2.: An MPC protocol that verifies that the given input corresponds to a solution of an $\mathcal{MQ}$ problem.

The communication cost of the scheme (in bits) is

$$4\lambda + \tau \cdot \left(\underbrace{(n \cdot \log_2(q)}_{x} + \underbrace{n \cdot \eta \cdot \log_2(q)}_{\alpha} + \underbrace{\eta \cdot \log_2(q)}_{c} + \underbrace{\lambda \cdot \log_2 N + 2\lambda}_{\text{MPCitH}} \right)$$

where λ is the security level, η is a scheme parameter and τ is computed such that the soundness error is of λ bits in the interactive case and such that $\text{cost}_{\text{forge}}$ is of λ bits in the non-interactive case.

Performance and comparison. In what follows, we compare our scheme with the state of the art on two $\mathcal{MQ}$ instances:

Instance 1. Multivariate Quadratic equations over a small field:

$$(q, m, n) = (4, 88, 88),$$

Instance 2. Multivariate Quadratic equations over a larger field:

$$(q, m, n) = (256, 40, 40).$$

Both of these instances are believed to correspond to a security of 128 bits [BMSV22].

We provide in Tables 7.2 and 7.3 a complete comparison of our scheme with the state of the art. In the comparison we put MQ-DSS [CHR+16] which corresponds to the non-interactive version of the 5-round identification scheme of [SSH11]. In the sake of completeness, we also put how the 3-round identification scheme of [SSH11] would perform when applying the Fiat-Shamir transform on it.

Over a small field, the Mesquite [Wan22] scheme has the smallest communication cost, even if our scheme produces competitive signature size. Over a larger field, we can produce signature size close to 7 KB, and thus we outperform all the former schemes.

Remark 7.3.2. *In constrast with the former state of the art, the communication cost of our scheme is independent to the number m of $\mathcal{MQ}$ relations.*

Table 7.2.: Sizes of the signatures relying on the $\mathcal{MQ}$ problem (restricting to the schemes using the FS heuristics).

Scheme Name	Security	Signature Size
[SSH11] (3 rounds)	$(3/2)^\tau$	$\mu_{\text{dig}} + \tau\,[2\mu_{\text{var}} + \mu_{\text{out}} + 2\mu_{\text{dig}}]$
MQ-DSS [CHR+16]	$\text{KZ}(\frac{1}{q}, \frac{1}{2})$	$2\mu_{\text{dig}} + \tau\,[2\mu_{\text{var}} + \mu_{\text{out}} + 2\mu_{\text{dig}}]$
MudFish [Beu20]	$\varepsilon_{\text{helper}}(\tau, M, \frac{1}{q'})^{-1}$	$\mu_{\text{dig}} + \tau\,[2\mu_{\text{var}} + \mu_{\text{out}} + 2\mu_{\text{seed}} + \mu_{\text{dig}} \cdot \log_2(q') + \mu_{\text{helper}}]$
Mesquite [Wan22]	$\varepsilon_{\text{helper}}(\tau, M, \frac{1}{N})^{-1}$	$\mu_{\text{dig}} + \tau\,[\mu_{\text{var}} + \mu_{\text{out}} + \mu_{\text{MPCitH}} + \mu_{\text{helper}}]$
Our scheme	$\text{KZ}(\frac{2}{q^\eta} - \frac{1}{q^{2\eta}}, \frac{1}{N})$	$2\mu_{\text{dig}} + \tau\,[(1+\eta) \cdot \mu_{\text{var}} + \eta \cdot \log_2 q + \mu_{\text{MPCitH}}]$

Note: the used notations are $\mu_{\text{var}} := n \log_2 q$, $\mu_{\text{out}} := m \log_2 q$, plus all the notations defined in Section 7.2.

Table 7.3.: Sizes of the signatures relying on the $\mathcal{MQ}$ problem (restricting to the schemes using the FS heuristics). Numerical comparison.

Instance	Protocol Name	Variant	Parameters				Signature Size
			N	M	τ	η	
$q = 4$, $m = 88$, $n = 88$	[SSH11] (3 rounds)	-	-	219	-	-	28 502 B
	MQ-DSS [CHR+16]	-	-	316	-	-	41 444 B
	MudFish [Beu20]	-	4	191	68	-	14 640 B
	Mesquite [Wan22]	Fast	8	187	49	-	9 578 B
		Short	32	389	28	-	**8 609 B**
	Our scheme	Fast	32	-	40	6	10 764 B
		Short	256	-	25	8	9 064 B
$q = 256$, $m = 40$, $n = 40$	[SSH11] (3 rounds)	-	-	219	-	-	40 328 B
	MQ-DSS [CHR+16]	-	-	156	-	-	28 768 B
	MudFish [Beu20]	Fast	8	176	51	-	15 958 B
		Short	16	250	36	-	13 910 B
	Mesquite [Wan22]	Fast	8	187	49	-	11 339 B
		Short	32	389	28	-	9 615 B
	Our scheme	Fast	32	-	36	2	8 488 B
		Short	256	-	25	2	**7 114 B**

7.4. Signature Scheme from MinRank and Rank SD

In this section, we propose arguments of knowledge for the MinRank problem (Section 7.4.2) and the Rank SD problem (Section 7.4.3). But before that, in Section 7.4.1, we propose two efficient MPC protocols which check that a matrix M has a rank of at most r.

In what follows, we denote $\mathrm{wt}_R(M)$ the rank of a matrix M.

7.4.1. Matrix Rank Checking Protocols

We want to build MPC protocols which check that a matrix has a rank of a most r. Such MPC protocols will be used for arguments of knowledge with the MPC-in-the-Head paradigm. We propose two protocols:

- the first one relies on the rank decomposition of matrices. It has the advantage to be quite *simple*, but its false positive rate is *large*.
- the second one relies on linearized polynomials. It has the advantage to have a *very small* false positive rate, but it sometimes requires to manipulate field extensions of *large degrees*.

7.4.1.1. Using Rank Decomposition.

Let us design an MPC protocol which checks that a matrix $M \in \mathbb{F}^{m\times n}$ has a rank of at most r, *i.e.* $\mathrm{wt}_R(M) \leq r$. To proceed, we will rely on the *rank decomposition*:

$$\begin{gathered}\text{a matrix } M \in \mathbb{F}_q^{n\times m} \text{ has a rank of at most } r \\ \text{if and only if there exists } T \in \mathbb{F}_q^{n\times r} \text{ and } R \in \mathbb{F}_q^{r\times m} \text{ such that } M = TR.\end{gathered}$$

In practice, our MPC protocol that we will denote $\Pi^{\eta}_{\text{RC-RD}}$ takes as input such matrices T and R (in addition to M) and simply executes the matrix multiplication checking protocol Π^{η}_{MM} (see Section 7.2.1), for some positive integer η.

Theorem 7.4.1. *If $\mathrm{wt}_R(M) \leq r$ and if T, R are genuinely computed, then $\Pi^{\eta}_{RC\text{-}RD}$ always outputs* Accept. *If $\mathrm{wt}_R(M) > r$, then $\Pi_{RC\text{-}RD}$ outputs* Accept *with probability at most $\frac{1}{q^{\eta}}$. More precisely, if $\mathrm{wt}_R(M) = w + \delta$ with $\delta \geq 1$, then $\Pi^{\eta}_{RC\text{-}RD}$ outputs* Accept *with probability at most $\frac{1}{q^{\delta\cdot\eta}}$.*

Proof. The final broadcast matrix V in Π^{η}_{MM} satisfies

$$V = (TR - M)\Sigma - (C - TA)$$

where matrices A and C have been built before receiving the random Σ. We have

$$\begin{aligned}\mathrm{wt}_R(M - TR) &\geq \mathrm{wt}_R(M) - \mathrm{wt}_R(TR) \\ &\geq (r + \delta) - r = \delta\end{aligned}$$

It means that $TR - M$ has at least δ non-zero coefficients $(i_1, j_1), \ldots, (i_\delta, j_\delta)$ which are over δ different rows and over δ different columns, *i.e.*

$$\forall k_1, k_2 \in [\delta],\ (i_{k_1} \neq i_{k_2}) \wedge (j_{k_1} \neq j_{k_2}).$$

Let us consider $k \in [\delta]$. The j_kth row of Σ is uniformly sampled in $\mathbb{F}_q^\eta$ and thus the i_kth row of $(M - TR)\Sigma$ is uniformly random in $\mathbb{F}_q^\eta$ (because one of the sum term is uniformly random). Thus, the probability that the i_kth row of V is zero is the probability that $(M - TR)\Sigma$ is equal to $(C - TA)$ on the row i_k whereas the row i_k of $(M - TR)\Sigma$ is uniformly random in $\mathbb{F}_q^\eta$, *i.e.* the probability that the i_kth row of V is zero is $\frac{1}{q^\eta}$. By taking a union bound over all k, we get that the probability that V is zero is at most $\frac{1}{q^{\delta \cdot \eta}}$. □

7.4.1.2. Using Linearized Polynomials.

In what follows, we represent a matrix of $\mathbb{F}_q^{m \times n}$ as an element of $(\mathbb{F}_q^m)^n$. We want to design an MPC protocol which checks that a matrix $M = (x_1, \ldots, x_n) \in (\mathbb{F}_{q^m})^n$ has rank at most r. Equivalently, it means that all x_i belongs to an $\mathbb{F}_q$-linear subspace U of $\mathbb{F}_{q^m}$ of dimension r. Let us define the polynomial $L_U(X)$ as

$$L_U(X) := \prod_{u \in U} (X - u) \in \mathbb{F}_{q^m}[X].$$

The degree of L_U is q^r since U has q^r elements. Showing that $\mathrm{wt}(M) \leq r$ can be done by showing that all x_i's are roots of L_U.

According to [LN96, Theorem 3.52], L_U is a q-polynomial over $\mathbb{F}_{q^m}$, meaning that it is of the form

$$L_U(X) = X^{q^r} + \sum_{i=0}^{r-1} \beta_i X^{q^i}.$$

Such polynomials are convenient for multi-party computation since the Frobenius endomorphism $X \mapsto X^q$ is a linear application in field extensions of $\mathbb{F}_q$ and thus it is communication-free to compute $[\![x^q]\!], [\![x^{q^2}]\!], \ldots$ from $[\![x]\!]$.

The core idea of the rank checking protocol is to check that $L_U(x_1) = L_U(x_2) = \ldots = L_U(x_n) = 0$. To proceed, the MPC protocol will batch these checkings by uniformly sampling $\gamma_1, \ldots, \gamma_n \in \mathbb{F}_{q^m}$ and checking that

$$\sum_{j=1}^{n} \gamma_j \cdot L_U(x_j) = 0. \tag{7.2}$$

If one x_i is not a root of the polynomial L_U, then Equation (7.2) is satisfied only with probability $\frac{1}{q^m}$. Let us rewrite the left term of (7.2):

$$\begin{aligned}\sum_{j=1}^{n} \gamma_j \cdot L_U(x_j) &= \sum_{j=1}^{n} \gamma_j \cdot \left(x_j^{q^r} + \sum_{i=0}^{r-1} \beta_i x_j^{q^i} \right) \\ &= \underbrace{\sum_{j=1}^{n} \gamma_j \cdot x_j^{q^r}}_{:=-z} + \sum_{i=0}^{r-1} \beta_i \cdot \underbrace{\sum_{j=1}^{n} \gamma_j x_j^{q^i}}_{:=w_i}.\end{aligned}$$

By defining $z := -\sum_{j=1}^{n} \gamma_j \cdot x_j^{q^r}$ and $w_i := \sum_{j=1}^{n} \gamma_j x_j^{q^i}$ for $i \in \{0, \ldots, r-1\}$, proving Equation (7.2) is equivalent to proving

$$z = \langle \beta, w \rangle.$$

Our MPC protocol that we will denote $\Pi^\eta_{\text{RC-LP}}$ takes as input $[\![x_1]\!], \ldots, [\![x_n]\!]$ and $[\![L_U]\!] := X^{q^r} + \sum_{i=0}^{r-1} [\![\beta_i]\!] X^{q^i}$ proceeds as follows:

1. The parties get random $\gamma_1, \ldots, \gamma_n \in \mathbb{F}_{q^{m\cdot\eta}}$.
2. The parties locally set $[\![z]\!] = -\sum_{j=1}^{n} \gamma_j [\![x_j]\!]^{q^r}$.
3. The parties locally set $[\![w_i]\!] = \sum_{j=1}^{n} \gamma_j [\![x_j]\!]^{q^i}$ for all $i \in \{0, \ldots, r-1\}$.
4. The parties execute the protocol Π_{MM} to check that $z = \langle \beta, w \rangle$ over $\mathbb{F}_{q^{m\cdot\eta}}$.

Theorem 7.4.2. *If $\mathrm{wt}_R(M) \leq r$ and if L_U is genuinely computed, then $\Pi^{\eta}_{RC\text{-}LP}$ always outputs* Accept. *If $\mathrm{wt}_R(M) > r$, then $\Pi^{\eta}_{RC\text{-}LP}$ outputs* Accept *with probability at most $\frac{1}{q^{m\cdot\eta}} + \left(1 - \frac{1}{q^{m\cdot\eta}}\right)\frac{1}{q^{m\cdot\eta}}$.*

Proof. $[\![L_U]\!]$ is a q-polynomial over $\mathbb{F}_{q^m}$ of degree exactly q^r. It means that its number of roots is at most q^r. According to [LN96, Theorem 3.50], the roots form an $\mathbb{F}_q$-linear subspace V of the field extension $\mathbb{F}_{q^s}$ of $\mathbb{F}_{q^m}$. Since $\mathbb{F}_{q^m}$ is also a linear subspace of $\mathbb{F}_{q^s}$, $V \cap \mathbb{F}_{q^m}$ is a linear subspace of $\mathbb{F}_{q^s}$ (and of $\mathbb{F}_{q^m}$). Its dimension is at most r (since it has at most q^r elements). If $\mathrm{wt}_R(M) > r$, there exist i^* such that

$$L_U(x_{i^*}) \neq 0.$$

We then have two options resulting in $\Pi^{\eta}_{\text{RC-LP}}$ outputing Accept:

- Either $\sum_{j=1}^{n} \gamma_j \cdot L_U(x_j) = 0$, which occurs with probability $\frac{1}{q^{m\cdot\eta}}$;
- Or $\sum_{j=1}^{n} \gamma_j \cdot L_U(x_j) \neq 0$, *i.e.* $z \neq \langle \beta, w \rangle$ and Π_{MM} outputs Accept, which occurs with probability $\frac{1}{q^{m\cdot\eta}}$ since Π_{MM} has a false positive rate of $\frac{1}{q^{m\cdot\eta}}$.

□

7.4.2. Signature Scheme from MinRank

We want to build a zero-knowledge proof of knowledge for the *MinRank problem*:

Definition 7.4.3 (MinRank Problem)**.** *Let (q, m, n, k) be positive integers. The MinRank problem with parameters (q, m, n, k) is the following problem:*

Let $M_0, M_1, \ldots, M_k$, E and x such that:

- *x is uniformly sampled from $\mathbb{F}_q^k$,*
- *for all $i \in [k]$, M_i is uniformly sampled from $\mathbb{F}_q^{n\times m}$,*
- *E is uniformly sampled from $\{E \in \mathbb{F}_q^{n\times m} : \mathrm{wt}_R(E) \leq w\}$,*
- *M_0 is defined as $M_0 = E - \sum_{i=1}^{k} x_i M_i$.*

From $(M_0, M_1, \ldots, M_k)$, find x.

The prover wants to convince the verifier that she knows such an x. To proceed, the prover will first share the secret vector x and then use an MPC protocol which verifies that this vector satisfies the above property.

MPC Protocol. We want to build an MPC protocol which takes as input (a sharing of) x and which outputs

$$\begin{cases} \text{Accept} & \text{if } \mathrm{wt}_R(E) \leq r \\ \text{Reject} & \text{otherwise.} \end{cases}$$

where $E := M_0 + \sum_{i=1}^{k} x_i M_i$.

Given $[\![x]\!]$, the parties can locally build $[\![E]\!]$ as $M_0 + \sum_{i=1}^{k} [\![x_i]\!] M_i$. It remains to check that $[\![E]\!]$ corresponds to the sharing of a matrix of rank at most r. It can be done using one of the two rank checking protocols described in Section 7.4.1: $\Pi^{\eta}_{\text{RC-RD}}$ relying on the rank decomposition or $\Pi^{\eta}_{\text{RC-LP}}$ relying on linearized polynomials, for some parameter η.

The complete MPC protocol is described in Figure 7.3 when relying on the rank decomposition and in Figure 7.4 when relying on linearized polynomials. In the second case, the rows of the matrix E are rewritten as elements of $\mathbb{F}_{q^m}$, but when $m \neq n$, it can be more convenient to work on the columns (depending of the values of m and n).

Public values: $M_0, M_1, \ldots, M_k \in \mathbb{F}_q^{n \times m}$.

Inputs: Each party takes a share of the following sharings as inputs: $[\![x]\!]$ where $x \in \mathbb{F}_q^k$, $[\![T]\!]$ where $T \in \mathbb{F}_q^{n \times r}$, $[\![R]\!]$ where $R \in \mathbb{F}_q^{r \times m}$, $[\![a]\!]$ where a has been uniformly sampled from $\mathbb{F}_q^{r \times \eta}$, and $[\![c]\!] \in \mathbb{F}_q^{n \times \eta}$, such that $M_0 + \sum_{i=1}^{k} x_i M_i = TR$ and $c = Ta$.

MPC Protocol:

1. The parties get a random $\Sigma \in \mathbb{F}_q^{m \times \eta}$.
2. The parties locally set $[\![E]\!] = M_0 + \sum_{i=1}^{k} [\![x_i]\!] M_i$.
3. The parties locally set $[\![\alpha]\!] = [\![R]\!]\Sigma + [\![a]\!]$.
4. The parties open $\alpha \in \mathbb{F}_q^{r \times \eta}$.
5. The parties locally set $[\![v]\!] = [\![T]\!]\alpha - [\![c]\!] - [\![E]\!]\Sigma$.
6. The parties open $v \in \mathbb{F}_q^{n \times \eta}$.
7. The parties outputs Accept if $v = 0$ and Reject otherwise.

Figure 7.3.: An MPC protocol based on the *rank decomposition* technique ($\Pi_{\text{RC-RD}}$) which verifies that the given input corresponds to a solution of a MinRank problem.

Proof of Knowledge. Using the MPC-in-the-Head paradigm, we transform the above MPC protocol into an interactive zero-knowledge proof of knowledge which enables to convince a verifier that a prover knows the solution of a MinRank problem. The soundness error of the resulting protocol is

$$\varepsilon := \frac{1}{N} + \left(1 - \frac{1}{N}\right) p_\eta$$

where $p_\eta := \frac{1}{q^\eta}$ when using $\Pi^{\eta}_{\text{RC-RD}}$ and $p_\eta := \frac{2}{q^{m \cdot \eta}} - \frac{1}{q^{2 \cdot m \cdot \eta}}$ when using $\Pi^{\eta}_{\text{RC-LP}}$. By repeating the protocol τ times, we get a soundness error of ε^τ. To obtain a soundness error of λ bits, we can take $\tau = \left\lceil \frac{-\lambda}{\log_2 \varepsilon} \right\rceil$. We can transform the interactive protocol into a non-interactive proof / signature thanks to the Fiat-Shamir transform [FS87]. According to [KZ20a], the security of the resulting scheme is

$$\text{cost}_{\text{forge}} := \min_{\tau_1, \tau_2 : \tau_1 + \tau_2 = \tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} p_\eta^i (1 - p_\eta)^{\tau - i}} + N^{\tau_2} \right\}.$$

When using $\Pi_{\text{RC-RD}}$, the communication cost of the scheme (in bits) is

$$4\lambda + \tau \cdot \left((\underbrace{k}_{x} + \underbrace{r \times m}_{R} + \underbrace{r \times n}_{T} + \underbrace{r \times \eta}_{\alpha} + \underbrace{n \times \eta}_{c}) \cdot \log_2 q + \underbrace{\lambda \cdot \log_2 N + 2\lambda}_{\text{MPCitH}} \right)$$

Public values: $M_0, M_1, \ldots, M_k \in \mathbb{F}_q^{n\times m}$.

Inputs: Each party takes a share of the following sharings as inputs: $[\![x]\!]$ where $x \in \mathbb{F}_q^k$, $[\![L_U]\!] := X^{q^r} + \sum_{i=0}^{r-1} [\![\beta_i]\!] X^{q^i}$ where $L_U(X) := \prod_{u\in U}(X-u) \in \mathbb{F}_{q^m}[X]$, $[\![a]\!]$ where a has been uniformly sampled from $\mathbb{F}_{q^{m\cdot\eta}}^r$, and $[\![c]\!] \in \mathbb{F}_{q^{m\cdot\eta}}$, such that $c = -\langle \beta, a\rangle$.

MPC Protocol:

1. The parties get random $\gamma_1, \ldots, \gamma_n \in \mathbb{F}_{q^{m\cdot\eta}}$.
2. The parties get a random $\varepsilon \in \mathbb{F}_{q^{m\cdot\eta}}$.
3. The parties locally set $[\![E]\!] = M_0 + \sum_{i=1}^k [\![x_i]\!] M_i$.
4. The parties locally write the rows of $[\![E]\!]$ as elements $(e_1, \ldots, e_m)$ of $\mathbb{F}_{q^m}$
5. The parties locally set $[\![z]\!] = -\sum_{j=1}^n \gamma_j [\![e_j]\!]^{q^r}$.
6. The parties locally set $[\![w_i]\!] = \sum_{j=1}^n \gamma_j [\![e_j]\!]^{q^i}$ for all $i \in \{0, \ldots, r-1\}$.
7. The parties locally set $[\![\alpha]\!] = \varepsilon \cdot [\![w]\!] + [\![a]\!]$.
8. The parties open $\alpha \in \mathbb{F}_{q^{m\cdot\eta}}^r$.
9. The parties locally set $[\![v]\!] = \varepsilon \cdot [\![z]\!] - \langle \alpha, [\![\beta]\!]\rangle - [\![c]\!]$.
10. The parties open $v \in \mathbb{F}_{q^{m\cdot\eta}}$.
11. The parties outputs Accept if $v = 0$ and Reject otherwise.

Figure 7.4.: An MPC protocol based on the technique using *linearized polynomials* ($\Pi_{\text{RC-LP}}$) which verifies that the given input corresponds to a solution of a MinRank problem. U is a $\mathbb{F}_q$-linear subspace of $\mathbb{F}_{q^m}$ of dimension r which contains the rows $(e_1, \ldots, e_n)$ of $E := M_0 + \sum_{i=1}^k x_i M_i \in \mathbb{F}_q^{n\times m}$ represented as elements of $\mathbb{F}_{q^m}$.

where λ is the security level, r is a scheme parameter and τ is computed such that the soundness error is of λ bits in the interactive case and such that $\text{cost}_{\text{forge}}$ is of λ bits in the non-interactive case.

And when using $\Pi_{\text{RC-LP}}$, the communication cost of the scheme (in bits) is

$$4\lambda + \tau \cdot \left((\underbrace{k}_{x} + \underbrace{r \times m}_{L_U} + \underbrace{r \times m \times \eta}_{\alpha} + \underbrace{m \times \eta}_{c}) \cdot \log_2 q + \underbrace{\lambda \cdot \log_2 N + 2\lambda}_{\text{MPCitH}} \right).$$

Performance and comparison. In what follows, we compare our scheme with the state of the art on the MinRank instance [BESV22]:

$$(q, m, n, k, r) = (16, 16, 16, 142, 4).$$

We provide in Tables 7.4 and 7.5 a complete comparison of our scheme with the state of the art. To provide a fair comparison, we propose two variants for [Cou01] and [SINY22]: the first one corresponds to the scheme as described in the original article and the second one is an optimized version. This optimized version includes the following tricks:

- Instead of revealing all the commitments during the first round, the prover just sends a hash digest of them. Then, to enable the verifier to recompute this digest, the prover just needs to send the commitment digests that the verifier can not compute herself.

- The random combination used in the schemes (usually denoted β) is derived from a seed. Then, instead of sending the coefficients of β, the prover can just send this seed. Moreover, this seed and the masks involved in the schemes (usually denoted T, S and X) are also derived from a *common* seed.

- Instead of revealing two matrices such that the difference are of rank (at most) r, the prover send one of the matrices and directly the difference (which is cheaper to send), and thus the verifier can deduce the non-sent matrix.

In the comparison we put how [BG22, Section 2] would perform if we apply the same technique for MinRank problem ([BG22] does not consider the MinRank problem in their article).

First, let us remark that [SINY22] presents no advantage compared to [Cou01]. The soundness error of each iteration is 1/2 instead of 2/3, but each iteration is more expensive. The achieved communication cost is thus equivalent to [Cou01]. [BESV22] is a protocol with helper [Beu20]. The components in the proof transcript are the same as for [Cou01] (and [SINY22]), but it succeeds in achieving a bit smaller signature size just by sending a smaller number of seeds and digests. The MPC-in-the-Head paradigm enables to obtain much smaller sizes. Using techniques from [BG22], the resulting size is around 10 KB. In an independent (and parallel) work, [ARV22] proposes a new scheme using techniques which are similar to our protocol with $\Pi_{\text{RC-RD}}$: they are working on another matrix relation[2] but use a less efficient matrix multiplication checking protocol. They succeed in producing signature with sizes below 8 KB. Our scheme with $\Pi_{\text{RC-RD}}$ achieves similar sizes than [ARV22], but our scheme with $\Pi_{\text{RC-LP}}$ outperforms all the previous ones achieving sizes below 6 KB. For the sake of completeness, we put in the comparison tables how [ARV22] would perform if we use Π_{MM} as subroutine.

Table 7.4.: Sizes of the signatures relying on the MinRank problem (restricting to the schemes using the FS heuristics).

Scheme Name	Security	Signature Size
[Cou01]	$(3/2)^\tau$	$3\tau \cdot \mu_{\text{dig}} + \tau \left[\frac{2}{3}\mu_{\text{mat}} + \frac{2}{3}\mu_{\text{combi}} + \frac{2}{3}\mu_{\text{seed}}\right]$
[Cou01], opt.	$(3/2)^\tau$	$\mu_{\text{dig}} + \tau \left[\frac{1}{3}(\mu_{\text{mat}} + \mu_{\text{rank}} + \mu_{\text{combi}} + 2\mu_{\text{seed}}) + \mu_{\text{dig}}\right]$
[SINY22]	2^τ	$6\tau \cdot \mu_{\text{dig}} + \tau \left[\mu_{\text{mat}} + \frac{1}{2}\mu_{\text{combi}} + \frac{10}{4}\mu_{\text{seed}}\right]$
[SINY22], opt.	2^τ	$\mu_{\text{dig}} + \tau \left[\frac{1}{2}(\mu_{\text{mat}} + \mu_{\text{rank}} + \mu_{\text{combi}} + 3\mu_{\text{seed}}) + 2\mu_{\text{dig}}\right]$
[BESV22]	$\varepsilon_{\text{helper}}(\tau, M, \frac{1}{2})^{-1}$	$\mu_{\text{dig}} + \tau \left[\frac{1}{2}(\mu_{\text{mat}} + \mu_{\text{rank}} + \mu_{\text{combi}} + \mu_{\text{seed}}) + \mu_{\text{dig}} + \mu_{\text{helper}}\right]$
[BG22]	$\varepsilon_{\text{helper}}(\tau, M, \frac{1}{N})^{-1}$	$\mu_{\text{dig}} + \tau \left[\mu_{\text{combi}} + \mu_{\text{rank}} + \mu_{\text{MPCitH}} + \mu_{\text{helper}}\right]$
[ARV22]	$\text{KZ}(\frac{1}{q^n}, \frac{1}{N})$	$2\mu_{\text{dig}} + \tau \left[\mu_{\text{combi}} + (n^2 + 2rn - r^2)\log_2 q + \mu_{\text{MPCitH}}\right]$
[ARV22]+Π_{MM}	$\text{KZ}(\frac{1}{q^\eta}, \frac{1}{N})$	$2\mu_{\text{dig}} + \tau \left[\mu_{\text{combi}} + (r(n-r) + \eta(n-2r))\log_2 q + \mu_{\text{MPCitH}}\right]$
Our scheme (RD)	$\text{KZ}(\frac{1}{q^\eta}, \frac{1}{N})$	$2\mu_{\text{dig}} + \tau \left[\mu_{\text{combi}} + \mu_{\text{rank}} + \eta(n+r)\log_2 q + \mu_{\text{MPCitH}}\right]$
Our scheme (LP)	$\text{KZ}(\frac{2}{q^{m\eta}} - \frac{1}{q^{2m\eta}}, \frac{1}{N})$	$2\mu_{\text{dig}} + \tau \left[\mu_{\text{combi}} + rm\log_2 q + \eta(r+1)m\log_2 q + \mu_{\text{MPCitH}}\right]$

Note: the used notations are $\mu_{\text{mat}} := mn\log 2q$, $\mu_{\text{rank}} := r(m+n)\log_2 q$, $\mu_{\text{combi}} := k\log_2 q$, plus all the notations defined in Section 7.2.

[2] They express the $m - r$ last columns *w.r.t.* the r first ones.

Table 7.5.: Comparison of the signatures relying on the MinRank problem (restricting to the schemes using the FS heuristics). Numerical comparison.

Instance	Protocol Name	Variant	Parameters				Signature Size
			N	M	τ	η	
$q = 16$, $m = 16$, $n = 16$, $k = 142$, $r = 4$	[Cou01]	-	-	-	219	-	52 430 B
		Optimized	-	-	219	-	28 575 B
	[SINY22]	-	-	-	128	-	50 640 B
		Optimized	-	-	128	-	28 128 B
	[BESV22]	-	-	256	128	-	26 405 B
	[BG22]	Fast	8	187	49	-	13 644 B
		Short	32	389	28	-	10 937 B
	[ARV22]	Fast	32	-	28	-	10 116 B
		Short	256	-	18	-	7 422 B
	[ARV22]+Π_{MM}	Fast	32	-	33	9	8 155 B
		Short	256	-	19	9	6 277 B
	Our scheme (RD)	Fast	32	-	33	5	9 288 B
		Short	256	-	19	9	7 122 B
	Our scheme (LP)	Fast	32	-	28	1	7 204 B
		Short	256	-	18	1	**5 518 B**

7.4.3. Signature Scheme from Rank SD

We want to build a zero-knowledge proof of knowledge for the *rank syndrome decoding problem*:

Definition 7.4.4 (Rank Syndrome Decoding Problem - Standard Form)**.** *Let $\mathbb{F}_{q^m}$ be the finite field with q^m elements. Let (n, k, r) be positive integers such that $k \leq n$. The rank syndrome decoding problem with parameters (q, m, n, k, r) is the following problem:*

Let H, x and y be such that:

1. *H is uniformly sampled from $\{(H'|I_{n-k}), H' \in \mathbb{F}_{q^m}^{(n-k)\times n}\}$,*
2. *x is uniformly sampled from $\{x \in \mathbb{F}_{q^m}^n : \mathrm{wt}_R(x) \leq r\}$,*
3. *y is built as $y := Hx$.*

From (H, y), find x.

Remark 7.4.5. *The rank $\mathrm{wt}_R(x)$ of an element x of $\mathbb{F}_{q^m}^n$ is the dimension of the $\mathbb{F}_q$-linear subspace spanned by $x_1, \ldots, x_n$. Equivalently, it is the rank of the matrix M for which the rows are $x_1, \ldots, x_n$ represented as vectors of $\mathbb{F}_q^m$.*

The prover wants to convince the verifier that she knows such an x, *i.e.* a vector $x \in \mathbb{F}_{q^m}^n$ such that $y = Hx$ and $\mathrm{wt}_R(x) \leq r$. Previous works propose proofs of knowledge where the constraint on the weight is an equality, but it is sometimes easier to just prove an inequality

(see [FJR22b] for the case of the Hamming weight). To proceed, the prover will first share the secret vector x and then use an MPC protocol which verifies that this vector satisfies the above property.

Remark 7.4.6. *In the above definition, the parity-check matrix is in* standard form*. It does not decrease the hardness of the problem (since the transformation into a standard form is a polynomial transformation), but it enables to simplify the contruction we propose.*

MPC Protocol. We want to build an MPC protocol which takes as input (a sharing of) x and which outputs

$$\begin{cases} \textsc{Accept} & \text{if } y = Hx \text{ and } \mathrm{wt}_R(x) \leq r \\ \textsc{Reject} & \text{otherwise.} \end{cases}$$

Since H is in standard form, having the equality $y = Hx$ is equivalent to define x as

$$\begin{pmatrix} x_A \\ y - H'x_A \end{pmatrix}$$

for some $x_A \in \mathbb{F}_q^k$. Therefore, we will build an MPC protocol which takes as input (a sharing of) x_A and which outputs

$$\begin{cases} \textsc{Accept} & \text{if } \mathrm{wt}_R(x) \leq r \text{ where } x := \begin{pmatrix} x_A \\ y - H'x_A \end{pmatrix} \\ \textsc{Reject} & \text{otherwise.} \end{cases}$$

Given $[\![x_A]\!]$, the parties can locally build $[\![x_B]\!]$ as $[\![x_B]\!] := y - H'[\![x_A]\!]$, and so they can deduce a sharing $[\![x]\!]$ of x (simply by concatenating the shares of $[\![x_A]\!]$ with the shares of $[\![x_B]\!]$). It remains to check that $[\![x]\!]$ corresponds to the sharing of a vector of $\mathbb{F}_{q^m}^n$ of rank at most r. The latter can be done using one of the two rank checking protocols described in Section 7.4.1: $\Pi^{\eta}_{\text{RC-RD}}$ relying on the rank decomposition or $\Pi^{\eta}_{\text{RC-LP}}$ relying on linearized polynomials, for some parameter η.

The complete MPC protocol is described in Figure 7.5 when relying on the rank decomposition and in Figure 7.6 when relying on linearized polynomials.

Proof of Knowledge. Using the MPC-in-the-Head paradigm, we transform the above MPC protocol into an interactive zero-knowledge proof of knowledge which enables to convince a verifier that a prover knows the solution of a rank syndrome decoding problem. The soundness error of the resulting protocol is

$$\varepsilon := \frac{1}{N} + \left(1 - \frac{1}{N}\right) p_\eta$$

where $p_\eta := \frac{1}{q^\eta}$ when using $\Pi^{\eta}_{\text{RC-RD}}$ and $p_\eta := \frac{2}{q^{m\cdot\eta}} - \frac{1}{q^{2\cdot m\cdot\eta}}$ when using $\Pi^{\eta}_{\text{RC-LP}}$. By repeating the protocol τ times, we get a soundness error of ε^τ. To obtain a soundness error of λ bits, we can take $\tau = \left\lceil \frac{-\lambda}{\log_2 \varepsilon} \right\rceil$. We can transform the interactive protocol into a non-interactive proof / signature thanks to the Fiat-Shamir transform [FS87]. According to [KZ20a], the security of the resulting scheme is

$$\mathrm{cost}_{\text{forge}} := \min_{\tau_1,\tau_2:\tau_1+\tau_2=\tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} p_\eta^i (1-p_\eta)^{\tau-i}} + N^{\tau_2} \right\}.$$

Public values: $H = (H'|I_{n-k}) \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $y \in \mathbb{F}_{q^m}^{n-k}$.

Inputs: Each party takes a share of the following sharings as inputs: $[\![x_A]\!]$ where $x_A \in \mathbb{F}_{q^m}^k$, $[\![T]\!]$ where $T \in \mathbb{F}_q^{n\times r}$, $[\![R]\!]$ where $R \in \mathbb{F}_q^{r\times m}$, $[\![a]\!]$ where a has been uniformly sampled from $\mathbb{F}_q^{r\times\eta}$, and $[\![c]\!]$ where $c \in \mathbb{F}_q^{n\times\eta}$, such that $c = Ta$ and $X = TR$ where X is the matrix form of x.

MPC Protocol:

1. The parties get a random $\Sigma \in \mathbb{F}_q^{m\times\eta}$.
2. The parties locally set $[\![x_B]\!] = y - H'[\![x_A]\!]$.
3. The parties locally write $[\![x]\!] := ([\![x_A]\!], [\![x_B]\!])$ as a matrix $[\![X]\!]$.
4. The parties locally set $[\![\alpha]\!] = [\![R]\!]\Sigma + [\![a]\!]$.
5. The parties open $\alpha \in \mathbb{F}_q^{r\times\eta}$.
6. The parties locally set $[\![v]\!] = [\![T]\!]\alpha - [\![c]\!] - [\![X]\!]\Sigma$.
7. The parties open $v \in \mathbb{F}_q^{m\times\eta}$.
8. The parties outputs Accept if $v = 0$ and Reject otherwise.

Figure 7.5.: An MPC protocol based on the *rank decomposition* technique ($\Pi_{\text{RC-RD}}$) which verifies that the given input corresponds to a solution of a rank syndrome decoding problem.

Public values: $H = (H'|I_{n-k}) \in \mathbb{F}_{q^m}^{(n-k)\times n}$ and $y \in \mathbb{F}_{q^m}^{n-k}$.

Inputs: Each party takes a share of the following sharings as inputs: $[\![x_A]\!]$ where $x \in \mathbb{F}_{q^m}^k$, $[\![L_U]\!] := X^{q^r} + \sum_{i=0}^{r-1}[\![\beta_i]\!]X^{q^i}$ where $L_U(X) := \prod_{u\in U}(X-u) \in \mathbb{F}_{q^m}[X]$, $[\![a]\!]$ where a has been uniformly sampled from $\mathbb{F}_{q^{m\cdot\eta}}^r$, and $[\![c]\!] \in \mathbb{F}_{q^{m\cdot\eta}}$, such that $c = -\langle\beta, a\rangle$.

MPC Protocol:

1. The parties get random $\gamma_1, \ldots, \gamma_n \in \mathbb{F}_{q^{m\cdot\eta}}$.
2. The parties get a random $\varepsilon \in \mathbb{F}_{q^{m\cdot\eta}}$.
3. The parties locally set $[\![x_B]\!] = y - H'[\![x_A]\!]$.
4. The parties locally set $[\![z]\!] = -\sum_{j=1}^n \gamma_j [\![x_j]\!]^{q^r}$.
5. The parties locally set $[\![w_i]\!] = \sum_{j=1}^n \gamma_j [\![x_j]\!]^{q^i}$ for all $i \in \{0, \ldots, r-1\}$.
6. The parties locally set $[\![\alpha]\!] = \varepsilon \cdot [\![w]\!] + [\![a]\!]$.
7. The parties open $\alpha \in \mathbb{F}_{q^{m\cdot\eta}}^r$.
8. The parties locally set $[\![v]\!] = \varepsilon \cdot [\![z]\!] - \langle\alpha, [\![\beta]\!]\rangle - [\![c]\!]$.
9. The parties open $v \in \mathbb{F}_{q^{m\cdot\eta}}$.
10. The parties outputs Accept if $v = 0$ and Reject otherwise.

Figure 7.6.: An MPC protocol based on the technique using *linearized polynomials* ($\Pi_{\text{RC-LP}}$) which verifies that the given input corresponds to a solution of a rank syndrome decoding problem. U is a $\mathbb{F}_q$-linear subspace U of $\mathbb{F}_{q^m}$ of dimension r which contains $x_1, \ldots, x_n$.

When using $\Pi_{\text{RC-RD}}$, the communication cost of the scheme (in bits) is

$$4\lambda + \tau \cdot \left((\underbrace{k \cdot m}_{x_A} + \underbrace{r \times m}_{R} + \underbrace{r \times n}_{T} + \underbrace{r \times \eta}_{\alpha} + \underbrace{n \times \eta}_{c}) \cdot \log_2 q + \underbrace{\lambda \cdot \log_2 N + 2\lambda}_{\text{MPCitH}} \right)$$

where λ is the security level, η is a scheme parameter and τ is computed such that the soundness error is of λ bits in the interactive case and such that $\mathrm{cost}_{\mathrm{forge}}$ is of λ bits in the non-interactive case.

And when using $\Pi_{\text{RC-LP}}$, the communication cost of the scheme (in bits) is

$$4\lambda + \tau \cdot \left((\underbrace{k \cdot m}_{x_A} + \underbrace{r \times m}_{L_U} + \underbrace{r \times m \times \eta}_{\alpha} + \underbrace{m \times \eta}_{c}) \cdot \log_2 q + \underbrace{\lambda \cdot \log_2 N + 2\lambda}_{\text{MPCitH}} \right).$$

Performance and comparison. In what follows, we compare our scheme with the state of the art on the Rank Syndrome Decoding instance [BG22]:

$$(q, m, n, k, r) = (2, 32, 30, 14, 9).$$

We provide in Tables 7.6 and 7.7 a complete comparison of our scheme with the state of the art. To get a more complete comparison, we include the schemes [Ste94], [Vér96] and [FJR21] which can be easily adapted for the rank metric (by replacing the permutations by rank isometries). Moreover, we put in Table 7.7 the achieved performance of [BG22] when relying on structured rank syndrome decoding problem (the parameters of the structured problem come from the original article).

The first schemes [Ste94] and [Vér96] can achieve signature sizes of around 30 KB (let us remark that some optimization tricks have been used to achieve these sizes). Then, using the MPC-in-the-Head technique of the "shared permutation", [FJR21] and [BG22] divide this size by half, achieving communication cost around 15 KB (13 – 19 KB). Finally, our new schemes outperform all these schemes by achieving sizes around 6 – 11 KB. The scheme using a q-polynomial even outperforms the [BG22]'s proposals[3] which rely on structured rank syndrome decoding problems.

Table 7.6.: Sizes of the signatures relying on the rank syndrome decoding problem (restricting to the schemes using the FS heuristics).

Scheme Name	Security	Signature Size
[Ste94]	$(3/2)^\tau$	$\mu_{\mathrm{dig}} + \tau \left[\frac{1}{3}(2\mu_{\mathrm{mat}} + \mu_{\mathrm{rank}} + 2\mu_{\mathrm{seed}}) + \mu_{\mathrm{dig}}\right]$
[Vér96]	$(3/2)^\tau$	$\mu_{\mathrm{dig}} + \tau \left[\frac{1}{3}(\mu_{\mathrm{mat}} + \mu_{\mathrm{ptx}} + \mu_{\mathrm{rank}} + 2\mu_{\mathrm{seed}}) + \mu_{\mathrm{dig}}\right]$
[FJR21]	$\varepsilon_{\mathrm{helper}}(\tau, M, \frac{1}{N})^{-1}$	$\mu_{\mathrm{dig}} + \tau\,[\mu_{\mathrm{mat}} + \mu_{\mathrm{ptx}} + \mu_{\mathrm{rank}} + \mu_{\mathrm{MPCitH}} + \mu_{\mathrm{helper}}]$
[BG22]	$\varepsilon_{\mathrm{helper}}(\tau, M, \frac{1}{N})^{-1}$	$\mu_{\mathrm{dig}} + \tau\,[\mu_{\mathrm{mat}} + \mu_{\mathrm{rank}} + \mu_{\mathrm{MPCitH}} + \mu_{\mathrm{helper}}]$
Our scheme (RD)	$\mathrm{KZ}(\frac{1}{q^\eta}, \frac{1}{N})$	$2\mu_{\mathrm{dig}} + \tau\,[\mu_{\mathrm{ptx}} + \mu_{\mathrm{rank}} + \eta(n+r)\log_2 q + \mu_{\mathrm{MPCitH}}]$
Our scheme (LP)	$\mathrm{KZ}(\frac{2}{q^{m\cdot\eta}} - \frac{1}{q^{2\cdot m\cdot\eta}}, \frac{1}{N})$	$2\mu_{\mathrm{dig}} + \tau\,[\mu_{\mathrm{ptx}} + rm\log_2 q + \eta(r+1)m\log_2 q + \mu_{\mathrm{MPCitH}}]$

Note: the used notations are $\mu_{\mathrm{mat}} := mn\log 2q$, $\mu_{\mathrm{rank}} := r(m+n)\log_2 q$, $\mu_{\mathrm{ptx}} := mk\log_2 q$, plus all the notations defined in Section 7.2.

[3] Theses sizes are larger than the ones in [BG22] because they take $N = 1024$, but here to have a fair comparison with the other schemes, we take $N = 256$.

Table 7.7.: Sizes of the signatures relying on the rank syndrome decoding problem (restricting to the schemes using the FS heuristics). Numerical comparison.

Instance	Protocol Name	Variant	Parameters				Signature Size
			N	M	τ	η	
$q = 2$, $m = 31$, $n = 30$, $k = 15$, $r = 9$	Stern [Ste94]	-	-	-	219	-	31 358 B
	Véron [Vér96]	-	-	-	219	-	27 115 B
	[FJR21]	Fast	8	187	49	-	19 328 B
		Short	32	389	28	-	14 181 B
	[BG22]	Fast	8	187	49	-	15 982 B
		Short	32	389	28	-	12 274 B
	Our scheme (RD)	Fast	32	-	33	19	11 000 B
		Short	256	-	21	24	8 543 B
	Our scheme (LP)	Fast	32	-	30	1	7 376 B
		Short	256	-	20	1	**5 899 B**
Ideal RSD	[BG22]	Fast	32	-	37	-	12 607 B
		Short	256	-	26	-	10 126 B
Ideal RSL	[BG22]	Fast	32	-	27	-	9 392 B
		Short	256	-	17	-	6 754 B

7.5. Running times

We implement our schemes thanks to the library described in Section 9.2. Until 2022, the only way to implement an MPCitH-based proof system was by emulating all the parties of the underlying MPC protocol, implying that we would need to emulate N times a party per repetition. The recent work [AGH+23] changes this drastically. The authors suggest generating the input shares of the parties in a correlated way using a hypercube approach. This optimization enables to emulate only $1 + \log_2(N)$ parties per repetition. For example, in Section 7.3, we propose to take $\tau = 25$ and $N = 256$ for the "short" trade-off of our scheme. Without the optimization of [AGH+23], we would need to emulate $\tau \cdot N = 6400$ times a party per signing. With it, we just need to emulate $\tau \cdot (1 + \log_2 N) = 225$ times a party, reducing the computational cost of the MPC emulation by a factor of 28.

We integrated the [AGH+23] optimization in our implementations. The obtained signing times are given in Table 7.8. The pseudo-randomness is generated using AES in counter mode, the used hash function is SHA3, and the MPC challenge is sampled using SHAKE. We benchmarked our scheme on a 3.8 GHz Intel Core i7 CPU with the support of AVX2 and AES instructions. All the reported timings were measured on this CPU while disabling Intel Turbo Boost.

In Table 7.8, we give the computational contribution of two subparts of the signing algorithm:

- the running time for preparing and committing all the input shares of the MPC protocol,
- and the running time for emulating the MPC protocol.

Table 7.8.: Benchmark of our implementations of the proposed signature schemes.

Scheme	Sizes			Signing time		
	$\lvert\mathsf{sk}\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\sigma\rvert$	Share Commit.	MPC Emulation	Total
Variant "Short" – 256 parties ($N = 256$)						
MQ over $\mathbb{F}_{256}$	96 B	56 B	7 114 B	3.72	6.25	10.56
MQ over $\mathbb{F}_{251}$				5.8	2.17	8.56
MinRank (with RD)	208 B	144 B	7 122 B	4.2	3.75	8.39
MinRank (with LP)	176 B		5 518 B	3.61	13.23	17.22
Rank SD (with RD)	208 B	76 B	8 543 B	3.34	2.36	6.12
Rank SD (with LP)	172 B		5 899 B	2.99	3.72	7.09
Variant "Fast" – 32 parties ($N = 32$)						
MQ over $\mathbb{F}_{256}$	96 B	56 B	8 488 B	0.69	6.9	7.83
MQ over $\mathbb{F}_{251}$				0.99	2.15	3.42
MinRank (with RD)	208 B	144 B	9 288 B	0.86	2.68	3.70
MinRank (with LP)	176 B		7 204 B	0.76	13.63	14.54
Rank SD (with RD)	208 B	76 B	11 000 B	0.71	2.30	3.19
Rank SD (with LP)	172 B		7 376 B	0.58	3.71	4.41

Note: All the timings are given in milliseconds.

We optimized the implementation of the first part which mainly relies on symmetric primitives (pseudo-randomness, commitments, ...) since the code is the same for all the schemes. For example, we rely on fourfold calls of Keccak (for SHA3) using AVX instructions. However, the arithmetic components used by the MPC protocols have *not been optimized*, since it would require dedicated work for each scheme (and is out of the scope of this chapter[4]).

In this chapter, we proposed two MPC protocols to check that a matrix has a small rank: one based on rank decomposition (RD), one based on q-polynomials (LP). The second protocol leads to smaller signature sizes, but it tends to be less efficient in running timing since it involves computation in a field extension. From the benchmark, we can observe that both protocols give similar running times when applied to the rank syndrome decoding problem. However, when applied to MinRank, the MPC protocol based on q-polynomials gives a slow scheme. As explained previously, the arithmetics of the implementations have not been optimized. The scheme "MinRank (with LP)" suffers from this lack of optimizations. An open question would be: can an optimized implementation of the field extension erase this difference in running times between both variants (RD and LP)? This question will probably be answered during the NIST standardization process for post-quantum cryptography since both approaches have been submitted in the NIST call (*c.f.* the schemes MIRA and MiRitH).

[4]Such optimized implementations have been built for the schemes submitted to the NIST call, see Section 9.1.

7.6. Conclusion

In this chapter, we studied the application of the MPC-in-the-Head paradigm to the multivariate quadratic problem, the MinRank problem and the rank syndrome decoding problem. The main contribution was to reduce the task of proving the low rank of a matrix to proving that some field elements are roots of a q-polynomial. Such polynomials are MPC-friendly thanks to the linearity of the Frobenius endomorphism. Using this reduction, we can produce signatures relying on the *MinRank problem* and on the *rank syndrome decoding problem* with sizes below 6 KB. We also proposed the first signature scheme relying on the *multivariate quadratic problem* which fits the MPC-in-the-Head paradigm as described in Chapter 3. This scheme outperforms the former ones when working on large fields such as $\mathbb{F}_{256}$.

The ideas introduced in this chapter have been used in several submissions to the NIST call (see Chapter 9 for details).

Chapter 8.

MPC-in-the-Head with Threshold Linear Secret Sharing

While the MPC-in-the-Head paradigm is not restricted to a particular secret sharing scheme, all the efficient instantiations for small circuits proposed before 2022 rely on additive secret sharings.

In this chapter, we show how using a threshold linear secret sharing scheme (threshold LSSS) can be beneficial to the MPC-in-the-Head paradigm. For a general passively-secure MPC protocol model capturing most of the existing MPCitH schemes, we show that our approach improves the soundness error of the underlying proof system from $1/N$ down to $1/\binom{N}{\ell}$, where N is the number of parties and ℓ is the privacy threshold of the sharing scheme.

Applying our approach with a low-threshold LSSS boosts the performance of the proof system by making the MPC emulation cost independent of N for both the prover and the verifier. The gain is particularly significant for the verification time which becomes logarithmic in N (while the prover still has to generate and commit to the N input shares). We further generalize our result to any threshold ramp LSSS and propose an efficient batching technique relying on Shamir's secret sharing.

The results presented in this chapter were released in October 2022 on the cryptology ePrint archive [FR22] in collaboration with Matthieu Rivain and will be published in the proceedings of the international conference *Asiacrypt 2023*.

Contents

8.1. Introduction 130
8.2. Formalizing the MPCitH-Friendly MPC Protocols 133
8.3. MPC-in-the-Head with Threshold LSS 139
8.4. Further Improvements 147
8.5. Applications 152
8.6. Conclusion 163

8.1. Introduction

As explained in Chapter 3, proof systems built from the MPCitH paradigm can be divided into two categories:

- Schemes targeting small circuits (*e.g.* to construct efficient signature schemes), such as [KKW18; BN20; KZ22]. In these schemes, the considered MPC protocol only needs to be secure in the *semi-honest model*, enabling efficient constructions, but the resulting proof is *linear* in the circuit size. Before the work in this chapter, the schemes in this category were all based on additive secret sharing.

- Schemes such as [AHIV17; GSV21] in which the considered MPC protocol is secure in the *malicious model* and the proof is *sublinear* in the circuit size (in $O(\sqrt{|C|})$ with $|C|$ being the circuit size). Due to their sublinearity, these schemes are more efficient for *middle-size circuits* (while the formers remain more efficient for smaller circuits arising *e.g.* in signature schemes).

We note that other quantum-resilient proof systems exist (a.k.a. SNARK, STARK) which do not rely on the MPCitH paradigm and which achieve polylogarithmic proof size (w.r.t. the circuit size), see *e.g.* [BCR+19; BBHR19]. These schemes are hence better suited for *large circuits*.

The results of this chapter belong to the first category of MPCitH-based schemes (*i.e.* targeting small circuits). In 2022, the best MPCitH-based schemes in this scope relied on $(N-1)$-private passively-secure MPC protocols with N parties [KKW18; BN20; DOT21; KZ22], where the parameter N provides different trade-offs between communication (or signature size) and execution time. In these schemes, the proof is composed of elements of size solely depending on the target security level λ (the "incompressible" part) and other elements of size $\mathcal{O}(\lambda^2/\log N)$ bits (the "variable" part). To obtain short proofs or signatures, one shall hence take a large number of parties N. On the other hand, the prover and verifier running times scale linearly with N (because of the MPC emulation) and hence quickly explode while trying to minimize the proof size.

In this chapter, we first describe a general model of multiparty computation protocol (with additive secret sharing) which captures a wide majority of the protocols used in the MPCitH context. (To the best of our knowledge, our model applies to all the MPCitH schemes except those derived from ZKBoo or Ligero.) As stated by Theorem 3.1.4, a zero-knowledge proof resulting from the transformation of a multiparty computation using only broadcast communication has a soundness error of

$$\frac{1}{N} + p \cdot \left(1 - \frac{1}{N}\right),$$

where N is the number of parties and p is the false-positive rate of the MPC protocol (see Section 3.3.2). We then show how to apply an arbitrary threshold linear secret sharing scheme (LSSS) to our general MPC model and how to transform the obtained MPC protocol into a zero-knowledge proof achieving the following soundness error:

$$\frac{1}{\binom{N}{\ell}} + p \cdot \frac{\ell \cdot (N-\ell)}{\ell+1},$$

where ℓ is the threshold of the LSSS (any ℓ shares leak no information while the secret can be reconstructed from any $\ell + 1$ shares). Our theorems cover all the MPC protocols complying with our general model, and for any threshold LSSS (covering additive sharing as a particular case).

Besides improving soundness, using an LSSS with a small threshold implies significant gains in terms of timings. Indeed, the prover and the verifier do not need to emulate all the N parties anymore, but only a small number of them ($\ell + 1$ for the prover and ℓ for the verifier). For instance, when working with Shamir's secret sharing [Sha79] with polynomials of degree $\ell = 1$, the prover only needs to emulate 2 parties (instead of N) and the verifier only needs to emulate 1 party (instead of $N - 1$) while keeping a soundness error about $\frac{1}{N}$ (assuming a small false positive rate p). On the other hand, the proof size is slightly larger than in the standard case (with additive sharing) since one needs to use a Merkle tree for the commitments (and include authentication paths for the opened commitments in the proof transcript). Overall, our approach provides better trade-offs between proof size and performance for MPCitH schemes while drastically reducing the verification time in particular.

We further generalize our approach to threshold ramp LSSS, for which a gap Δ exists between the number of parties ℓ which leak no information and the number of parties $\ell+1+\Delta$ necessary to reconstruct the secret. We particularly analyze algebraic geometric threshold ramp schemes [CC06] but our result is mostly negative: we show that using such schemes does not bring a direct advantage to our framework. We then show that our result on ramp schemes is still useful in the context of batched proofs (*i.e.* proving simultaneously several statements with a single verification process). We propose a batching technique based on Shamir's secret sharing which enables us to efficiently batch proofs in our framework (for a subset of the existing MPCitH schemes).

Finally, we describe some applications of our techniques. We first adapt the SDitH signature scheme (presented in Chapter 5) to our framework with Shamir's secret sharing. We obtain a variant of this scheme that achieves new interesting size-performance trade-offs. For instance, for a signature size of 10 KB, we obtain a signing time of around 3 ms and a verification time lower than 0.5 ms, which is competitive with SPHINCS$^+$ [ABB+22] in terms of size and verification time while achieving much faster signing. We further apply our batching technique to two different contexts: batched proofs for the SDitH scheme and batched proofs for general arithmetic circuits based on the Limbo proof system [DOT21]. In both cases and for the considered parameters, we obtain an amortized proof size lower than 1/10 of the baseline scheme when batching a few dozen statements, while the amortized running times are also significantly improved (in particular for the verifier).

Related works. The MPC-in-the-Head paradigm was introduced in the seminal work [IKOS07]. The authors propose general MPCitH constructions relying on MPC protocols in the *semi-honest model* and in the *malicious model*. In the former case (semi-honest model), they only consider 2-private MPC protocols using an additive sharing as input (they also propose an alternative construction with 1-private protocols). In the latter case (malicious model), they are not restricted to any type of sharing. The exact security of [IKOS07] is analyzed in [GMO16]. As other previous works about the MPCitH paradigm, our work can be seen as a specialization of the IKOS framework. In particular, we restrict the considered MPC model, optimize the communication in this model and provide a refined analysis for the soundness (in the exact security setting) to achieve good practical performance.

Construction	Sharing Scheme	Priv.	Rob.	Soundness	Restriction
[IKOS07, Sec. 3]	Additive	2	0	$1 - \frac{1}{\binom{N}{2}}$	-
[IKOS07, Sec. 4]	Any	t	t	When $N = \Omega(t)$, $2^{-\Omega(t)}$	-
[GMO16]	Any	t	r	$\max\left\{\frac{\binom{r}{t}}{\binom{N}{t}}, \sum_{j=0}^{k} 2^j \frac{\binom{k}{j}\binom{N-2k}{t-j}}{\binom{N}{t}}\right\}$ with $k = \lfloor r/2 \rfloor + 1$	-
[AHIV17]	Any	t	r	$(1 - \frac{r}{N})^t + \delta$	Broadcast
[DOT21]	Additive	$N-1$	0	$\frac{1}{N} + p\left(1 - \frac{1}{N}\right)$	Broadcast
Our work, Sec. 8.3	LSSS	ℓ	0	$\frac{1}{\binom{N}{\ell}} + p\frac{\ell(N-\ell)}{\ell+1}$	Broadcast Linear operations
Our work, Sec. 8.4.1	LSSS with threshold gap $\Delta + 1$	ℓ	0	$\frac{\binom{\ell+\Delta}{\ell}}{\binom{N}{\ell}} + p \cdot \frac{\ell}{\ell+\Delta+1} \cdot \binom{N-\ell}{\Delta+1}$	Broadcast Linear operations

Table 8.1.: Existing general transformations of an MPC protocol into a zero-knowledge proof, with associated MPC model and resulting soundness error. The column "Priv." indicates the privacy threshold of the MPC protocol, while the column "Rob." indicates its robustness threshold (an MPC protocol is said r-robust in the malicious model when it outputs the right values as soon as there are at most r dishonest parties, up a small probability named the robustness error). N denotes the number of parties in the MPC protocol, δ denotes the robustness error, and p denotes the false positive rate as defined in this work.

To the best of our knowledge, besides [IKOS07], the only previous work which considers MPCitH without relying on an additive secret sharing scheme is Ligero [AHIV17]. Ligero is a practical MPCitH-based zero-knowledge proof system for generic circuits which uses Shamir's secret sharing (or Reed-Solomon codes). The authors consider a particular type of MPC protocol in the *malicious model* and analyze the soundness of the resulting proof system. Ligero achieves sublinear communication cost by packing several witness coordinates in one sharing which is made possible by the use of Shamir's secret sharing.

In comparison, this chapter formalizes the MPC model on which many recent MPCitH-based schemes (with additive sharing) rely and shows how using LSSS in this model can be beneficial. We consider a slightly more restricted MPC model than the one of Ligero: we impose that the parties only perform linear operations on the sharings. On the other hand, we only need the MPC protocol to be secure in the *semi-honest model* and not in the malicious model as Ligero. In fact, this difference of settings (semi-honest versus malicious) makes our techniques and Ligero's different in nature. While Ligero makes use of proximity tests to get a robust MPC protocol, we can use lighter protocols in our case (since we do not need robustness). Moreover, for a given number of parties and a given privacy threshold, the soundness error of our work is smaller than the one of [AHIV17]. On the other hand, we consider MPC protocols which only performs linear operations on shares which, in the current state of the art, cannot achieve sublinearity. For this reason, our work targets proofs of knowledge for small circuits (for example, to build efficient post-quantum signature schemes) while Ligero remains better for middle-size circuits (thanks to the sublinearity).

Finally, let us cite [DOT21] which is another article providing a refined analysis for the

transformation of a general MPC model. The scope of the transformation differs from ours, since it covers $(N-1)$-private MPC protocols using broadcast.

In Table 8.1, we sum up all the MPC models considered in the state of the art of the MPC-in-the-Head paradigm with the soundness errors and limitations of the general schemes.

8.2. Formalizing the MPCitH-Friendly MPC Protocols

Several simple MPC protocols have been proposed that yield fairly efficient zero-knowledge proofs and signature schemes in the MPC-in-the-Head paradigm, see for instance [KZ20b; BD20; BDK+21b; FJR22b]. These protocols lie in a specific subclass of MPC protocols in the semi-honest model which we formalize hereafter.

8.2.1. General Model of MPC Protocol

We consider a passively-secure MPC protocol that performs its computation on a base finite field $\mathbb{F}$ so that all the manipulated variables (including the witness w) are tuples of elements from $\mathbb{F}$. In what follows, the sizes of the different tuples involved in the protocol are kept implicit for the sake of simplicity. The parties take as input an additive sharing $[\![w]\!]$ of the witness w (one share per party). Then the parties compute one or several rounds in which they perform three types of actions:

Receiving randomness: the parties receive a random value (or random tuple) ε from a randomness oracle $\mathcal{O}_R$. When calling this oracle, all the parties get the same random value ε. This might not be convenient in a standard multi-party computation setting (since such an oracle would require a trusted third party or a possibly complex coin-tossing protocol), but in the MPCitH context, these random values are provided by the verifier as challenges.

Receiving hint: the parties can receive a sharing $[\![\beta]\!]$ (one share per party) from a hint oracle $\mathcal{O}_H$, which is sometimes called *the dealer* in the MPC literature. The hint β can depend on the witness w and the previous random values sampled from $\mathcal{O}_R$. Formally, for some function ψ, the hint is sampled as $\beta \leftarrow \psi(w, \varepsilon^1, \varepsilon^2, \ldots; r)$ where $\varepsilon^1, \varepsilon^2, \ldots$ are the previous outputs of $\mathcal{O}_R$ and where r is a fresh random tape.

Hints enable to build more efficient MPC protocols. For example, instead of computing a product of two shared values, the parties can get this product using $\mathcal{O}_H$ and simply check that the product is correct, which is cheaper in communication [BN20].

Computing & broadcasting: the parties can locally compute $[\![\alpha]\!] := [\![\varphi(v)]\!]$ from a sharing $[\![v]\!]$ where φ is an $\mathbb{F}$-linear function, then broadcast all the shares $[\![\alpha]\!]_1, \ldots, [\![\alpha]\!]_N$ to publicly reconstruct $\alpha := \varphi(v)$. If φ is in the form $v \mapsto Av + b$, then the parties can compute $[\![\varphi(v)]\!]$ from $[\![v]\!]$ by letting

$$[\![\varphi(v)]\!]_i := A[\![v]\!]_i + [\![b]\!]_i \quad \text{for each party } i$$

where $[\![b]\!]$ is a publicly-known sharing of b.[1] This process is usually denoted $[\![\varphi(v)]\!] = \varphi([\![v]\!])$. The function φ can depend on the previous random values $\{\varepsilon^i\}_i$ from $\mathcal{O}_R$ and on the previous broadcasted values.

[1] Usually, $[\![b]\!]$ is chosen as $(b, 0, \ldots, 0)$ in the case of the additive sharing.

After t rounds of the above actions, the parties finally output Accept if and only if the publicly reconstructed values $\alpha^1, \ldots, \alpha^t$ satisfy the relation

$$g(\alpha^1, \ldots, \alpha^t) = 0$$

for a given function g.

Protocol 10 gives a general description of an MPC protocol in this paradigm, which we shall use as a model in the rest of the chapter. In general, the computing & broadcasting step can be composed of several iterations, which is further depicted in Protocol 11. For the sake of simplicity, we shall consider a single iteration in our presentation (as in Protocol 10) but we stress that the considered techniques and proofs equally apply to the multi-iteration setting (*i.e.* while replacing step (c) of Protocol 10 by Protocol 11).

1. The parties take as input a sharing $[\![w]\!]$.

2. For $j = 1$ to t, the parties:

   a) get a sharing $[\![\beta^j]\!]$ from the hint oracle $\mathcal{O}_H$, such that
   $$\beta^j \leftarrow \psi^j(w, \varepsilon^1, \ldots, \varepsilon^{j-1}; r^j)$$
   for a uniform random tape r^j;

   b) get a common random ε^j from the oracle $\mathcal{O}_R$;

   c) for some $\mathbb{F}$-linear function $\varphi^j_{(\varepsilon^i)_{i\leq j},(\alpha^i)_{i<j}}$, compute
   $$[\![\alpha^j]\!] := \varphi^j_{(\varepsilon^i)_{i\leq j},(\alpha^i)_{i<j}}([\![w]\!], ([\![\beta^i]\!])_{i\leq j}) \ ,$$
   broadcast $[\![\alpha^j]\!]$, and then publicly reconstruct α^j. *Note: This step can be composed of several iterations as described in Protocol 11.*

3. The parties finally accept if $g(\alpha^1, \ldots, \alpha^t) = 0$ and reject otherwise.

Note: In the above description w, β^j, ε^j, α^j are elements from the field $\mathbb{F}$ or tuples with coordinates in $\mathbb{F}$ (whose size is not made explicit to keep the presentation simple).

Protocol 10: General MPC protocol.

Output distribution. In the following, we shall denote $\vec{\varepsilon} := (\varepsilon^1, \ldots, \varepsilon^t)$, $\vec{\beta} := (\beta^1, \ldots, \beta^t)$, $\vec{\alpha} := (\alpha^1, \ldots, \alpha^t)$ and $\vec{r} := (r^1, \ldots, r^t)$. From the above description, we have that the output of the protocol deterministically depends on the broadcasted values $\vec{\alpha}$ (through the function g), which in turn deterministically depend on the input witness w, the sampled random values $\vec{\varepsilon}$, and the hints $\vec{\beta}$ (through the functions φ's). It results that the functionality computed

(c) for $k = 1$ to η_j:

- compute a sharing

$$[\![\alpha^{j,k}]\!] := \varphi^{j,k}_{(\varepsilon^i)_{i\leq j},(\alpha^i)_{i<j},(\alpha^{j,i})_{i<k}}([\![w]\!], ([\![\beta^i]\!])_{i\leq j}) \ ,$$

  for some $\mathbb{F}$-linear function $\varphi^{j,k}_{(\varepsilon^i)_{i\leq j},(\alpha^i)_{i<j},(\alpha^{j,i})_{i<k}}$;

- broadcast their shares $[\![\alpha^{j,k}]\!]$;
- publicly reconstruct $\alpha^{j,k}$;

We denote $\alpha^j := (\alpha^{j,1}, \ldots, \alpha^{j,\eta_j})$.

Protocol 11: General MPC protocol – Iterative computing & broadcasting step for iteration j (with η_j denoting the number of inner iterations).

by the protocol can be expressed as:

$$f(w, \vec{\varepsilon}, \vec{\beta}) = \begin{cases} \textsc{Accept} & \text{if } g(\vec{\alpha}) = 0, \\ \textsc{Reject} & \text{otherwise,} \end{cases} \qquad \text{with} \quad \vec{\alpha} = \Phi(w, \vec{\varepsilon}, \vec{\beta}) \ , \tag{8.1}$$

where Φ is the deterministic function mapping $(w, \vec{\varepsilon}, \vec{\beta})$ to $\vec{\alpha}$ (defined by the coordinate functions $\varphi^1, \ldots, \varphi^t$). We shall restrict our model to MPC protocols for which the function f satisfies the following properties:

- If w is a *good witness*, namely w is such that $(x, w) \in \mathcal{R}$, and if the hints $\vec{\beta}$ are genuinely sampled as $\beta^j \leftarrow \psi^j(w, (\varepsilon^i)_{i<j}; r^j)$ for every j, then the protocol always accepts. More formally:

$$\Pr_{\vec{\varepsilon},\vec{r}}\left[f(w, \vec{\varepsilon}, \vec{\beta}) = \textsc{Accept} \;\middle|\; \begin{array}{c} (x, w) \in \mathcal{R} \\ \forall j, \beta^j \leftarrow \psi^j(w, (\varepsilon^i)_{i<j}; r^j) \end{array} \right] = 1.$$

- If w is a *bad witness*, namely w is such that $(x, w) \notin \mathcal{R}$, then the protocol rejects with probability at least $1 - p$, for some constant probability p. The latter holds even if the hints $\vec{\beta}$ are not genuinely computed. More formally, for any (adversarially chosen) deterministic functions $\chi^1, \ldots, \chi^t$, we have:

$$\Pr_{\vec{\varepsilon},\vec{r}}\left[f(w, \vec{\varepsilon}, \vec{\beta}) = \textsc{Accept} \;\middle|\; \begin{array}{c} (x, w) \notin \mathcal{R} \\ \forall j, \beta^j \leftarrow \chi^j(w, (\varepsilon^i)_{i<j}; r^j) \end{array} \right] \leq p.$$

We are in the setting where the MPC protocol performs a statistical test on the given witness w, as described in Section 3.3.2. We thus say that a *false positive* occurs whenever the MPC protocol outputs Accept on input a bad witness w, and we call p the *false positive rate*.

The general MPC model introduced above captures a wide majority of the protocols used in the MPCitH context. To the best of our knowledge, all the practical instantiations of the MPCitH paradigm comply with this model except the ZKBoo [GMO16] and Ligero [AHIV17] proof systems. In particular, our model captures:

- the KKW18 protocol [KKW18] which computes arbitrary arithmetic circuits (used in Picnic2 and Picnic3),
- the product checking protocols in [BN20] and BN++ [KZ22],
- the product checking protocols in Limbo [DOT21] and Helium [KZ22],
- the MPC protocols in BBQ [DDOS19] and Banquet [BDK+21b],
- the MPC protocols in LegRoast [BD20] and PorcRoast [BD20],
- the MPC protocol in SDitH [FJR22b],
- the MPC protocols in [FMRV22b].

Example. As an illustration, we recall BN20 protocol to show how it fits our model. This MPC protocol takes as inputs three sharings $[\![x]\!]$, $[\![y]\!]$ and $[\![z]\!]$ where $x, y, z \in \mathbb{F}$ and aims to check that $z = x \cdot y$. It proceeds as follows:

- The parties get from $\mathcal{O}_H$ three hint sharings $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ such that $a, b \leftarrow \mathbb{F}$ and $c = a \cdot b$.
- The parties get from $\mathcal{O}_R$ a common random point $\varepsilon \leftarrow \mathbb{F}$.
- The parties locally compute and broadcast

$$[\![\alpha]\!] = \varepsilon \cdot [\![x]\!] + [\![a]\!] \quad \text{and} \quad [\![\beta]\!] = [\![y]\!] + [\![b]\!] \ .$$

- The parties publicly reconstruct α and β from $[\![\alpha]\!]$ and $[\![\beta]\!]$.
- The parties locally compute and broadcast

$$[\![v]\!] = \varepsilon \cdot [\![z]\!] - [\![c]\!] + \alpha \cdot [\![b]\!] + \beta \cdot [\![a]\!] - \alpha \cdot \beta.$$

- The parties publicly reconstruct v from $[\![v]\!]$.
- The parties output Accept if $v = 0$ and Reject otherwise.

The idea of this protocol is to take (as hint) a multiplicative triple (a, b, c) satisfying $c = a \cdot b$ and to "sacrifice" it using the randomness ε to check that $z = x \cdot y$. If $z \neq x \cdot y$ (or if the hint is not well-constructed), the protocol will output Reject with probability $1 - \frac{1}{|\mathbb{F}|}$, thus its false positive rate is $p = \frac{1}{|\mathbb{F}|}$. This protocol (that checks a multiplication triple) fits our model. Indeed, the number of round is $t := 1$ and we have

- ψ^1 is a randomized function that returns a triple (a, b, c) such that (a, b) is random and $c = a \cdot b$;
- ϵ^1 is a random field element;
- φ^1 is split in two subfunctions $\varphi^{1,1}$ and $\varphi^{1,2}$, as described in Protocol 11 when $\eta_1 = 2$:

$$\varphi^{1,1}(x, y, z, a, b, c, \epsilon) := (x + \epsilon \cdot a, y + b)$$
$$\varphi^{1,2}(x, y, z, a, b, c, \epsilon, \alpha, \beta) := \epsilon \cdot z - c + \alpha \cdot b + \beta \cdot a + \alpha \cdot \beta$$

  where $(\alpha, \beta) := \varphi^{1,1}(x, y, z, a, b, c, \epsilon)$.

8.2.2. Application of the MPCitH Principle

Any MPC protocol complying with the above description gives rise to a practical short-communication zero-knowledge protocol in the MPCitH paradigm. The resulting zero-knowledge protocol is described in Protocol 12: after sharing the witness w, the prover emulates the MPC protocol "in her head", commits the parties' inputs, and sends a hash digest of the broadcast communications; finally, the prover reveals the requested parties' inputs as well as the broadcast messages of the unopened party, thus enabling the verifier to emulate the computation of the opened parties and to check the overall consistency.

Soundness. Assuming that the underlying MPC protocol follows the model of Section 8.2.1 with a false positive rate p, the soundness error of Protocol 12 is (see Theorem 3.1.4)

$$\frac{1}{N} + \left(1 - \frac{1}{N}\right) \cdot p.$$

The above formula results from the fact that a malicious prover might successfully cheat with probability $1/N$ by corrupting the computation of one party or with probability p by making the MPC protocol produce a false positive.

Performance. The communication of Protocol 12 includes:

- the input shares $([\![w]\!]_i, [\![\beta^1]\!]_i, \ldots, [\![\beta^t]\!]_i)$ of the opened parties. In practice, a seed $\mathsf{seed}_i \in \{0,1\}^\lambda$ is associated to each party so that for each committed variable v (among the witness w and the hints $\beta^1, \ldots, \beta^t$) the additive sharing $[\![v]\!]$ is built as

$$\begin{array}{l} [\![v]\!]_i \leftarrow \mathrm{PRG}(\mathsf{seed}_i) \text{ for } i \neq N \\ [\![v]\!]_N = v - \sum_{i=1}^{N-1} [\![v]\!]_i. \end{array}$$

  Thus, instead of committing $([\![w]\!]_i, [\![\beta^1]\!]_i)$, the initial commitments simply include the seeds for $i \neq N$, and com_i^j becomes useless for $j \geq 2$ and $i \neq N$. Formally, we have:

$$\mathsf{com}_i^j = \begin{cases} \mathsf{Com}(\mathsf{seed}_i; \rho_i^1) & \text{for } j = 1 \text{ and } i \neq N \\ \mathsf{Com}([\![w]\!]_N, [\![\beta^1]\!]_N; \rho_N^1) & \text{for } j = 1 \text{ and } i = N \\ \emptyset & \text{for } j > 1 \text{ and } i \neq N \\ \mathsf{Com}([\![\beta^j]\!]_N; \rho_N^j) & \text{for } j > 1 \text{ and } i = N \end{cases}$$

  Some coordinates of the β^j might be uniformly distributed over $\mathbb{F}$ (remember that the β^j are tuples of $\mathbb{F}$ elements). We denote β^{unif} the sub-tuple composed of those uniform coordinates. In this context, the last share $[\![\beta^{\mathrm{unif}}]\!]_N$ can be built as $[\![\beta^{\mathrm{unif}}]\!]_N \leftarrow \mathrm{PRG}(\mathsf{seed}_N)$ so that a seed seed_N can be committed in com_N^1 (instead of committing $[\![\beta^{\mathrm{unif}}]\!]_N$). This way the prover can save communication by revealing seed_N instead of $[\![\beta^{\mathrm{unif}}]\!]_N$ whenever the latter is larger;

- the messages $[\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*}$ broadcasted by the unopened party. Let us stress that one can sometimes save communication by sending only some elements of $[\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*}$ and use the relation $g(\alpha^1, \ldots, \alpha^t) = 0$ to recover the missing ones;

- the hash digests $h_1, \ldots, h_{t+1}$ and the unopened commitments $\mathsf{com}_{i^*}^1, \ldots, \mathsf{com}_{i^*}^t$ (as explained above, we have $\mathsf{com}_{i^*}^j = \emptyset$ for $j > 1$ if $i^* \neq N$).

1. The prover shares the witness w into a sharing $[\![w]\!]$.
2. The prover emulates "in her head" the N parties of the MPC protocol.

   For $j = 1$ to t:

   a) the prover computes
   $$\beta^j = \psi^j(w, (\varepsilon^i)_{1 \leq i < j}),$$
   shares it into a sharing $[\![\beta^j]\!]$;

   b) the prover computes the commitments
   $$\mathsf{com}_i^j := \begin{cases} \mathsf{Com}([\![w]\!]_i, [\![\beta^1]\!]_i; \rho_i^1) & \text{if } j = 1 \\ \mathsf{Com}([\![\beta^j]\!]_i; \rho_i^j) & \text{if } j > 1 \end{cases}$$
   for all $i \in \{1, \ldots, N\}$, for some commitment randomness ρ_i^j;

   c) the prover sends
   $$h_j := \begin{cases} \mathrm{Hash}(\mathsf{com}_1^1, \ldots, \mathsf{com}_N^1) & \text{if } j = 1 \\ \mathrm{Hash}(\mathsf{com}_1^j, \ldots, \mathsf{com}_N^j, [\![\alpha^{j-1}]\!]) & \text{if } j > 1 \end{cases}$$
   to the verifier;

   d) the verifier picks at random a challenge ε^j and sends it to the prover;

   e) the prover computes
   $$[\![\alpha^j]\!] := \varphi^j_{(\varepsilon^i)_{i \leq j}, (\alpha^i)_{i<j}}([\![w]\!], ([\![\beta^i]\!])_{i \leq j})$$
   and recomposes α^j.
   Note: This step is computed according to Protocol 11 in case of an iterative computing & broadcasting step.

   The prover further computes $h_{t+1} := \mathrm{Hash}([\![\alpha^t]\!])$ and sends it to the verifier.
3. The verifier picks at random a party index $i^* \in [1 : N]$ and sends it to the prover.
4. The prover opens the commitments of all the parties except party i^* and further reveals the commitments and broadcast messages of the unopened party i^*. Namely, the prover sends $([\![w]\!]_i, ([\![\beta^j]\!]_i, \rho_i^j)_{j \in [1:t]})_{i \neq i^*}$, $\mathsf{com}_{i^*}^1, \ldots, \mathsf{com}_{i^*}^t$, $[\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*}$ to the verifier.
5. The verifier recomputes the commitments com_i^j and the broadcast values $[\![\alpha^j]\!]_i$ for $i \in [1 : N] \setminus \{i^*\}$ and $j \in [1 : t]$ from $([\![w]\!]_i, ([\![\beta^j]\!]_i, \rho_i^j)_{j \in [1:t]})_{i \neq i^*}$ in the same way as the prover.
6. The verifier accepts if and only if:

   a) the views of the opened parties are consistent with each other, with the committed input shares and with the hash digest of the broadcast messages, *i.e.* for $j = 1$ to $t + 1$,
   $$h_j \stackrel{?}{=} \begin{cases} \mathrm{Hash}(\mathsf{com}_1^1, \ldots, \mathsf{com}_N^1) & \text{if } j = 1 \\ \mathrm{Hash}(\mathsf{com}_1^j, \ldots, \mathsf{com}_N^j, [\![\alpha^{j-1}]\!]) & \text{if } j > 1 \\ \mathrm{Hash}([\![\alpha^t]\!]) & \text{if } j = t + 1 \end{cases}$$

   b) the output of the MPC protocol is Accept, *i.e.*
   $$g(\alpha^1, \ldots, \alpha^t) \stackrel{?}{=} 0.$$

Protocol 12: Zero-knowledge protocol - Application of the MPCitH principle to Protocol 10.

Moreover, instead of revealing the $(N-1)$ seeds of the opened parties, one can generate them from a GGM tree as suggested in [KKW18]. One then only needs to reveal $\log_2 N$ λ-bit seeds. We finally obtain a total communication cost for Protocol 12 of

- when $i^* \neq N$,

$$\mathsf{Cost} = \underbrace{(t+1)\cdot 2\lambda}_{h_1,h_2,\ldots,h_{t+1}} + (\underbrace{\mathsf{inputs}}_{[\![w]\!]_N,[\![\beta^1]\!]_N,\ldots,} + \underbrace{\mathsf{comm}}_{[\![\alpha^1]\!]_{i^*},\ldots,[\![\alpha^t]\!]_{i^*}} + \underbrace{\lambda\cdot\log_2 N}_{\mathsf{seed}_i \text{ for } i\neq i^*} + \underbrace{2\lambda}_{\mathsf{com}^1_{i^*}}).$$

- when $i^* = N$,

$$\mathsf{Cost} = \underbrace{(t+1)\cdot 2\lambda}_{h_1,h_2,\ldots,h_{t+1}} + (\underbrace{\mathsf{comm}}_{[\![\alpha^1]\!]_{i^*},\ldots,[\![\alpha^t]\!]_{i^*}} + \underbrace{\lambda\cdot\log_2 N}_{\mathsf{seed}_i \text{ for } i\neq i^*} + \underbrace{t\cdot 2\lambda}_{\mathsf{com}^1_{i^*},\ldots,\mathsf{com}^t_{i^*}}).$$

where inputs denote the bitsize of $(w,\beta^1,\ldots,\beta^t)$ excluding the uniformly distributed elements β^{unif}, and where comm denotes the bitsize of $(\alpha^1,\ldots,\alpha^t)$ excluding the elements which can be recovered from $g(\alpha^1,\ldots,\alpha^t)=0$.

To achieve a soundness error of $2^{-\lambda}$, one must repeat the protocol $\tau = \frac{\lambda}{\log_2 N}$ times. The resulting averaged cost is the following:

$$\mathsf{Cost} = (t+1)\cdot 2\lambda + \tau\cdot\left(\frac{N-1}{N}\cdot\mathsf{inputs} + \mathsf{comm} + \lambda\cdot\log_2 N + \frac{N-1+t}{N}\cdot 2\lambda\right).$$

Several recent works based on the MPCitH paradigm [BD20; KZ22; FJR22b] provides zero-knowledge identification protocols with communication cost below 10 KB for a 128-bit security level. Unfortunately, to obtain a small communication cost, one must take a large number of parties N, which induces an important computational overhead compared to other approaches to build zero-knowledge proofs. Indeed, the prover must emulate N parties in her head for each of the τ repetitions of the protocol, which makes a total of $\frac{\lambda N}{\log_2 N}$ party emulations to achieve a soundness error of $2^{-\lambda}$. Thus, increasing N has a direct impact on the performance. For instance, scaling from $N=16$ to $N=256$ roughly halves the communication but increases the computation by a factor of eight. Given this state of affairs, a natural question is the following:

Can we build zero-knowledge proofs in the MPC-in-the-head paradigm while avoiding the computational overhead of emulating all the parties of the multiparty computation?

In what follows, we show how applying (low-threshold) linear secret sharing to the MPCitH paradigm provides a positive answer to this question.

8.3. MPC-in-the-Head with Threshold LSS

8.3.1. General Principle

Let ℓ and N be integers such that $1 \leq \ell < N$. We consider an $(\ell+1, N)$-threshold linear secret sharing scheme (LSSS), as formally introduced in Definition 2.4.1, which shares a secret $s \in \mathbb{F}$ into N shares $[\![s]\!] \in \mathbb{F}^N$. In particular, the vector spaces of Definition 2.4.1 are

simply defined as $\mathbb{V}_1 = \mathbb{V}_2 = \mathbb{F}$ hereafter (other definitions of these sets will be considered in Section 8.4). We recall that such a scheme implies that the secret can be reconstructed from any $\ell+1$ shares while no information is revealed on the secret from the knowledge of ℓ shares. The following lemmas shall be useful to our purpose. The first lemma holds assuming the MDS conjecture [MS78] while the second one comes from the equivalence between threshold LSSS and interpolation codes [CDN15, Theorem 11.103].

Lemma 8.3.1. *Let $\mathbb{F}$ be a finite field and let ℓ, N be integers such that $1 \leq \ell < N-1$. If an $(\ell+1, N)$-threshold LSSS exists for $\mathbb{F}$, and assuming the MDS conjecture, then $N \leq |\mathbb{F}|$ with the following exception: if $|\mathbb{F}|$ is a power of 2 and $\ell \in \{2, |\mathbb{F}|-2\}$ then $N \leq |\mathbb{F}|+1$.*

Lemma 8.3.2. *Let (Share, Reconstruct) be an $(\ell+1, N)$-threshold LSSS. For every tuple $v_0 \in \mathbb{V}_2^{\ell+1}$ and every subset $J_0 \subseteq [N]$ with $|J_0| = \ell+1$, there exists a unique sharing $[\![s]\!] \in \mathbb{V}_2^N$ such that $[\![s]\!]_{J_0} = v_0$ and such that*

$$\forall J \text{ s.t. } |J| = \ell+1, \mathsf{Reconstruct}_J([\![s]\!]_J) = s \ ,$$

where $s := \mathsf{Reconstruct}_{J_0}(v_0)$. Moreover, there exists an efficient algorithm Expand_{J_0} which returns this unique sharing from $[\![s]\!]_{J_0}$.

Remark 8.3.3. *In the case of the additive sharing scheme, we have $\ell+1 = N$ so that the algorithm Expand is trivial (it simply consists of the identity function). In the case of Shamir's secret sharing scheme (see Definition 2.4.4), the algorithm Expand builds the underlying polynomial and evaluates it into each party' point.*

Application to the MPCitH paradigm. We apply a threshold LSSS to the MPCitH paradigm instead of a simple additive sharing scheme. Let us consider a protocol Π_{add} complying with the MPC model introduced in the previous section (Protocol 10). We can define a protocol Π_{LSSS} similar to Π_{add} with the following differences:

- the parties initially receive an $(\ell+1, N)$-threshold linear secret sharing of the witness w,
- when invoked for a hint β^j, the oracle $\mathcal{O}_H$ returns an $(\ell+1, N)$-threshold linear secret sharing of β^j,
- when the shares of α^j are broadcasted, the value α^j is reconstructed using the algorithm Reconstruct. Namely, the parties choose an arbitrary subset $([\![\alpha^j]\!]_i)_{i \in J_0}$ of size $\ell+1$ from the set of broadcasted shares, run the algorithm $\mathsf{Reconstruct}_{J_0}$ to get α^j, and check that all the broadcast shares are consistent with the output of Expand_{J_0}. If the check fails, the protocol returns Reject.

The resulting MPC protocol, formally described in Protocol 13, is well-defined and ℓ-private in the semi-honest model (meaning that the views of any ℓ parties leak no information about the secret, see Definition 2.4.5). This is formalized in the following theorem (see proof in Appendix A).

Theorem 8.3.4. *Let us consider an MPC protocol Π_{add} complying with the protocol format described in Protocol 10. If Π_{add} is well-defined and $(N-1)$-private, then the protocol Π_{LSSS} corresponding to Π_{add} with an $(\ell+1, N)$-threshold linear secret sharing scheme (see Protocol 13) is well-defined and ℓ-private.*

1. The parties take as input an $(\ell+1, N)$-threshold linear sharing $[\![w]\!]$.
2. For $j = 1$ to t, the parties:
   a) get an $(\ell+1, N)$-threshold linear sharing $[\![\beta^j]\!]$ from the hint oracle $\mathcal{O}_H$, such that
   $$\beta^j \leftarrow \psi^j(w, \varepsilon^1, \ldots, \varepsilon^{j-1}; r^j)$$
   for a uniform random tape r^j;
   b) get a common random ε^j from the oracle $\mathcal{O}_R$;
   c) for some $\mathbb{F}$-linear function $\varphi^j_{(\varepsilon^i)_{i\leq j},(\alpha^i)_{i<j}}$,
      - compute
      $$[\![\alpha^j]\!] := \varphi^j_{(\varepsilon^i)_{i\leq j},(\alpha^i)_{i<j}}([\![w]\!], ([\![\beta^i]\!])_{i\leq j}) \ ,$$
      - broadcast $[\![\alpha^j]\!]$,
      - compute
      $$\alpha^j := \mathsf{Reconstruct}_{J_0}([\![\alpha^j]\!]_{J_0})$$
      for some J_0 of size $\ell+1$,
      - verify that $\mathsf{Expand}_{J_0}([\![\alpha]\!]_{J_0})$ is consistent with $[\![\alpha^j]\!]$ (*i.e.* that $[\![\alpha^j]\!]$ forms a valid sharing) and reject otherwise.

   Note: This step can be composed of several iterations as described in Protocol 11.
3. The parties finally accept if $g(\alpha^1, \ldots, \alpha^t) = 0$ and reject otherwise.

Note: In the above description w, β^j, ε^j, α^j are elements from the field $\mathbb{F}$ or tuples with coordinates in $\mathbb{F}$ (whose size is not made explicit to keep the presentation simple).

Protocol 13: General MPC protocol Π_{LSSS} with LSSS.

8.3.2. Conversion to Zero-Knowledge Proofs

We can convert the MPC protocol using threshold linear secret sharings into a zero-knowledge protocol using the MPC-in-the-Head paradigm. Instead of requesting the views of $N-1$ parties, the verifier only asks for the views of ℓ parties. Since the MPC protocol is ℓ-private, we directly get the zero-knowledge property. One key advantage of using a threshold LSSS is that only $\ell+1$ parties out of N need to be computed by the prover, which we explain further hereafter.

Besides the commitments on the input sharing $[\![w]\!]$, and the hints' sharings $[\![\beta^1]\!], \ldots, [\![\beta^t]\!]$, the prover must send to the verifier the communication between the parties, which for the considered MPC model (see Protocol 13) consists in the broadcast sharings $[\![\alpha^1]\!], \ldots, [\![\alpha^t]\!]$. Observe that such a sharing $[\![\alpha^j]\!]$ is also an LSSS sharing of the underlying value α^j since it

is computed as

$$[\![\alpha^j]\!] := \varphi^j_{(\varepsilon^i,\alpha^i)_{i\leq j}}([\![w]\!], ([\![\beta^i]\!])_{i\leq j})$$

where $[\![w]\!], [\![\beta^1]\!], \ldots, [\![\beta^t]\!]$ are LSSS sharings and φ^j is an affine function. This notably implies that, for all i, the broadcast sharing $[\![\alpha^j]\!] = ([\![\alpha^j]\!]_1, \ldots, [\![\alpha^j]\!]_N)$ contains redundancy. According to Lemma 8.3.2, in order to uniquely define such a sharing, one only needs to commit $\ell + 1$ shares of $[\![\alpha^j]\!]$. In other words, we can choose a fixed subset S of $\ell + 1$ parties and only commit the broadcast shares from these parties, which then acts as a commitment of the full sharing $[\![\alpha^j]\!]$. For all $j \in [1 : t]$, the prover needs to send the broadcast share $[\![\alpha^j]\!]_{i^*}$ of an arbitrary unopened party i^*. To verify the computation of the ℓ opened parties $I = \{i_1, \ldots, i_\ell\} \subseteq [N]$, the verifier can recompute the shares $[\![\alpha^j]\!]_{i_1}, \ldots, [\![\alpha^j]\!]_{i_\ell}$. Then, from these ℓ shares together with $[\![\alpha^j]\!]_{i^*}$, the verifier can reconstruct the shares $[\![\alpha^j]\!]_S$ using $\mathsf{Expand}_{\{i^*,i_1,\ldots,i_\ell\}}$ and check their commitments.

By committing the broadcast messages of only a subset S of parties, the proof becomes independent of the computation of the other parties. It means that the prover must commit the input shares of all the parties but only need to emulate $\ell + 1$ parties to commit their broadcast shares. When ℓ is small with respect to N, this has a great impact on the computational performance of the prover. The resulting zero-knowledge protocol is described in Protocol 14.

8.3.3. Soundness

Consider a malicious prover $\tilde{\mathcal{P}}$ who does not know a correct witness w for the statement x but still tries to convince the verifier that she does. We shall say that such a malicious prover cheats for some party $i \in [1 : N]$ if the broadcast shares $[\![\alpha^1]\!]_i, \ldots, [\![\alpha^t]\!]_i$ recomputed from the committed input/hint shares $[\![w]\!]_i, [\![\beta^1]\!]_i, \ldots, [\![\beta^t]\!]_i$ are not consistent with the committed broadcast shares $([\![\alpha^1]\!]_S, \ldots, [\![\alpha^t]\!]_S)$.

Let us first consider the simple case of false positive rate $p = 0$. If a malicious prover cheats on less than $N - \ell$ parties, then at least $\ell + 1$ parties have broadcast shares which are consistent with $([\![\alpha^1]\!]_i, \ldots, [\![\alpha^t]\!]_i)_{i\in S}$ and give rise to broadcast values $\alpha^1, \ldots, \alpha^t$ for which the protocol accepts, *i.e.* $g(\alpha^1, \ldots, \alpha^t) = 0$. Since $p = 0$, the input shares of those $\ell + 1$ parties necessarily define a good witness w (*i.e.* satisfying $(x, w) \in \mathcal{R}$), which is in contradiction with the definition of a malicious prover. We deduce that in such a zero-false-positive scenario, a malicious prover (who does not know a good witness) has to cheat for at least $N - \ell$ parties. Then, if the malicious prover cheats on more than $N - \ell$ parties, the verifier shall always discover the cheat since she shall necessarily ask for the opening of a cheating party. We deduce that a malicious prover must necessarily cheat on exactly $N - \ell$ parties, and the only way for the verifier to be convinced is to ask for the opening of the exact ℓ parties which have been honestly emulated. The probability of this event to happen is

$$\frac{1}{\binom{N}{N-\ell}} = \frac{1}{\binom{N}{\ell}},$$

which corresponds to the soundness error of the protocol, assuming $p = 0$.

Let us now consider a false positive rate p which is not zero. A malicious prover can then rely on a false positive to get a higher probability to convince the verifier. In case the committed input shares $[\![w]\!]_1, \ldots, [\![w]\!]_N$ were consistent (*i.e.* they formed a valid secret

1. The prover shares the witness w into an $(\ell+1, N)$-threshold linear secret sharing $[\![w]\!]$.

2. The prover emulates "in her head" a (public) subset S of $\ell+1$ parties of the MPC protocol.

   For $j = 1$ to t:

   a) the prover computes
   $$\beta^j = \psi^j(w, (\varepsilon^i)_{i<j}),$$
   shares it into an $(\ell+1, N)$-threshold linear secret sharing $[\![\beta^j]\!]$;

   b) the prover computes the commitments
   $$\mathsf{com}_i^j := \begin{cases} \mathsf{Com}([\![w]\!]_i, [\![\beta^j]\!]_i; \rho_i^j) & \text{if } j = 1 \\ \mathsf{Com}([\![\beta^j]\!]_i; \rho_i^j) & \text{if } j > 1 \end{cases}$$
   for all $i \in [N]$, for some commitment randomness ρ_i^j, and computes the Merkle root
   $$\tilde{h}_j := \mathrm{MerkleTree}(\mathsf{com}_1^j, \ldots, \mathsf{com}_N^j).$$

   c) the prover sends
   $$h_j := \begin{cases} \tilde{h}_j & \text{if } j = 1 \\ \mathrm{Hash}(\tilde{h}_j, [\![\alpha^{j-1}]\!]_S) & \text{if } j > 1 \end{cases}$$
   to the verifier;

   d) the verifier picks at random a challenge ε^j and sends it to the prover;

   e) the prover computes, for $i \in S$,
   $$[\![\alpha^j]\!]_i := \varphi^j_{(\varepsilon^k)_{k\le j}, (\alpha^k)_{k<j}}([\![w]\!]_i, ([\![\beta^k]\!]_i)_{k\le j})$$
   and recomposes α^j. *This step is repeated as many times as in the MPC protocol (cf Protocol 11).*

   The prover further computes $h_{t+1} := \mathrm{Hash}([\![\alpha^t]\!]_S)$ and sends it to the verifier.

3. The verifier picks at random a subset $I \subset [N]$ of ℓ parties (*i.e.* $|I| = \ell$) and sends it to the prover.

4. The prover opens the commitments of all the parties in I, namely she sends $([\![w]\!]_i, ([\![\beta^j]\!]_i, \rho_i^j)_{j\in[t]})_{i\in I}$ to the verifier. The prover further sends the authentication paths $\mathsf{auth}_1, \ldots, \mathsf{auth}_t$ to these commitments, *i.e.* auth_j is the authentication path for $\{\mathsf{com}_i^j\}_{i\in I}$ w.r.t. Merkle root $\tilde{h}_j$ for every $j \in [t]$. Additionally, the prover sends broadcast shares $[\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*}$ of an unopened party $i^* \in S \setminus I$.

5. The verifier recomputes the commitments com_i^j and the broadcast values $[\![\alpha^j]\!]_i$ for $i \in I$ and $j \in [t]$ from $([\![w]\!]_i, ([\![\beta^j]\!]_i, \rho_i^j)_{j\in[t]})_{i\in I}$. Then she recovers $\alpha^1, \ldots, \alpha^t$, by $\alpha^j = \mathsf{Reconstruct}_{I\cup\{i^*\}}([\![\alpha^j]\!]_{I\cup\{i^*\}})$ for every $j \in [t]$.

6. The verifier accepts if and only if:

   a) the views of the opened parties are consistent with each other, with the committed input shares and with the hash digest of the broadcast messages, *i.e.* for $j = 1$ to $t+1$,
   $$h_j \stackrel{?}{=} \begin{cases} \tilde{h}_j & \text{if } j = 1 \\ \mathrm{Hash}(\tilde{h}_j, [\![\alpha^{j-1}]\!]_S) & \text{if } 2 \le j \le t \\ \mathrm{Hash}([\![\alpha^{j-1}]\!]_S) & \text{if } j = t+1 \end{cases}$$
   where $\tilde{h}_j$ is the Merkle root deduced from $(\{\mathsf{com}_i^j\}_{i\in I}, \mathsf{auth}_j)$ and $[\![\alpha^{j-1}]\!]_S$ are the shares in subset S deduced from $[\![\alpha^{j-1}]\!] = \mathsf{Expand}_{I\cup\{i^*\}}([\![\alpha^{j-1}]\!]_{I\cup\{i^*\}})$;

   b) the output of the opened parties are Accept, *i.e.* $g(\alpha^1, \ldots, \alpha^t) \stackrel{?}{=} 0$.

Protocol 14: Zero-knowledge protocol: application of the MPCitH principle to Protocol 13 with an $(\ell+1, N)$-threshold linear secret sharing scheme.

sharing), the soundness error would be

$$\frac{1}{\binom{N}{\ell}} + \left(1 - \frac{1}{\binom{N}{\ell}}\right) \cdot p.$$

However, we cannot enforce a malicious prover to commit a valid secret sharing $[\![w]\!]$ since the verifier never sees more than the shares of ℓ parties. More precisely, let us denote

$$\mathcal{J} := \{J \subset [1:N] : |J| = \ell + 1\}$$

and let $w^{(J)}$ be the witness corresponding to the shares $[\![w]\!]_J$ for some subset $J \in \mathcal{J}$, formally $w^{(J)} := \mathsf{Reconstruct}_J([\![w]\!]_J)$. Then we could have

$$w^{(J_1)} \neq w^{(J_2)}$$

for distinct subsets $J_1, J_2 \in \mathcal{J}$. A malicious prover can exploit this degree of freedom to increase the soundness error.

Soundness attack. Let us take the example of the [BN20] protocol on a field $\mathbb{F}$. In this protocol, the MPC functionality f outputs Accept for a bad witness w (*i.e.* such that $(x, w) \notin \mathcal{R}$) with probability $p = \frac{1}{|\mathbb{F}|}$, *i.e.* if and only if the oracle $\mathcal{O}_R$ samples a specific element ε_w of $\mathbb{F}$. In this context, a possible strategy for the malicious prover is the following:

1. Build the shares $[\![w]\!]_1, \ldots, [\![w]\!]_N$ such that

   $$\forall J_1, J_2 \in \mathcal{J},\ \varepsilon_{w^{(J_1)}} \neq \varepsilon_{w^{(J_2)}}.$$

   We implicitly assume here that $\binom{N}{\ell+1} \leq |\mathbb{F}|$ and that constructing such collision-free input sharing is possible. We assume that $(x, w^{(J)}) \notin \mathcal{R}$ for every J (otherwise the malicious prover can recover a good witness by enumerating the $w^{(J)}$'s).

2. After receiving the initial commitments, the verifier sends the challenge ε.

3. If there exists $J_0 \in \mathcal{J}$ such that $\varepsilon = w^{(J_0)}$, which occurs with probability $\binom{N}{\ell+1} \cdot p$ since all the $\varepsilon^{(J)}$ are distinct, then the malicious prover defines the broadcast values $\alpha^1, \ldots, \alpha^t$ (and the broadcast shares in the set S) according to the broadcast shares of the parties in J_0. It results that the computation of the parties in J_0 is correct and the prover will be able to convince the verifier if the set I of opened parties is a subset of J_0 ($I \subset J_0$).

4. Otherwise, if no subset $J_0 \in \mathcal{J}$ is such that $\varepsilon = w^{(J_0)}$, the malicious prover is left with the option of guessing the set I. Namely, she (randomly) chooses a set I_0 of ℓ parties as well as broadcast values $\alpha^1, \ldots, \alpha^t$ such that $g(\alpha^1, \ldots, \alpha^t) = 0$, and then she deduces and commits the broadcast shares $[\![\alpha^j]\!]_S$ from the $[\![\alpha^j]\!]_{I_0}$ (computed from the committed input shares) and the chosen α^j's. The malicious prover will be able to convince the verifier if and only if the challenge set I matches the guess I_0.

The probability p_{attack} that the malicious prover convinces the verifier using the above strategy satisfies

$$\begin{aligned} p_{\text{attack}} &:= \overbrace{\binom{N}{\ell+1}p}^{\Pr[\exists J_0:\varepsilon=w^{(J_0)}]} \cdot \overbrace{\frac{\binom{\ell+1}{\ell}}{\binom{N}{\ell}}}^{\Pr[I\subset J_0]} + \overbrace{\left(1-\binom{N}{\ell+1}p\right)}^{\Pr[\forall J,\varepsilon\neq w^{(J)}]} \cdot \overbrace{\frac{1}{\binom{N}{\ell}}}^{\Pr[I=I_0]} \\ &= \frac{1}{\binom{N}{\ell}} + p\cdot\frac{\ell\cdot(N-\ell)}{\ell+1} \geq \underbrace{\frac{1}{\binom{N}{\ell}} + \left(1-\frac{1}{\binom{N}{\ell}}\right)\cdot p.}_{\substack{\text{Soundness error if the} \\ \text{committed sharing is well-formed.}}} \end{aligned}$$

Soundness proof. We can prove that the above strategy to forge successful transcripts for the [BN20] protocol is actually optimal and that it further applies to other protocols complying with our model. This is formalized in the following theorem (together with the completeness and HVZK property of the protocol).

Theorem 8.3.5. *Let us consider an MPC protocol Π_{LSSS} complying with the protocol format described in Protocol 13 using an $(\ell+1,N)$-threshold LSSS, such that Π_{LSSS} is ℓ-private in the semi-honest model and of false positive rate p. Then, Protocol 14 built from Π_{LSSS} satisfies the three following properties:*

- ***Completeness.*** *A prover $\mathcal{P}$ who knows a witness w such that $(x,w)\in\mathcal{R}$ and who follows the steps of the protocol always succeeds in convincing the verifier $\mathcal{V}$.*
- ***Soundness.*** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input x, convinces the honest verifier $\mathcal{V}$ to accept with probability*

  $$\tilde{\varepsilon} := \Pr[\langle\tilde{\mathcal{P}},\mathcal{V}\rangle(x)\to 1] \;>\; \varepsilon$$

  where the soundness error ε is defined as

  $$\varepsilon := \frac{1}{\binom{N}{\ell}} + p\cdot\frac{\ell\cdot(N-\ell)}{\ell+1}\ .$$

  Then, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given rewindable black-box access to $\tilde{\mathcal{P}}$, outputs either a witness w satisfying $(x,w)\in\mathcal{R}$, or a commitment/hash collision, by making an average number of calls to $\tilde{\mathcal{P}}$ which is upper bounded by

  $$\frac{4}{\tilde{\varepsilon}-\varepsilon}\cdot\left(1+\tilde{\varepsilon}\cdot\frac{8\cdot(N-\ell)}{\tilde{\varepsilon}-\varepsilon}\right).$$

- ***Honest-Verifier Zero-Knowledge.*** *There exists an efficient simulator $\mathcal{S}$ which, given the random challenge I outputs a transcript which is indistinguishable from a real transcript of Protocol 14.*

Proof. The completeness holds from the completeness property of the underlying MPC protocol. The zero-knowledge property directly comes from the ℓ-privacy property of the MPC protocol with an $(\ell+1,N)$-threshold linear secret sharing scheme. See Appendix B for the soundness proof. □

Remark 8.3.6. *The above theorem includes the MPCitH setting with additive sharing as a particular case. Indeed, when $\ell = N - 1$, we obtain the usual formula for the soundness error, that is:*

$$\ell = N - 1 \quad \Longrightarrow \quad \varepsilon = \frac{1}{N} + p \cdot \left(1 - \frac{1}{N}\right).$$

Remark 8.3.7. *When $\ell = 1$, we have $\varepsilon \approx \frac{1}{N}$ (assuming p is small). It can look as surprising that we can have such soundness error by revealing a single party's view. Since the communication is only broadcast, a verifier does not need to check for inconsistency between several parties, she just needs to check that the revealed views are consistent with the committed broadcast messages. Moreover, the verifier has the guarantee that the shares broadcast by all the parties form* a valid sharing of the open value*. It means that even if the prover reveals only one party's view, the latter can be inconsistent with the committed broadcast. Assuming we use Shamir's secret sharing, committing to a valid broadcast sharing consists in committing a degree-ℓ polynomial such that evaluations are the broadcast shares. By interpolating the broadcast shares of ℓ honest parties (and given the plain value of the broadcast message), one shall entirely fix the corresponding Shamir's polynomial, and the other parties can not be consistent with this polynomial without being consistent with the honest parties (and the latter can only occur if there is a false positive).*

8.3.4. Performance

The advantage of using a threshold LSSS over a standard additive sharing mainly resides in a much faster computation time, for both the prover and the verifier. Indeed, according to the above description, the prover only emulates $\ell + 1$ parties while the verifier only emulates ℓ parties, which is particularly efficient for a small ℓ. For example, assuming that p is negligible and taking $\ell = 1$, the soundness error is $1/N$ (which is similar to standard MPCitH with additive sharing) and the prover only needs to emulate $\ell + 1 = 2$ parties (instead of N) while the verifier only needs to emulate $\ell = 1$ party (instead of $N - 1$).

When targeting a soundness error of λ bits, one needs to repeat the protocol $\tau := \frac{-\lambda}{\log_2 \varepsilon}$ times and thus the number of times that a prover emulates a party is multiplied by τ. Table 8.2 summarizes the number of party emulations for the prover and the verifier for the standard case (additive sharing) and for the case of an $(\ell+1, N)$-threshold LSSS. Interestingly, we observe that the emulation phase is more expensive when increasing N for the additive sharing case while it becomes cheaper for the threshold LSSS case (with some constant ℓ).

The computational bottleneck for the prover when using an LSSS with low threshold ℓ and possibly high N becomes the generation and commitment of all the parties' input shares, which is still linear into N. Moreover the sharing generation for a threshold LSSS might be more expensive than for a simple additive sharing. On the other hand, the verifier does not suffer from this bottleneck since she only has to verify ℓ opened commitments (per repetition). One trade-off to reduce the prover commitment bottleneck is to increase ℓ, which implies a smaller τ (for the same N) and hence decreases the number of commitments.

In terms of communication, using a threshold LSSS implies a slight overhead. In particular, since only ℓ parties out of N are opened, we use a Merkle tree for the commitments and include the authentication paths in the communication.

Let us recall the notations defined in Section 8.2.2:

- inputs: the bitsize of $(w, \beta^1, \ldots, \beta^t)$ excluding the uniformly-distributed elements β^{unif}, and

	With additive sharing	With threshold LSSS	
		$\ell = 1$	Any ℓ
Prover	$\approx \lambda \frac{N}{\log_2 N}$	$\approx \lambda \frac{2}{\log_2 N}$	$\approx \lambda \frac{\ell+1}{\log_2 \binom{N}{\ell}}$
Verifier	$\approx \lambda \frac{N-1}{\log_2 N}$	$\approx \lambda \frac{1}{\log_2 N}$	$\approx \lambda \frac{\ell}{\log_2 \binom{N}{\ell}}$

Table 8.2.: Number of party emulations to achieve a soundness error of $2^{-\lambda}$ (assuming a negligible false positive rate p).

- comm: the bitsize of $([\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*})$ excluding the elements which can be recovered from $g(\alpha^1, \ldots, \alpha^t) = 0$.

We denote unif the bitsize of the uniformly-distributed elements β^{unif}. Then, the proof size (in bits) when repeating the protocol τ times is

$$\mathsf{Cost} = \underbrace{(t+1) \cdot 2\lambda}_{h_1, h_2, \ldots, h_{t+1}} + \tau \cdot (\underbrace{\ell \cdot (\mathsf{inputs} + \mathsf{unif})}_{\{[\![w]\!]_i, [\![\beta^1]\!]_i, \ldots, [\![\beta^t]\!]_i\}_{i \in I}} + \underbrace{\mathsf{comm}}_{[\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*}} + \underbrace{2\lambda \cdot t \cdot \ell \cdot \log_2 \frac{N}{\ell}}_{\mathsf{auth}_1, \ldots, \mathsf{auth}_t}).$$

Let us remark that the bitsize unif appears here while it was not the case for additive sharings. This comes from the fact that, even if β^{unif} is uniformly sampled, $[\![\beta^{\text{unif}}]\!]$ has some structure (*i.e.* some redundancy) when using an arbitrary linear secret sharing scheme.

Remark 8.3.8. *As in the additive case, the prover can generate the input shares from seeds for ℓ parties, and those seeds can be built using a seed tree. However, this tweak will improve (significantly) the communication cost only when the underlying LSSS has a high threshold (as e.g. in the case of additive sharing).*

Let us illustrate the overhead in communication cost when $\ell = 1$ and $t = 1$ and negligible unif (which is often small in practice). In this setting, we obtain an average overhead of $\Delta\mathsf{Cost} \approx \tau \cdot \lambda \cdot (\log_2 N - 2)$. When targeting a λ-bit security, we have $\tau \approx \frac{\lambda}{\log_2 N}$, which gives

$$\Delta\mathsf{Cost} \approx \lambda^2 \cdot \left(1 - \frac{2}{\log_2 N}\right) .$$

We can observe that the communication overhead due to the use of a LSSS is fixed[2] for a given security parameter, and roughly independent on the underlying MPC protocol. When targeting a 128-bit security, this base cost is around 2 KB (for the case $\ell = 1$ and $t = 1$).

8.4. Further Improvements

In this section, we suggest potential ways to generalize our approach.

[2]Let us stress that this fact is true only when the false positive rate p is negliglible compared to $\frac{1}{N}$, to not impact the soundness error of the zero-knowledge protocol.

8.4.1. Using Threshold Ramp Linear Secret Sharing

Theorem 8.3.5 only considers linear secret sharing schemes, but we can generalize the result to any threshold ramp linear secret sharing scheme (see Definition 2.4.2). In such schemes, ℓ shares leak no information about the secret and $\ell + 1 + \Delta$ shares are necessary to reconstruct the secret, with $\Delta > 0$, namely we have a gap between the two thresholds. In our context, this gap shall impact the soundness of the protocol. Indeed, the prover just needs to cheat for $N - \ell - \Delta$ parties (such that there is less than $\ell + \Delta$ honest parties), but the verifier asks to open only ℓ parties. Considering threshold ramp schemes bring more versatility to our approach and opens the door to techniques that are not possible with tight threshold schemes (*e.g.* batching such as proposed below).

Let us remark that the set S of emulated parties in Protocol 14 must be chosen such that $[\![v]\!]_S$ enables to deduce all the shares $[\![v]\!]_{[1:N]}$. In the tight threshold case, such a set S is always of size $\ell + 1$ (see Lemma 8.3.2), but in the case of threshold ramp LSSS, this set S might be larger than $\ell + \Delta + 1$. Moreover, sending shares $[\![\alpha^1]\!]_{i^*}, \ldots, [\![\alpha^t]\!]_{i^*}$ for one non-opened party $i^* \in S$ might not be enough to enable the verifier to recompute $[\![\alpha^j]\!]_S$ for all j. Therefore the size of S and the number of additional shares $[\![\alpha^j]\!]_i$ to be revealed depend on the underlying threshold ramp linear secret sharing, which impacts the communication cost. On the other hand, the soundness error of the obtained proof of knowledge is not impacted.

Theorem 8.4.1. *Let us consider an MPC protocol $\Pi_{QT\text{-}LSSS}$ complying with the protocol format described in Protocol 13, but using an $(\ell, \ell + \Delta + 1, N)$-threshold ramp LSSS in place of an $(\ell + 1, N)$-threshold LSSS, and such that $\Pi_{QT\text{-}LSSS}$ is ℓ-private in the semi-honest model and of false positive rate p. Then, Protocol 14 built from $\Pi_{QT\text{-}LSSS}$ satisfies the three following properties:*

- ***Completeness.*** *A prover $\mathcal{P}$ who knows a witness w such that $(x, w) \in \mathcal{R}$ and who follows the steps of the protocol always succeeds in convincing the verifier $\mathcal{V}$.*

- ***Soundness.*** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input x, convinces the honest verifier $\mathcal{V}$ to accept with probability*

$$\tilde{\varepsilon} := \Pr[\langle \tilde{\mathcal{P}}, \mathcal{V} \rangle(x) \to 1] > \varepsilon$$

  where the soundness error ε is equal to

$$\frac{\binom{\ell+\Delta}{\ell}}{\binom{N}{\ell}} + p \cdot \frac{\ell}{\ell + \Delta + 1} \cdot \binom{N - \ell}{\Delta + 1}.$$

  Then, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given rewindable black-box access to $\tilde{\mathcal{P}}$, produces with either a witness w satisfying $(x, w) \in \mathcal{R}$, or a commitment collision, by making an average number of calls to $\tilde{\mathcal{P}}$ which is upper bounded by

$$\frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{8 \cdot (N - \ell)}{\tilde{\varepsilon} - \varepsilon}\right).$$

- ***Honest-Verifier Zero-Knowledge.*** *There exists an efficient simulator $\mathcal{S}$ which, given random challenge I outputs a transcript which is indistinguishable from a real transcript of Protocol 14.*

Proof. The completeness holds from the completeness property of the underlying MPC protocol. The zero-knowledge property directly comes from the ℓ-privacy property of the MPC protocol with an $(\ell, \ell + \Delta + 1, N)$-threshold linear secret sharing scheme. We refer the reader to [FR22] for the proof of the soundness (this proof is similar to the threshold case in Appendix B). □

8.4.1.1. Using algebraic geometric secret sharing?

One drawback while using a tight threshold LSSS is that the number N of parties is limited by the size of the underlying field $\mathbb{F}$, specifically we have $N \leq |\mathbb{F}|$ (see Lemma 8.3.1). Some sharing schemes on algebraic curves, which are not (tight) threshold but quasi-threshold, have been proposed in [CC06] to handle this issue.

Assuming a negligible false positive rate p, the soundness error is $\binom{\ell+\Delta}{\ell}/\binom{N}{\ell}$ for a threshold ramp scheme instead of $1/\binom{N}{\ell}$ for a tight threshold scheme. Let us focus on the case $\ell = 1$. The soundness error for a threshold ramp scheme then satisfies

$$\frac{\binom{\ell+\Delta}{\ell}}{\binom{N}{\ell}} = \frac{\Delta + 1}{N} \ .$$

In order to gain in soundness (and hence in performance), the above formula should be lower than $1/|\mathbb{F}|$ which is the minimal achievable soundness error for a (tight) threshold scheme (since $N \leq \mathbb{F}$).

In [CC06], the gap Δ is $2g$ where g is the genus of the underlying curve. We must then search for threshold ramp sharing schemes such that

$$\frac{2g+1}{N} \leq \frac{1}{|\mathbb{F}|} \quad \Leftrightarrow \quad N \geq |\mathbb{F}| \cdot (2g+1) \ ,$$

while for such sharing, we have $N \leq |C(\mathbb{F})|$ where $|C(\mathbb{F})|$ is the order of the underlying curve over the field $\mathbb{F}$. However, according to the Hasse-Weil inequality [Was08], we have

$$|C(\mathbb{F})| \leq |\mathbb{F}| + 1 + 2g\sqrt{|\mathbb{F}|}$$

which shows the impossibility of finding an algebraic geometric secret sharing scheme satisfying the above constraint. We deduce that a direct application of algebraic geometric secret sharing schemes [CC06] does not achieve better soundness (and hence better performance) than standard threshold sharing schemes in our context.

We stress that the above argument focuses on the case $\ell = 1$ for simplicity (and since it is relevant to optimize the performance with our approach), but it also holds for any $\ell \geq 1$.

While the above analysis discards the interest in using threshold ramp LSSS based on algebraic geometry to improve the soundness-performance trade-off of our scheme, we let this question open for other types of threshold ramp schemes. However, [CDN15, Theorem 11.121] gives that an $(\ell, \ell + \Delta + 1, N)$-threshold ramp $\mathbb{F}$-linear secret-sharing scheme satisfies

$$\Delta + 1 \geq \frac{N+2}{2|\mathbb{F}| - 1}.$$

Thus, we get a lower bound on the soundness error (on the case $\ell = 1$):

$$\varepsilon := \frac{\Delta + 1}{N} \geq \frac{1}{N} \cdot \frac{N+2}{2|\mathbb{F}| - 1} \geq \frac{1}{2|\mathbb{F}| - 1},$$

implying that such sharing schemes could only have a limited interest to optimize the soundness error.

We show hereafter that the above generalization to threshold ramp LSSS is useful for another purpose, namely an efficient batching technique in our framework.

8.4.2. Batching Proofs with Shamir's Secret Sharing

Principle. Shamir's secret sharing is traditionally used to share a single element of the underlying field, but it can be extended to share several elements simultaneously. To share $v_1, v_2, \ldots, v_u \in \mathbb{F}$, we can sample ℓ random elements $r_1, \ldots, r_\ell$ of $\mathbb{F}$ and build the polynomial P of degree $\ell + u - 1$ such that, given distinct fixed field elements $e_1, \ldots, e_{u+\ell}$,

$$\begin{cases} P(e_1) & = v_1 \\ P(e_2) & = v_2 \\ & \vdots \\ P(e_u) & = v_u \end{cases} \quad \text{and} \quad \begin{cases} P(e_{u+1}) & = r_1 \\ & \vdots \\ P(e_{u+\ell}) & = r_\ell \end{cases}$$

The shares are then defined as evaluations of P on fixed points of $\mathbb{F}\backslash\{e_1, \ldots, e_u\}$. Revealing at most ℓ shares does not leak any information about the shared values $v_1, \ldots, v_u$, while one needs at least $\ell + u$ shares to reconstruct all of them. In other words, this is an $(\ell, \ell + u, N)$-threshold ramp linear secret sharing scheme for the tuple $(v_1, \ldots, v_u)$. Thus, while applying such a sharing to our context, the soundness error is given by (see Theorem 8.4.1)

$$\frac{\binom{\ell+u-1}{\ell}}{\binom{N}{\ell}} + p \cdot \frac{\ell}{\ell + u} \cdot \binom{N - \ell}{u}.$$

When running an MPC protocol on such batch sharing, the operations are simultaneously performed on all the shared secrets $v_1, \ldots, v_u$. It means that we can batch the proof of knowledge of several witnesses which have the same verification circuit (*i.e.* the same functions φ^j in our MPC model – see Protocol 10). Using this strategy, the soundness error is slightly larger, but we can save a lot of communication by using the same sharing for several witnesses.

Specifically, the proof size while batching u witnesses is impacted as follows. The parties' input shares are not more expensive, but to open the communication, the prover now needs to send u field elements by broadcasting (instead of a single one). Thus the communication cost for τ executions is given by

$$\mathsf{Cost} = \underbrace{(t+1) \cdot 2\lambda}_{h_1, h_2, \ldots, h_{t+1}} + \tau \cdot (\underbrace{\ell \cdot (\mathsf{inputs} + \mathsf{rtapes})}_{\{[\![w]\!]_i, [\![\beta^1]\!]_i, \ldots, [\![\beta^t]\!]_i\}_{i \in I}} + \underbrace{u \cdot \mathsf{comm}}_{\alpha^1, \ldots, \alpha^t} + \underbrace{2\lambda \cdot t \cdot \ell \cdot \log_2 \frac{N}{\ell}}_{\mathsf{auth}_1, \ldots, \mathsf{auth}_t}).$$

Unfortunately, the scope of application of this batching technique is limited. In particular, while we can multiply the batched shared secrets by the same scalar, with

$$[\![\begin{pmatrix} \gamma \cdot v_1 \\ \vdots \\ \gamma \cdot v_u \end{pmatrix}]\!] := \gamma \cdot [\![\begin{pmatrix} v_1 \\ \vdots \\ v_u \end{pmatrix}]\!]$$

for some $\gamma \in \mathbb{F}$, we cannot compute

$$[\![\begin{pmatrix} \gamma_1 \cdot v_1 \\ \vdots \\ \gamma_u \cdot v_u \end{pmatrix}]\!] \quad \text{from} \quad [\![\begin{pmatrix} v_1 \\ \vdots \\ v_u \end{pmatrix}]\!]$$

for distinct scalars $\gamma_1, \ldots, \gamma_u$ (whenever at least two scalars are distinct). This restriction implies that the scalar factors used in the verification circuit must be independent of the different witnesses which are batched together. More precisely, it implies that the functions φ^j in our MPC model (Protocol 10) must be of the form

$$\varphi^j_{(\varepsilon^i)_{i\leq j},(\alpha^i)_{i<j}}(\,\cdot\,) \quad = \quad \underbrace{\bar{\varphi}^j_{(\varepsilon^i)_{i\leq j}}(\,\cdot\,)}_{\substack{\text{Linear function with} \\ \varepsilon^i\text{-dependent coefficients}}} \quad + \quad \underbrace{b^j_{(\varepsilon^i)_{i\leq j},(\alpha^i)_{i<j}}}_{\substack{\text{Constant offset which} \\ \text{depends on the } \varepsilon^i\text{'s and } \alpha^i\text{'s}}}$$

This restriction prevents the use of this batching strategy for several MPCitH protocols. For example, all the protocols using the multiplication checking protocol from [BN20] as a subroutine cannot use this batching strategy. To the best of our knowledge, the only protocols in the current state of the art which support this batching strategy are Banquet [BDK+21b] and Limbo [DOT21].

Batching strategies. In what follows, we propose three strategies to batch MPCitH proofs relying on the same verification circuit:

Naive strategy: The naive way to batch u MPCitH proofs is to emulate u independent instances of MPC protocol, one for each input witness. Compared to sending u independent proofs, one can save communication by using the same seed trees and the same commitments for the u instances. This strategy can be applied for standard MPCitH schemes based on additive sharing as well as for our framework of threshold LSSS-based MPCitH. When using additive sharings, the main drawback of this strategy is that the prover and the verifier need to emulate the party computation a large number of times, *i.e.* N times (or $N-1$ times for the verifier) per iteration and per statement. When batching $u \geq 25$ statements with $N = 256$, the prover and the verifier must emulate more than 100 000 parties to achieve a security of 128 bits. When using a low-threshold LSSS, the emulation cost is much cheaper, but the proof transcript is larger. While batching u statements, the emulation cost and the soundness error are given by the following table:

	# Emulations	Soundness Error
Prover	$\tau \cdot (\ell+1) \cdot u$	$\frac{1}{\binom{N}{\ell}} + p \cdot \frac{(N-\ell)\cdot\ell}{\ell+1}$
Verifier	$\tau \cdot \ell \cdot u$	

SSS-based strategy: We can use the batching strategy based on Shamir's secret sharing (SSS) described above. Instead of having u independent input sharings (one per witness), we have a single input sharing batching the u witnesses. The number of MPC emulations is lower than for the naive strategy. The proof size is also smaller and (mostly) below that of the standard setting for small u, but it grows exponentially when considering a small field $\mathbb{F}$. Each batched statement consumes one evaluation point (in $\mathbb{F}$), the

number N of parties is hence limited by $N \leq |\mathbb{F}| + 1 - u$. Because of this limitation together with the security loss due to the use of a ramp sharing scheme, the soundness error of this batched protocol degrades rapidly as u grows. While batching u statements using Shamir's secret sharings, the emulation cost and the soundness error are given by the following table:

	# Emulations	Soundness Error
Prover	$\tau \cdot (\ell + u)$	$\frac{\binom{\ell+u-1}{\ell}}{\binom{N}{\ell}} + p \cdot \frac{\ell}{\ell+u} \cdot \binom{N-\ell}{u}$
Verifier	$\tau \cdot \ell$	

Hybrid strategy: In the previous strategy, the proof size is convex w.r.t. the number u of batched proofs and, for small some u, the curve slope is flatter than the slope in the additive case (see Figure 8.1 from Section 8.5 for illustration). It means that using a hybrid approach can achieve smaller proof sizes (as well as better performance) than with the two above strategies. Specifically, instead of having one input sharing encoding the u witnesses (one per batched statement) and a single emulation of the MPC protocol, we can use ν input sharings each of them encoding $\frac{u}{\nu}$ witnesses and have then ν emulations of the MPC protocol. Using this hybrid strategy, the emulation cost and the soundness error are given by the following table:

	# Emulations	Soundness Error
Prover	$\tau \cdot (\ell + \frac{u}{\nu}) \cdot \nu$	$\frac{\binom{\ell+u/\nu-1}{\ell}}{\binom{N}{\ell}} + p \cdot \frac{\ell}{\ell+u/\nu} \cdot \binom{N-\ell}{u/\nu}$
Verifier	$\tau \cdot \ell \cdot \nu$	

Section 8.5.2 presents some application results for these batching strategies. In particular Figure 8.1 compares the three strategies for batched proofs of the SDitH scheme [FJR22b].

Remark 8.4.2. *In our analysis, we use the number of emulated parties as an indicator of the computational performance. As explained in Section 8.3, we would also need to take into account the computation cost for computing and committing the input sharings (and hints' sharings) which is not negligible, but this cost is hard to estimate without a concrete implementation. We yet remind that the latter cost only impacts the prover and not the verifier. The verification time is soundly predicted by the number of party emulations.*

8.5. Applications

In the past few years, many proof systems relying on the MPC-in-the-Head paradigm have been published. Table 8.3 provides a tentatively exhaustive list of these schemes while indicating for each scheme:

- the base field (or ring) of the function computed by the underlying MPC protocol,
- whether the underlying MPC protocol fits our general model (see Section 8.2.1),
- the hard problem (or one-way function) for which the witness knowledge is proved.

In column *Base Ring*, the notation "$\mathbb{F}$ ($\mathbb{K}$)" means that the function computed by the underlying MPC protocol is composed of $\mathbb{F}$-linear functions and multiplications over $\mathbb{K}$. For example, the schemes for AES use $\mathbb{F}_2$-linear functions and $\mathbb{F}_{256}$-multiplications.

Applying our framework with an arbitrary (low-)threshold linear secret sharing scheme instead of an additive sharing scheme is possible whenever

- the underlying MPC protocol fits the model introduced in Section 8.2.1,
- the underlying MPC protocol is defined over a field (and not only a ring),
- this base field is large enough (since the number of parties N is limited by the size of the field).

Because of this last condition, all the proof systems for Boolean circuits and/or one-way functions with $\mathbb{F}_2$ operations (*e.g.* AES, Rain, SDitH over $\mathbb{F}_2$) do not support our framework of MPCitH based on (low-)threshold LSSS. Same for the scheme recently proposed in [FMRV22b] and which achieves short communication using secret sharing over the integers: this idea is not compatible with our approach.

In the following, we present two applications of our strategy with Shamir's secret sharing as threshold LSSS:

- we first apply our general strategy to the SDitH signature scheme [FJR22b] to obtain a new variant with faster signing and verification times;
- we then apply our batching technique (see Section 8.4.2) to the SDitH scheme (batch proofs for syndrome decoding) and to the Limbo proof system [DOT21] (batched proofs for general arithmetic circuits).

8.5.1. Application to the SDitH Signature Scheme

We can transform the zero-knowledge proofs of knowledge described in Section 8.3 into signature schemes using the Fiat-Shamir heuristic [FS87]. We describe the signature scheme obtained when following this approach for the 5-round case (*i.e.* for $t = 1$ iteration in the MPC protocol) in Appendix C and further prove that this scheme achieves EUF-CMA security in the random oracle model.

In the following, we focus on the signature scheme obtained when applying this approach to the SDitH protocol, presented in Chapter 5. The underlying MPC protocol involves three fields $\mathbb{F}_{\mathrm{SD}} \subseteq \mathbb{F}_{\mathrm{poly}} \subseteq \mathbb{F}_{\mathrm{points}}$ which are extensions of each other and such that $\mathbb{F}_{\mathrm{SD}}$ is the base field of the SD instance. This MPC protocol fits the model introduced in Section 8.2.1 (see Protocol 10), with the number t of loop iterations equal to 1. Using the same notation as in Section 8.3, the proof size involves the following quantities:

- $\mathsf{inputs} = k \cdot \log_2 |\mathbb{F}_{\mathrm{SD}}| + 2w \cdot \log_2 |\mathbb{F}_{\mathrm{poly}}| + t' \cdot \log_2 |\mathbb{F}_{\mathrm{points}}|$,
- $\mathsf{unif} = 2 \cdot d \cdot t' \cdot \log_2 |\mathbb{F}_{\mathrm{points}}|$,
- $\mathsf{comm} = 2 \cdot d \cdot t' \cdot \log_2 |\mathbb{F}_{\mathrm{points}}|$,

where (m, k, w) are the syndrome decoding parameters and (d, t') are additional parameters (see Chapter 5 for more details). The signature size (in bits), including the 2λ-bit salt, is then given by

$$\textsc{Size} = 6\lambda + \tau \cdot \Big(\mathsf{inputs} + \mathsf{comm} + \lambda \cdot \log_2(N) + 2\lambda\Big)$$

Table 8.3.: Generic MPC-in-the-Head Techniques and Signature Schemes from MPC-in-the-Head Techniques.

Scheme Name	Year	Format				Hard Problem	Signature Size	
		Base Ring	Model	#Rounds	Helper		Original	Normalized
ZKBoo [GMO16]	2016	Any ring	✗	3	✗	Any $(2,3)$-decomposition circuit	-	-
ZKB++ [CDG+17]	2017	Any ring	✗	3	✗		-	-
Ligero [AHIV17]	2017	Any field	✗	5	✗	Any arithmetic circuit C (additions and multiplications)	-	-
Ligero++ [BFH+20]	2020	Any field	✗	5	✗		-	-
KKW [KKW18]	2018	Any ring	✓	3 or 5	✓		-	-
BN [BN20]	2020	Any field	✓	5	✗		-	-
Limbo [DOT21]	2021	Any field	✓	$\log \lvert C \rvert$	✗		-	-
BN++ [KZ22]	2021	Any field	✓	5	✗		-	-
Helium [KZ22]	2021	Any field	✓	7	✗		-	-
Picnic1 [CDG+17]	2016	$\mathbb{F}_2$	✗	3	✗	LowMC (partial)	32.1	-
Picnic2 [KKW18]	2018	$\mathbb{F}_2$	✓	3	✓		12.1	12.1 – 15.4
Picnic3 [KZ20b]	2019	$\mathbb{F}_2$	✓	3	✓	LowMC (full)	12.3	11.1 – 13.7
Helium+LowMC [KZ22]	2022	$\mathbb{F}_2$ ($\mathbb{F}_8$)	✓	7	✗		5.4 – 12.1	6.4 – 9.2
BBQ [DDOS19]	2020	$\mathbb{F}_2$ ($\mathbb{F}_{256}$)	✓	3	✓	AES	30.9	31.8 – 48.6
Banquet [BDK+21b]	2021	$\mathbb{F}_2$ ($\mathbb{F}_{256}$)	✓	7	✗		13.0 – 19.3	13.0 – 17.1
Limbo-Sign [DOT21]	2021	$\mathbb{F}_2$ ($\mathbb{F}_{256}$)	✓	13	✗		14.2 – 17.9	14.2 – 17.9
Helium+AES [KZ22]	2022	$\mathbb{F}_2$ ($\mathbb{F}_{256}$)	✓	7	✗		9.7 – 17.2	9.7 – 14.4
LegRoast [BD20]	2020	$\mathbb{F}_{2^{127}-1}$	✓	7	✗	Legendre PRF	12.2 – 16.0	12.2 – 14.8
PorcRoast [BD20]	2020	$\mathbb{F}_{2^{127}-1}$	✓	7	✗	Higher-Power Residue Characters	6.3 – 8.6	6.3 – 7.8
Rainier-128 [DKR+21]	2021	$\mathbb{F}_2$ ($\mathbb{F}_{128}$)	✓	5	✗	Rain [DKR+21]	5.1 – 9.4	5.9 – 8.1
BN++Rain [KZ22]	2022	$\mathbb{F}_2$ ($\mathbb{F}_{128}$)	✓	5	✗		4.4 – 5.8	4.9 – 6.4
SDitH [FJR22b]	2022	$\mathbb{F}_2$	✓	5	✗	Syndrome Decoding over $\mathbb{F}_2$	11.8 – 17.0	10.9 – 15.6
	2022	$\mathbb{F}_{256}$	✓	5	✗	Syndrome Decoding over $\mathbb{F}_{256}$	8.3 – 11.5	8.3 – 11.5
[FMRV22b]	2022	$\mathbb{Z}$	✓	5	✓/✗	Subset-Sum Problem	21.1 – 33.2	24.3 – 34.8
	2022	$\mathbb{Z}$	✓	5	✗	BHH PRF [BHH01]	4.8	4.8 – 6.5

All the signature sizes are in kilobytes and target a security of 128 bits. The *original* signature sizes correspond to values given by the underlying articles. The *normalized* signature sizes are given for a range of $8-32$ parties (in the underlying MPC protocol) when there is a preprocessing phase and for a range of $32-256$ parties otherwise. The column "Model" indicates whether the underlying MPC protocol fits our general model.

where N is the number of MPC parties, τ the number of executions and λ is the security level. Its security is given by the attack of [KZ20a] and is equal to

$$\mathrm{cost}_{\mathrm{forge}} := \min_{\tau_1,\tau_2:\tau_1+\tau_2=\tau} \left\{ \frac{1}{\sum_{i=\tau_1}^{\tau} \binom{\tau}{i} p^i (1-p)^{\tau-i}} + N^{\tau_2} \right\}$$

where p is the false positive rate of the SDitH MPC protocol satisfying

$$p \leq \sum_{i=0}^{t'} \frac{\max_{\ell \leq (m+w)/d-1} \left\{ \binom{\ell}{i} \binom{|\mathbb{F}_{\mathrm{points}}|-\ell}{t'-i} \right\}}{\binom{|\mathbb{F}_{\mathrm{points}}|}{t'}} \left(\frac{1}{|\mathbb{F}_{\mathrm{points}}|} \right)^{t'-i} .$$

In Chapter 5, we presented the SDitH scheme with several parameter sets: some of them with $\mathbb{F}_{\mathrm{SD}} := \mathbb{F}_2$ and the others with $\mathbb{F}_{\mathrm{SD}} := \mathbb{F}_{256}$, some of them aiming for short signatures and the others aiming for fast signature/verification time.

We apply the ideas of Section 8.3 to this scheme using Shamir's secret sharing. Since the number N of parties is limited by the field size, $N \leq |\mathbb{F}_{\mathrm{SD}}|$,[3] we consider the instance with $\mathbb{F}_{\mathrm{SD}} := \mathbb{F}_{256}$ as base field. As explained previously, our MPCitH strategy with $(\ell+1, N)$-threshold LSSS does not make the signature smaller but substantially improves the signing and verification times. According to Section 8.3.4, we obtain signatures of size (in bits):

$$\textsc{Size} = 6\lambda + \tau \cdot \left(\ell \cdot (\mathsf{inputs} + \mathsf{unif}) + \mathsf{comm} + 2\lambda \cdot \ell \cdot \log_2 \frac{N}{\ell} \right)$$

In Chapter 5, we chose p a bit lower than 2^{-64} which implies that the number of executions τ just needs to be increased by one while turning to the non-interactive case. Here, by taking $\ell > 1$, we decrease τ and each execution has more impact on the communication cost. Therefore we take p negligible in order to avoid to increase τ while turning to the non-interactive setting. At the same time, it means that we can apply an idea from Limbo [DOT21] which consists in using the same first challenge for all parallel executions of the underlying MPC protocol to save communication (due to the fact that the plain broadcasted values will be the same across all the parallel executions).

As explained in Section 8.3.3 and formally analyzed in our proof of soundness (see Appendix B), in case of a non-negligible false positive rate, an adversary can try to forge a proof of knowledge by committing an invalid sharing of the witness (which is not possible in the case of additive sharing). This ability is also exploitable in the non-interactive setting while considering the attack of [KZ20a]. In order to thwart this type of attack on our variant of the SDitH scheme, we make the conservative choice of taking a false positive rate p satisfying

$$\tau \cdot \binom{N}{\ell+1} \cdot p \leq 2^{-128} .$$

This way, the probability that a single witness encoded by a subset of $\ell+1$ shares among N leads to a false positive (in at least one of the τ iterations) is upper bounded by 2^{-128} so that any attack strategy which consists in guessing (even partially) the first challenge shall cost at least 2^{128} operations. Then, we simply need to take τ such that $\binom{N}{\ell}^{\tau} \geq 2^{128}$ in order to

[3] The Shamir's secret sharing over a field $\mathbb{F}$ can have at most $|\mathbb{F}|-1$ shares (one share by non-zero evaluation point), but we can have an additional share by defining it as the leading coefficient of the underlying polynomial (*i.e.* using the point at infinity as evaluation point).

achieve a 128-bit security in the non-interactive setting. We propose three possible instances of our scheme for $\ell \in \{1, 3, 7\}$ and $N = 256$ (the maximal number of parties).

We have implemented this variant of the SDitH signature scheme in C. In our implementation, the pseudo-randomness is generated using AES in counter mode and the hash function is instantiated with SHAKE. We have benchmarked our implementation on a 3.8 GHz Intel Core i7 CPU with support of AVX2 and AES instructions. All the reported timings were measured on this CPU while disabling Intel Turbo Boost. Instead of emulating $\ell + 1$ parties as described in Protocol 14, the implementation runs the MPC protocol directly on the coefficients of the degree-ℓ polynomials of the Shamir's secret sharing, thus avoiding costly polynomial evaluations and interpolations.

Table 8.4 summarizes the obtained performance for the different sets of parameters. We observe that the verification time is significantly smaller –between one and two orders of magnitude– than for the original scheme. This was expected since the verifier only emulates the views of ℓ parties instead of $N - 1$. The gain in signing time is more mitigated: even if the signer emulates only few parties, she must still commit the input shares of N parties. Nevertheless, the number of executions τ decreases while increasing the threshold ℓ, which further improves the signing time. The resulting signatures are slightly larger than for the original scheme with the same number of parties (the short version), but our scheme gains a factor 10 in signing and verification time. Compared to the fast version of the original signature scheme (which uses a lower number of parties $N = 32$) and for similar signature size, our scheme gains a factor 3 in signing time and a factor 10 in verification time.

Table 8.4.: Parameters, performance and comparison. The parameters for [FJR22b] and our scheme are $(m, k, w) = (256, 128, 80)$ and $\mathbb{F}_{\text{SD}} = \mathbb{F}_{\text{poly}} = \mathbb{F}_{256}$.

Scheme	N	τ	ℓ	t'	$\lvert\mathbb{F}_{\text{points}}\rvert$	$\lvert\mathsf{sgn}\rvert$	t_{sgn}	t_{verif}
Our scheme	256	16	1	3	2^{64}	10.47 KB	7.1 ms	0.46 ms
	256	6	3	3	2^{64}	**9.97 KB**	3.2 ms	**0.38 ms**
	256	3	7	4	2^{64}	11.10 KB	**2.5 ms**	0.47 ms
[FJR22b] - Var3f	32	27	-	5	2^{24}	11.5 KB	6.4 ms	5.9 ms
[FJR22b] - Var3s	256	17	-	5	2^{24}	8.26 KB	30 ms	27 ms
Banquet (AES)	16	41	-	1	2^{32}	19.3 KB	6.4 ms	4.9 ms
	255	21	-	1	2^{48}	13.0 KB	44 ms	40 ms
Limbo-Sign (AES)	16	40	-	-	2^{48}	21.0 KB	2.7 ms	2.0 ms
	255	24	-	-	2^{48}	14.2 KB	29 ms	27 ms
Helium+AES	17	31	-	1	2^{144}	17.2 KB	6.4 ms	5.8 ms
	256	16	-	1	2^{144}	9.7 KB	16 ms	16 ms
SPHINCS$^+$-128f	-	-	-	-	-	16.7 KB	14 ms	1.7 ms
SPHINCS$^+$-128s	-	-	-	-	-	7.7 KB	239 ms	0.7 ms

Note: Timings for SDitH and our scheme have been benchmarked on a 3.8 GHz Intel Core i7. Timings for Banquet, Helium and SPHINCS$^+$ have been benchmarked on a 3.6 GHz Intel Xeon W-2133 CPU [BDK+21b; KZ22]. Timings for Limbo have been benchmarked on a 3.1 GHz Intel i9-9900 CPU [DOT21].

Table 8.4 further compares our scheme with recent MPCitH schemes based on AES (both AES and SD for random linear codes being deemed as a conservative assumption) as well

as with SPHINCS$^+$ [ABB+22] as a baseline conservative scheme. We can observe that our scheme outperforms AES-based candidates for comparable signature sizes (around 10 KB). In particular, compared to Helium+AES [KZ22], signing is 5 times faster with our scheme while verification is 40 times faster. Fast versions of those schemes have signatures about twice larger, while being still slower than ours in signing and verification. Compared to SPHINCS$^+$, our scheme achieves slightly better verification time and much better trade-offs for signature size vs. signing time. Some other MPCitH signature schemes reported in Table 8.3 achieve smaller signature sizes (down to 5KB) but they are based on less conservative assumptions (LowMC, Rain, BHH PRF). Yet none of these schemes achieve fast verification as SPHINCS$^+$ or our scheme.

Let us remark that, as in Chapter 5, we did not investigate software optimizations (like vectorization or bitslicing), so there is room for improvement in the reported performance.

8.5.2. Application of the Batching Strategy

8.5.2.1. Application to SDitH.

As the first application of the batching strategy described in Section 8.4.2, we show how to batch proofs of knowledge for the syndrome decoding problem. Specifically, we consider a context where, from a public parity-check matrix $H \in \mathbb{F}_{\mathrm{SD}}^{(m-k)\times m}$ and vectors $y_1, \ldots, y_u \in \mathbb{F}_{\mathrm{SD}}^{m-k}$, one wants to batch proofs of knowledge for u small-weight syndromes $x_1, \ldots, x_u \in \mathbb{F}_{\mathrm{SD}}^{m}$ such that

$$\forall i \in [1:u], y_i = H \cdot x_i \ .$$

For this purpose, we apply our batching strategy to the SDitH scheme (see Chapter 5). However, in its original version, this scheme is not compatible with the application of batched Shamir's secret sharing (on which rely our batching strategy) since it involves multiplications by witness-dependent scalars. Those appear in the final product verification based on the [BN20] protocol. To overcome this issue, we propose a tweak of the SDitH scheme to verify the final multiplication triple without relying on [BN20]. Specifically, to prove that three sharings $[\![a]\!]$, $[\![b]\!]$ and $[\![c]\!]$ verify $c = a \cdot b$ ($a, b, c \in \mathbb{F}_{\mathrm{points}}$), the parties proceed as follows:

- they get as hints
$$\begin{array}{ll} [\![r_1]\!] & \text{where } r_1 \leftarrow \mathbb{F}_{\mathrm{points}} \\ [\![r_2]\!] & \text{where } r_2 \leftarrow \mathbb{F}_{\mathrm{points}} \\ [\![h_1]\!] & \text{where } h_1 = r_1 \cdot b + r_2 \cdot a \\ [\![h_2]\!] & \text{where } h_2 = r_1 \cdot r_2, \end{array}$$
- they get a random point $\xi \leftarrow \mathbb{F}_{\mathrm{points}} \backslash \{0\}$,
- they locally compute and broadcast
$$\begin{cases} [\![v_1]\!] & = \xi \cdot [\![r_1]\!] + [\![a]\!] \\ [\![v_2]\!] & = \xi \cdot [\![r_2]\!] + [\![b]\!] \\ [\![v_3]\!] & = \xi^2 \cdot [\![h_2]\!] + \xi \cdot [\![h_1]\!] + [\![c]\!], \end{cases}$$
- they output Accept if $v_1 \cdot v_2 = v_3$, Reject otherwise.

If $c = a \cdot b$ and the hints are correctly computed, then the polynomial

$$P(X) := (r_1 X + a) \cdot (r_2 X + b) - (h_2 X^2 + h_1 X + c)$$

equals 0 so that $v_1 \cdot v_2 - v_3 = P(\xi) = 0$ and the parties accept. If $c \neq a \cdot b$, then $P(X)$ is different from zero. Thus according to the Schwartz-Zippel Lemma, the probability that the parties accept is at most $\frac{2}{|\mathbb{F}_{\text{points}}|-1}$ (for any adversarial choice of the hints).

Replacing the BN20 protocol with the above one in the SDitH scheme increases the number of rounds in the underlying zero-knowledge protocol from 5 to 7. Indeed the challenge ξ cannot be sent at the same time as the evaluation point anymore, since $[\![h_1]\!]$ and $[\![h_2]\!]$ must be committed before receiving ξ. We then restrict our variant of the SDitH protocol to a single evaluation point ($t' = 1$) to ease the analysis of the [KZ20a] attack while turning to the non-interactive setting. Using the same notations as Section 8.3, the new proof size involves the following quantities:

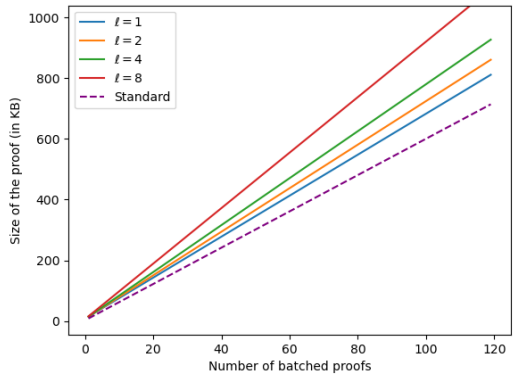
- $\mathsf{inputs} = k \cdot \log_2 |\mathbb{F}_{\text{SD}}| + 2w \cdot \log_2 |\mathbb{F}_{\text{poly}}| + \underbrace{2 \cdot \log_2 |\mathbb{F}_{\text{points}}|}_{[\![h_1]\!],[\![h_2]\!]}$,
- $\mathsf{unif} = \underbrace{2 \cdot \log_2 |\mathbb{F}_{\text{points}}|}_{[\![r_1]\!],[\![r_2]\!]}$,
- $\mathsf{comm} = \underbrace{2 \cdot \log_2 |\mathbb{F}_{\text{points}}|}_{[\![v_1]\!],[\![v_2]\!]}$,

We apply the three batching strategies described in Section 8.4.2 for the syndrome decoding parameters $(m, k, w) = (256, 128, 80)$ and using the field extension $\mathbb{F}_{\text{points}} := \mathbb{F}_{2^{192}}$. Figure 8.1 illustrates the resulting performance in terms of proof size and number of party emulations. Batching with additive sharings (using the naive strategy) with $N = 256$ is represented by the dashed line for which the achieved amortized cost per statement is around 6 KB.

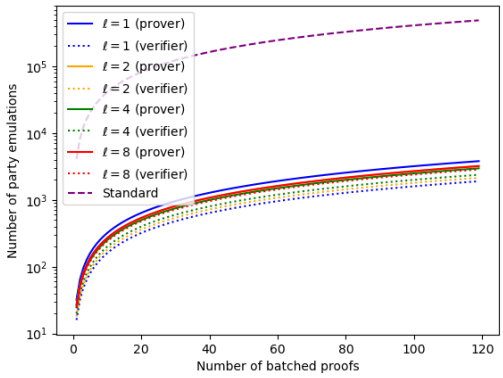
Naive strategy: When using the naive strategy with low-threshold Shamir's secret sharing (for $N = 256$), the proof size is a bit larger than for the additive case, but the number of emulations is divided by more than 100.

SSS-based strategy: When the number u of batched proofs is small enough ($u \leq 80$), this strategy outperforms the additive case in terms of proof size. But it grows exponentially as u increases (slower when ℓ is larger). As explained in Section 8.4.2, this behavior is amplified when the underlying field is small. Here the number N of parties is limited by $N \leq |\mathbb{F}_{256}| + 1 - u = 257 - u$. For instance, when $u = 110$ and $\ell = 2$, the soundness error ε is approximately 0.57 with N maximal ($N = 147$), requiring a high number of executions $\tau \geq 150$.

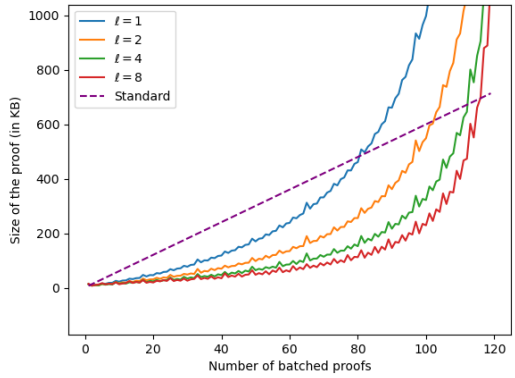
Hybrid strategy: As expected, for such a context with limited N, the hybrid approach gives the best results. We get an amortized proof size of around 2.3 KB when $\ell = 1$ and around 0.83 KB when $\ell = 8$.



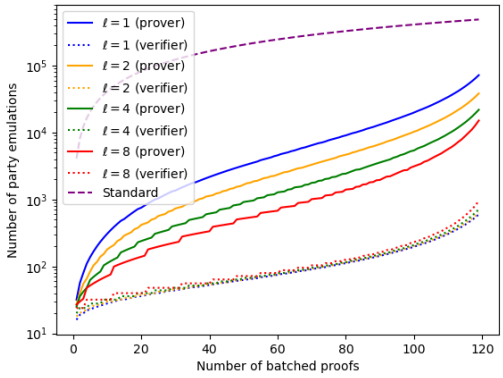
(a) Naive strategy: proof size



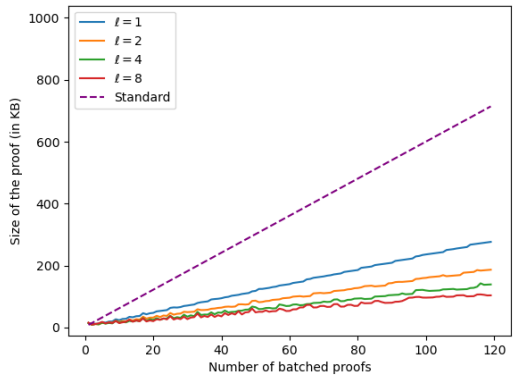
(b) Naive strategy: # party emul.



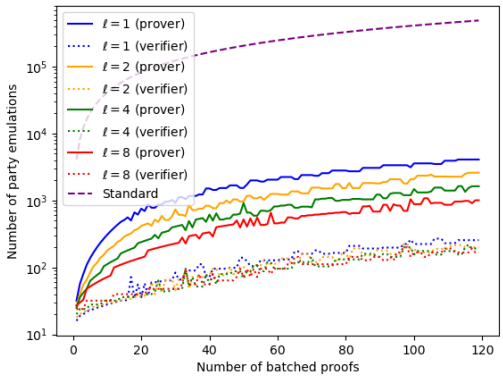
(c) SSS-based strategy: proof size



(d) SSS-based strategy: # party emul.



(e) Hybrid strategy: proof size



(f) Hybrid strategy: # party emul.

Figure 8.1.: Performance of the three batching strategies applied to our tweaked SDitH scheme.

8.5.2.2. Application to Limbo.

We now apply our batching strategy to Limbo [DOT21], a proof system for general arithmetic circuits. In Limbo, sharings are multiplied by scalars in three different contexts:

- After sampling a random challenge $\{r_i\}_{i=1}^m$, a set of multiplicative triples $([\![x_i]\!], [\![y_i]\!], [\![z_i]\!])_{i=1}^m$ is verified as an inner product $\langle a, b\rangle = c$ where

$$[\![a]\!] = \begin{pmatrix} r_1 \cdot [\![x_1]\!] \\ \vdots \\ r_m \cdot [\![x_m]\!] \end{pmatrix}, \quad [\![b]\!] = \begin{pmatrix} [\![y_1]\!] \\ \vdots \\ [\![y_m]\!] \end{pmatrix}, \quad [\![c]\!] = \sum_{i=1}^m r_i \cdot [\![z_i]\!].$$

  If $\langle a, b\rangle = c$ for a random choice of $\{r_i\}_{i=1}^m$, then we can deduce that $x_i \cdot y_i = z_i$ for every $i \in [1 : m]$ with high probability.[4]

- To interpolate a polynomial f of degree d such that

$$[\![f]\!](e_i) = [\![x_i]\!] \quad \text{for} \quad i \in \{0, \ldots, d\}$$

  where $e_0, \ldots, e_d$ are fixed distinct public points. With the Lagrange interpolation formula, we get that

$$[\![f]\!](X) = \sum_{i=0}^d [\![x_i]\!] \cdot \underbrace{\prod_{j=0, j\neq i}^m \frac{X - e_j}{e_i - e_j}}_{\ell_i(X)}$$

  where $\{\ell_i(X)\}_i$ are public constant polynomials.

- To evaluate a polynomial $[\![f]\!](X) := \sum_{i=0}^d [\![a_i]\!] X^i$ in a challenge point s:

$$[\![f(s)]\!] = \sum_{i=0}^d [\![a_i]\!] \cdot s^i.$$

In all those three contexts, the scalars which multiply the shares are constant or derived from random challenges. In particular, they are witness-independent which makes our batching strategy applicable (see Section 8.4.2 for details).

Let us briefly explain the high-level idea of the MPC protocol underlying Limbo. This protocol aims to check a list of m multiplicative triples. To proceed, it first converts the list into an inner product of dimension m (as recalled above), and then repeats a compression step that reduces the inner product dimension until reaching a small enough instance. Given a compression parameter k, the compression step works as follows (the process is illustrated in Figure 8.2):

1. it splits the current inner product into k smaller inner products,

2. it compresses the k inner products into a single inner product. Let us consider the k values $v^{(1)}$, ..., $v^{(k)}$ of these inner products in the same position. To compress these k values into a single one v, it builds the polynomial P of degree $k-1$ such that

$$\forall i \in \{1, \ldots, k\}, \; P(i) = v^{(i)}$$

   and it computes v as $P(\xi)$ where ξ is a random verifier challenge.

$$\left\langle \begin{pmatrix} x_1^{(1)} \\ \vdots \\ x_\ell^{(1)} \\ \hline \vdots \\ \hline x_1^{(k)} \\ \vdots \\ x_\ell^{(k)} \end{pmatrix}, \begin{pmatrix} y_1^{(1)} \\ \vdots \\ y_\ell^{(1)} \\ \hline \vdots \\ \hline y_1^{(k)} \\ \vdots \\ y_\ell^{(k)} \end{pmatrix} \right\rangle = z^{(1)} + \ldots + z^{(k)}$$

$$\text{Step 1} \Big\Downarrow \text{Splitting}$$

$$\left\langle \begin{pmatrix} x_1^{(1)} \\ \vdots \\ x_\ell^{(1)} \end{pmatrix}, \begin{pmatrix} y_1^{(1)} \\ \vdots \\ y_\ell^{(1)} \end{pmatrix} \right\rangle = z^{(1)}, \ \ldots, \ \left\langle \begin{pmatrix} x_1^{(k)} \\ \vdots \\ x_\ell^{(k)} \end{pmatrix}, \begin{pmatrix} y_1^{(k)} \\ \vdots \\ y_\ell^{(k)} \end{pmatrix} \right\rangle = z^{(k)}$$

$$\text{Step 2} \Big\Downarrow \text{Compression}$$

$$\left\langle \begin{pmatrix} x_1 \\ \vdots \\ x_\ell \end{pmatrix}, \begin{pmatrix} y_1 \\ \vdots \\ y_\ell \end{pmatrix} \right\rangle = z$$

where $v = \sum_{i=1}^{k} v^{(i)} \cdot \prod_{j=1, j\neq i}^{k} \frac{\xi - j}{i - j}$
with x_u, y_u or z for v
and with ξ a verifier challenge.

Figure 8.2.: Limbo's compression step - from dimension $k\ell$ to dimension ℓ

Let us consider the case where Limbo will always finish on a final inner product of dimension 1, after $\rho := \lceil \log_k(m) \rceil$ compression steps. Using the same notations as in Section 8.3, the proof size involves the following quantities:

- $\mathsf{inputs} = \underbrace{(|w| + |C|) \cdot \log_2 \mathbb{F}}_{\text{Protocol inputs}} + \underbrace{\rho \cdot (k-1) \cdot \log_2 \mathbb{G}}_{\text{Splitting cost}} + \underbrace{(\rho \cdot (k-1) + 2) \cdot \log_2 \mathbb{G}}_{\text{Compression cost}},$
- $\mathsf{unif} = 2 \cdot \log_2 \mathbb{G}$,
- $\mathsf{comm} = 2 \cdot \log_2 \mathbb{G}$,

[4]In Limbo, the challenge is of the form $\{r_i\}_{i=1}^m := \{r^i\}_{i=1}^m$ for a random $r \in \mathbb{F}$.

where $|w|$ is the size of the witness (*i.e.* the circuit input), $|C|$ is the number of multiplications in the circuit, $\mathbb{F}$ is the base field of the circuit, $\mathbb{G}$ is a field extension of $\mathbb{F}$ and k is the compression factor (see [DOT21] for more details). Limbo's proof size (in bits) is then given by:

$$\textsc{Size} = (\rho + 2) \cdot 2\lambda + \tau \cdot \Big(\mathsf{inputs} + \mathsf{comm} + \lambda \cdot \log_2(N) + 2\lambda\Big)$$

where N is the number of MPC parties, τ is the number of executions and λ is the security level. Limbo's soundness error is given by:

$$\left(\frac{1}{N} + p_k \cdot (1 - \frac{1}{N})\right)^{\tau}$$

where p_k is the false positive rate of the underlying MPC protocol (which depends on k) – see [DOT21, Proposition 5].

Let us assume that we want to batch u proofs for an arbitrary arithmetic circuit C defined over a base field $\mathbb{F}$. We use the Limbo proof system in interactive mode targeting a security of 40 bits, which is one of the considered use-cases of the original paper [DOT21]. If $\mathbb{F}$ is large enough, we can apply our batching strategy described in Section 8.4.2. Tables 8.5 and 8.6 summarize the achieved performance for the base fields $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^{32}}$ and for circuit sizes 2^8 and 2^{16}. While for $\mathbb{F}_{2^8}$ we are limited in the number of parties since $N \leq |\mathbb{F}| + 1 - u$, this is not an issue for $\mathbb{F}_{2^{32}}$. From these tables, we can observe that our batching strategy drastically reduces the amortized cost in terms of proof size. Considering a batching of $u = 100$ statements, the amortized proof size is more than 10 times smaller than the standard version in all the considered settings and this ratio is closer to 1/20 for the larger circuit or field.

Table 8.5.: Performance of batched Limbo proofs for arithmetic circuits on $\mathbb{F}_{2^8}$ in the interactive setting (40-bit of security).

u	ℓ	$\lvert w\rvert + \lvert C\rvert = 2^8$					$\lvert w\rvert + \lvert C\rvert = 2^{16}$				
		N	u/ν	τ	size	size/u	N	u/ν	τ	size	size/u
1	ADD	256	-	6	5.5	5.55	256	-	6	389.4	389.42
	1	256	1	6	9	9.22	256	1	6	398	397.59
100	1	223	34	16	61	0.61	207	50	21	2762	27.62
	4	223	34	4	58	0.58	223	34	4	3151	31.51
10000	1	225	32	15	5865	0.59	203	54	22	269290	26.93
	4	210	47	5	5434	0.54	200	57	6	277894	27.79

Note: The compression factor k of Limbo is 16 and the extension field $\mathbb{G}$ is $\mathbb{F}_{2^{64}}$ when $\ell = 1$ and $\mathbb{F}_{2^{96}}$ when $\ell = 4$. All the sizes are given in kilobytes. $\#\mathcal{P}$ and $\#\mathcal{V}$ correspond respectively to the number of emulated parties for the prover and the verifier. The first row ($\ell = \textsc{add}$) is the baseline Limbo scheme with additive sharing.

Table 8.6.: Performance of batched Limbo proofs for arithmetic circuits on $\mathbb{F}_{2^{32}}$ in the interactive setting (40-bit of security).

u	ℓ	$\lvert w\rvert + \lvert C\rvert = 2^8$					$\lvert w\rvert + \lvert C\rvert = 2^{16}$				
		N	u/ν	τ	size	size/u	N	u/ν	τ	size	size/u
1	ADD	256	-	6	10.0	10.03	256	-	6	1536.9	1536.92
	1	256	1	6	14	13.72	256	1	6	1550	1549.59
100	1	256	50	18	96	0.96	256	100	31	8053	80.53
	4	256	100	8	91	0.91	256	100	8	8284	82.84
10000	1	256	55	19	8170	0.82	256	100	31	801486	80.15
	4	256	74	6	7130	0.71	256	88	7	823455	82.35

Note: The compression factor k of Limbo is 16 and the extension field $\mathbb{G}$ is $\mathbb{F}_{2^{64}}$ when $\ell = 1$ and $\mathbb{F}_{2^{96}}$ when $\ell = 4$. All the sizes are given in kilobytes. $\#\mathcal{P}$ and $\#\mathcal{V}$ correspond respectively to the number of emulated parties for the prover and the verifier. The first row ($\ell = \text{ADD}$) is the baseline Limbo scheme with additive sharing.

8.6. Conclusion

In this chapter, we have proposed a new way to transform an MPC protocol into a zero-knowledge proof of knowledge. We have shown how using low-threshold LSSS enables us to produce faster schemes. When building signature schemes, it tends to give larger signatures compared to the standard approach based on additive sharings. In other contexts (larger circuits, batching), it can also achieve shorter communication.

In November 2022 (one month after the release of this work), a concurrent and independent work [AGH+23] proposes a computational optimization of the MPCitH transformation based on additive sharing (see Section 3.1.2.2). Using the so-called hypercube technique, the authors show that the prover can emulate the entire MPC protocol by performing the computation of only $\log_2 N + 1$ parties instead of N, while keeping the same communication cost. While both the hypercube approach and our work enable us to significantly speed up MPCitH-based schemes, they provide different interesting trade-offs and their relative performance depends on the context.

Chapter 9.

From Research to Specification

After the transition to the fourth round of the NIST PQC standardization process, NIST decided to re-open a call for additional digital signature proposals. The deadline for this call was the 1st of June 2023. In the context of this thesis, we submitted several schemes that we will briefly describe in this chapter.

Moreover, in the last months of the thesis, we built a library to ease the development of the MPCitH-based schemes. We will also present this library in this chapter.

Contents

9.1. NIST Call for Additional Signatures **166**
9.2. MPC-in-the-Head Library **169**

9.1. NIST Call for Additional Signatures

In September 2022, the NIST released a call for additional digital signature proposals to be considered in the PQC standardization process [NIS22].

> NIST is primarily looking to diversify its signature portfolio, so signature schemes that are not based on structured lattices are of greatest interest. NIST would like submissions for general-purpose signature schemes, as well as those which have short signatures and fast verification. Submissions should not significantly overlap with signature schemes that have already been selected by NIST for standardization. At a minimum:
>
> - lattice-based schemes should provide at least one large performance advantage over both Dilithium and Falcon.
> - non-lattice-based algorithms should provide at least one large performance advantage over SPHINCS+.
>
> – NIST, September 2022

Since the schemes proposed in this thesis are competitive with the existing post-quantum state of the art, we decided to submit some of them to the NIST call, in collaboration with several teams. More precisely, we submitted four signature schemes, each of them relying on a specific hard problem:

- SD-in-the-Head, relying on the syndrome decoding problem for random linear codes in *Hamming metric*;
- MIRA, relying on the MinRank problem;
- RYDE, relying on the syndrome decoding problem for random linear codes in *rank metric*;
- MQOM, relying on the unstructured multivariate quadratic problem.

There are two reasons why we submitted several submissions:

- Each of these schemes has its own specificities leading to different advantages:
  - MIRA and RYDE produce the shortest signatures;
  - SD-in-the-Head could be considered as the more conservative scheme;
  - RYDE has the shortest uncompressed public key (which is more convenient for embedded systems);
  - SD-in-the-Head (threshold variant) has the fastest verification algorithm;
  - MQOM has sizes close to those of MIRA and RYDE, without relying on the rank metric.
- The MPC-in-the-Head state of the art remains an active research field and it is not stable yet. There will probably be new results leading to more efficient schemes (in term of sizes and/or running times). Each submission could be impacted differently by them. Currently, the shortest sizes are achieved by MIRA and RYDE, but this could change in the future.

In what follows, we briefly present the design choices made for each of these submissions.

9.1.1. SD-in-the-Head Signature Scheme

The SD-in-the-Head signature scheme has been submitted to the NIST in collaboration with

Carlos Aguilar Melchor, Nicolas Gama, Shay Gueron,
James Howe, David Joseph, Antoine Joux, Edoardo Persichetti,
Tovohery H. Randrianarisoa, Matthieu Rivain, and Dongze Yue.

The security of this scheme relies on the hardness of the syndrome decoding problem for random linear codes in Hamming metric (see Definition 4.1.1). The used MPC protocol is a variant of the one presented in Chapter 5:

- The original protocol needs to compute a Lagrange interpolation to build the polynomial S from the secret x. We can avoid this interpolation by tweaking the definition of y in the public key. Instead of defining $y := Hx$, we can define:

  $$y := HVx$$

  where V is the matrix satisfying:

  $$S = \text{LagrangeInterpolation}(x) \Leftrightarrow s = Vx$$

  for s the vector of coefficients of S. Let us remark that, for a uniformly random linear code $\mathcal{C}_H$ represented by the matrix H as parity-check matrix, the linear code $\mathcal{C}_{HV}$ represented by HV is also uniformly random. This is because V is an invertible $m \times m$ matrix. This trick enables us to avoid all the polynomial interpolations in the signing and verification algorithms.

- In the original protocol, the random points $r_1, \ldots, r_t \in \mathbb{F}_{\text{points}}$ sent by the verifier to check the polynomial relation $S \cdot Q = F \cdot P$ should be distinct. Here we remove this constraint on $\vec{r}$ to make the scheme simpler and less prone to implementation errors. While this tweak slightly increases the false positive probability p (*i.e.* the probability that the protocol outputs Accept for an invalid input witness), this increase is small enough and does not impact much the security. The false positive probability of this tweaked protocol (taking r uniformly at random from $\mathbb{F}^t_{\text{points}}$) is:

  $$p := \sum_{i=0}^{t} \binom{t}{i} \left(\frac{m+w-1}{|\mathbb{F}_{\text{points}}|}\right)^i \left(1 - \frac{m+w-1}{|\mathbb{F}_{\text{points}}|}\right)^{t-i} \left(\frac{1}{|\mathbb{F}_{\text{points}}|}\right)^{t-i}.$$

We propose two variants of our signature scheme:

- The short variant (also named as the hypercube variant) lowers the communication cost to produce short signatures. It relies on the MPCitH transformation for the broadcast-based multiparty computation model with additive sharing (see Section 3.1.2.2, optimized by the hypercube technique [AGH+23]), with $N := 256$ parties.

- The fast variant (also named as the threshold variant) lowers the computational cost to get a fast signature computation. It relies on the transformation for the low-threshold broadcast-based multiparty computation model (see Section 3.1.3), with $N := q$ parties where q is the size of the base field.

We also consider the two fields for the syndrome decoding problem: $\mathbb{F}_{251}$ and $\mathbb{F}_{256}$. The prime field will lead to a more efficient scheme for the fast variant since the computational bottleneck of the used MPCitH transformation is due to the arithmetics. However, $\mathbb{F}_{256}$ will lead to a more efficient scheme for the short variant since its computational bottleneck is the usage of the symmetric primitives (as the pseudo-random generation). Even if it would arguably give a more conservative code-based scheme, we did not consider the binary field $\mathbb{F}_2$ since the resulting signature sizes are not competitive.

We refer the reader to the scheme's specification [AFG+23] for the precise description of the submitted scheme.

9.1.2. MIRA Signature Scheme

The MIRA signature scheme has been submitted to the NIST in collaboration with

Nicolas Aragon, Magali Bardet, Loïc Bidoux,
Jesús-Javier Chi-Domínguez, Victor Dyseryn, Philippe Gaborit,
Romaric Neveu, Matthieu Rivain, and Jean-Pierre Tillich.

The security of this scheme relies on the hardness of the MinRank problem (see Definition 7.4.3). The used MPC protocol is the one in Section 7.4.2 based on linearized polynomials.

We propose two variants of our signature scheme. Both rely on the MPCitH transformation for the broadcast-based multiparty computation model with additive sharing (see Section 3.1.2.2, optimized by the hypercube technique [AGH+23]). The short variant lowers the communication cost to produce short signatures by taking $N := 256$ parties, while the fast variant lowers the computational cost to get a fast signature computation by taking $N := 32$ parties. We did not design the fast variant with low-threshold LSSS as SD-in-the-Head, since it would require to work on larger fields (for which the MinRank cryptanalysis is much less mature).

We refer the reader to the scheme's specification [ABB+23b] for the precise description of the submitted scheme.

9.1.3. RYDE Signature Scheme

The RYDE signature scheme has been submitted to the NIST in collaboration with

Nicolas Aragon, Magali Bardet, Loïc Bidoux,
Jesús-Javier Chi-Domínguez, Victor Dyseryn, Philippe Gaborit, Antoine Joux,
Matthieu Rivain, Jean-Pierre Tillich, and Adrien Vinçotte.

The security of this scheme relies on the hardness of the rank syndrome decoding problem (see Definition 7.4.4). The used MPC protocol is a variant of the one in Section 7.4.3 based on linearized polynomials. We slightly change the underlying hardness assumption by assuming that 1 is in the linear subspace U generated by the coordinates of the secret vector. It implies that the q-polynomial L_U of Section 7.4.3 satisfies $L_U(1) = 0$, and we can use this relation to decrease the communication cost.

We propose two variants of our signature scheme. As for MIRA, both rely on additive sharings and use the hypercube technique. The short variant lowers the communication

cost to produce short signatures by taking $N := 256$ parties, while the fast variant lowers the computational cost to get a fast signature computation by taking $N := 32$ parties. As explained in [Fen22], it would be possible to design the fast variant with low-threshold sharings even when working on the binary field $\mathbb{F}_2$. However, the obtained size would be around 9.3 KB, while the proposed fast variant is around 7.4 KB. We could achieve smaller sizes with low-threshold if we work on larger fields, but as for MIRA, the corresponding cryptanalysis is much less mature on such fields.

We refer the reader to the scheme's specification [ABB+23a] for the precise description of the submitted scheme.

9.1.4. MQOM Signature Scheme

The MQOM signature scheme has been submitted to the NIST in collaboration with Matthieu Rivain. The security of this scheme relies on the hardness of the unstructured multivariate quadratic problem (see Definition 7.3.1). The used MPC protocol is an improvement of the protocol described in Section 7.3. As the construction in Section 7.3, it batches all the m equations into a single one. However, it differs in how the MPC protocol checks the batched equations: it uses a polynomial-based multiplication verification which asks the verifier for randomness twice (while the scheme in Section 7.3 used the sacrificing-based technique), obtaining a 7-round zero-knowledge proof.

We propose two variants of our signature scheme. As for MIRA and RYDE, both rely on additive sharings and use the hypercube technique, with $N := 256$ for the short variant and with $N := 32$ for the fast variant. Since we work on large fields (as $\mathbb{F}_{251}$), it would be possible to use low-threshold sharings. Unfortunately, the size degradation is large: while we have sizes below 7 KB with additive sharings, we get sizes of at least 14 KB with low-threshold sharings.

We refer the reader to the scheme's specification [FR23] for the precise description of the submitted scheme.

9.2. MPC-in-the-Head Library

9.2.1. Introduction

The MPC-in-the-Head paradigm has been an active research field for the last few years. Many signature schemes based on it have been proposed, including those of this thesis. Since the source codes of these schemes are similar except for the part about the MPC protocols, we decided to develop a unified MPC-in-the-Head library. The idea is to factorize as much as possible the common code of the MPCitH-based signatures. As long as they respect the expected API, users just need to implement some specific parts of the scheme and they can then rely on the library to get the desired signature scheme. It enables us to speed up the development of the MPCitH-based schemes. For example, we used this library to implement all the signatures proposed in Chapter 7. The library presents some other advantages:

- **Comparison**: it is not always easy to fairly compare the running times of two schemes in the literature since their implementations do not necessarily use the same primitives and/or the same level of optimizations. When two schemes are implemented via this

library, all the common primitives rely on the same code, and it thus provides a fairer comparison.

- **Detailed benchmark**: the library integrates some benchmark tools. The latter can provide a detailed benchmark of the schemes, which includes for example the timings due to the commitments, the pseudo-randomness... Thanks to them, we can easily estimate the running times of implemented schemes and identify their computational bottlenecks.
- **Versatility**: all the code blocks respect some API. It is thus easy to replace a primitive with another one and compare the impact on the running time, just by changing the configuration file.

Remark 9.2.1. *For the NIST call described in Section 9.1, the source code of MQOM and of the fast variant of SD-in-the-Head came from the library.*

9.2.2. Structure and configuration

The structure of the library is described in Figure 9.1. It is implemented through four layers:

1. **Symmetric Primitives.** The lowest layer provides implementations for hash functions, extendable output functions (XOFs) and pseudo-random bit generators. At the time of writing,
   - the available hash function and XOF are respectively SHA3 and SHAKE (their source codes come from the XKCP library, see `https://github.com/XKCP/XKCP`),
   - the two available pseudo-random bit generators are an XOF-based generator and an AES-based one.

   These three primitives are implemented in a `sym` folder.
2. **Arithmetics.** The `arithmetic` folder provides the implementation of operations over many fields. At the time of writing, the available fields are the fields with the following sizes:

$$2,\ 2^4,\ 2^8,\ 2^{16},\ 2^{31},\ 2^{32},\ 2^{64},\ 2^{96},\ 2^{128}$$
$$31,\ 31^6,\ 31^7,\ 31^8,\ 31^{10},\ 31^{11}$$
$$251,\ 251^2,\ 251^4,\ 251^5,\ 251^7,\ 251^{12},\ 251^{16}.$$

   For all the fields, the implemented operations are at least the addition, the subtraction, the multiplication and the pseudo-random number generation[1]. In some cases, other operations have been implemented (when they were needed to implement one of the schemes of this thesis), as the inversion.
3. **MPC Protocols.** The `mpc` folder contains all the source files that depend on the underlying MPC protocols. For each scheme, it should contain the implementation for the two following APIs:
   - `witness.h`. This file describes the prototypes of the functions that generate a couple instance-solution of the underlying hard problem. For example, in the case of MQOM, it should generate an instance of the multivariate quadratic problem with its solution.

[1] relying on the pseudo-random bit generator of the lower layer.

- `mpc.h`. This file describes the prototypes of the functions that emulate a party in the MPC protocol (and which generate the verifier challenges used in the MPC protocol).

4. **MPCitH Transformation.** The highest layer implements the signature schemes relying on the previous layers. The key generation uses the generation of the couple instance-solution provided by `witness.h`, while the signing and the verification use the party emulation provided by `mpc.h`. At the time of writing, the library provides the source code of several variants (*i.e.* MPCitH transformations):
   - **Traditional 5R**: transformation of an MPC protocol using additive sharings into a 5-round proof of knowledge, without the hypercube optimization.
   - **Hypercube 5R**: transformation of an MPC protocol using additive sharings into a 5-round proof of knowledge, with the hypercube optimization.
   - **Threshold 5R**: transformation of an MPC protocol using threshold LSSS into a 5-round proof of knowledge.
   - **Threshold 5R NFPR**: transformation of an MPC protocol using threshold LSSS into a 5-round proof of knowledge, assuming that the false positive rate of the MPC protocol is negligible.
   - **Hypercube 7R**: transformation of an MPC protocol using additive sharings into a 7-round proof of knowledge, with the hypercube optimization.

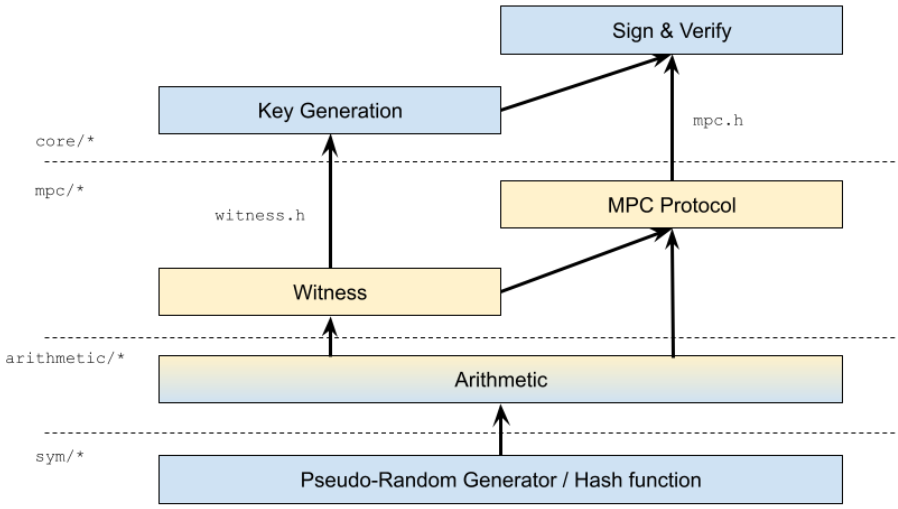


Figure 9.1.: Structure of the MPC-in-the-Head library. The yellow components are the parts that a user must implement when working on a new signature scheme, while the blue components are managed by the library.

To implement a new MPCitH-based scheme, one needs to

1. implement the field operations in the folder `arithmetic`, if they are not already available;

2. implement the generation of the couple instance-solution of the underlying hard problem, with respect to the API described by `witness.h`;
3. implement the emulation of a party of the MPC protocol, with respect to the API described by `mpc.h`;
4. build the configuration file of the scheme to select the desired symmetric primitives and the MPCitH variant.

Here is a (simplified) configuration file for a scheme implemented in the library:

```
export LIBMPCITH_SCHEME_PATH=sd/sdith_any
export LIBMPCITH_ARITH_LIBS=gf251
export LIBMPCITH_CONFIG_FLAGS=THRESHOLD_NFPR,GF251,L1
export LIBMPCITH_HASH_PATH=sha3/sha3-several
export LIBMPCITH_DRBG_PATH=hash
export LIBMPCITH_ADD_FLAGS=HASHX4,XOFX4,PRGX4
```

This configuration indicates that:

- the files implementing `witness.h` and `mpc.h` are in the folder `mpc/sd/sdith_any`,
- the scheme needs the files in `arithmetic/gf251` (meaning that it works over $\mathbb{F}_{251}$),
- the scheme uses the variant `THRESHOLD_NFPR`,
- it uses the hash function (and XOF) available at `sym/hash/sha3/sha3-several`,
- it uses the pseudo-random bit generator available at `sym/drbg/hash`,
- the scheme can rely on the fourfold implementation of the symmetric primitives when available.

Moreover, the flags `GF251` and `L1` enable us to select the desired parameter sets.

9.2.3. Some benchmarks

We used the library to implement the schemes proposed in Chapter 7. All these schemes rely on the variant "Hypercube 5R". The library has been configured such that the pseudo-randomness is generated using AES in counter mode, the hash function is SHA3, and the extendable output function (used for the MPC challenge) is sampled using SHAKE. We benchmarked our schemes on a 3.8 GHz Intel Core i7 CPU with the support of AVX2 and AES instructions. All the reported timings were measured on this CPU while disabling Intel Turbo Boost.

The obtained signing times are given in Table 9.1. As explained previously, the library can produce detailed benchmarks. It enables the developer to estimate the computational contribution of each part of the implemented signature scheme. In Table 9.1, we decompose the running time of our schemes in six parts: the expansion of the seed trees, the commitments of the input shares, the expansion of the input shares from seeds, the remaining operations to prepare input shares (e.g. the computation of the shares of the "main" parties of the hypercube technique), the emulation of the MPC protocol and the rest of the computation.

Here is an analysis of the obtained running times:

Table 9.1.: Benchmark of the signature schemes proposed in Chapter 7.

Scheme	Detailed Benchmark						High-level Benchmark	
	Tree	Commit.	Rand. Expan.	Share Prep.	MPC Emul.	Misc	Total signing time	Size
Variant "Short" – 256 parties ($N = 256$)								
MQ over $\mathbb{F}_{256}$	1.37	1.42	0.53	0.40	6.25	0.59	10.56	7 114 B
MQ over $\mathbb{F}_{251}$	1.37	1.42	1.24	1.77	2.17	0.59	8.56	7 114 B
MinRank (with RD)	1.06	1.11	1.52	0.51	3.75	0.44	8.39	7 122 B
MinRank (with LP)	0.99	1.05	1.12	0.45	13.23	0.38	17.22	5 518 B
Rank SD (with RD)	1.16	1.22	0.69	0.27	2.36	0.42	6.12	8 543 B
Rank SD (with LP)	1.10	1.14	0.51	0.24	3.72	0.38	7.09	5 899 B
Variant "Fast" – 32 parties ($N = 32$)								
MQ over $\mathbb{F}_{256}$	0.26	0.28	0.10	0.05	6.9	0.24	7.83	8 488 B
MQ over $\mathbb{F}_{251}$	0.26	0.28	0.22	0.23	2.15	0.28	3.42	8 488 B
MinRank (with RD)	0.24	0.27	0.28	0.07	2.68	0.16	3.70	9 288 B
MinRank (with LP)	0.20	0.23	0.21	0.12	13.63	0.15	14.54	7 204 B
Rank SD (with RD)	0.24	0.27	0.13	0.07	2.30	0.18	3.19	11 000 B
Rank SD (with LP)	0.22	0.24	0.09	0.03	3.71	0.12	4.41	7 376 B

- **Tree Expansion**: it consists in deriving N seeds from a master seed using the structure of a binary tree. This operation only depends on the number of parties N, and it is repeated at each repetition (*i.e.* τ times). Thus, when we fix N, the computation contribution is linear into τ. It can be observed from the benchmark: when $N = 32$, it takes $0.0073 \cdot \tau$ ms, and when $N = 256$ it takes $0.055 \cdot \tau$ ms.

- **Commitment**: it consists in committing the input shares of N parties. In practice, it consists in committing a λ-bit seed for all the parties except the last one. The cost of committing the entire input share of the last party tends to be negligible compared to the cost of committing $N - 1$ seeds. The computation contribution of the commitments is thus roughly linear into $N \cdot \tau$. From the benchmark, we get that it takes $0.0575 \cdot \tau$ ms when $N = 256$ and $0.0082 \cdot \tau$ ms when $N = 32$ (committing a seed with a salt takes around 220 nanoseconds).

- **Randomness Expansion**: it consists in expanding seeds to get input shares. The computational cost depends on the number τ of repetitions, the size of the input shares, and the field from which elements should be sampled. When the field is an extension of $\mathbb{F}_2$, the sampling can be efficient. However, sampling in another field is less efficient since we need to deal with rejection. This explains why the cost of this step is larger for MQ over $\mathbb{F}_{251}$ than for MQ over $\mathbb{F}_{256}$.

- **Share Preparation**: it consists in getting the input share of the last party from the other ones and computing the shares of the "main" parties of the hypercube technique (see [AGH+23] for details). It depends on τ, the size of the input shares, and the additive law of the underlying field. This step is very efficient when working in characteristic

two since the addition is the bitwise XOR. When working in prime fields, we need to deal with reduction.

- **MPC Emulation**: it consists in emulating the MPC protocols. Thanks to the hypercube technique, it consists in emulating $1 + \log_2(N)$ parties by repetition. The important point to remark here is that the choice of N does not much impact the emulation cost. It comes from the fact that $\tau \approx \frac{\lambda}{\log_2(N)}$, so the total computation cost of the emulation corresponds[2] to the cost of emulating $\tau \cdot (1 + \log_2(N)) \approx \lambda + \frac{\lambda}{\log_2(N)}$ parties.

- **Misc**: it corresponds to the rest of the signing computation (decompression of the public key, building of the signature, ...).

We can observe that the running time due to the symmetric primitives (the three first columns) is not negligible compared to the emulation of the MPC protocol. For 256 parties, it takes at least 2.5 milliseconds. Let us stress that the factorized code of the library, which mainly relies on symmetric primitives, has been optimized. For example, it relies on fourfold calls of Keccak (for SHA3) using AVX instructions. However, the arithmetic parts used by the MPC protocols have *not been optimized*. When the MPC protocol is heavy (as for the scheme based on the MinRank problem with linearized polynomials), this is the computational bottleneck. However, for light MPC protocols (for example, those based on the rank syndrome decoding problem), the timing contribution of the MPC emulation is similar to the one of symmetric primitives. We could expect that the latter becomes the computational bottleneck when using an optimized implementation of the MPC protocols. In fact, that can be observed in the implementations of the NIST candidate SD-in-the-Head (see Section 9.1.1).

[2] We omit here that τ is larger than $\frac{\lambda}{\log_2(N)}$ to be secure against the forgery attack of [KZ20a], but the conclusion would be the same.

Chapter 10.

Conclusion and Open Questions

In this thesis, we have investigated the MPC-in-the-Head paradigm to build post-quantum signature schemes. Prior works only applied this paradigm to symmetric primitives (such as LowMC or AES). We demonstrated that it can also be applied to other hardness assumptions, e.g. from code-based cryptography. Most of the proposed schemes achieve sizes between 5 KB and 10 KB while relying on unstructured assumptions. They are thus competitive with the existing schemes in the post-quantum literature. They outperform SPHINCS$^+$ by achieving smaller sizes with faster signing algorithms, while staying conservative (even though their security assumptions do not only rely on hash functions as in SPHINCS$^+$). The MPC-in-the-Head framework currently leads to the best code-based schemes for the common "signature size + public key size" metric. These good sizes have been achieved by designing very efficient MPC protocols for the considered problems. An open question would be:

Question 10.1. *Do there exist MPC protocols which would lead to better schemes for those hardness assumptions (while working in the broadcast-based multiparty computation model with additive sharing)?*

However, we can already conjecture that we should not observe big improvements. We can indeed exhibit lower bounds for the size of MPCitH-based signatures with additive sharing (using the fact that one uses GGM trees and one needs to share the secret). For example, for the rank syndrome decoding problem, we can show that the signature size is at least (in bits)

$$4\lambda + \tau \cdot (k \cdot m \cdot \log_2 q + \lambda \cdot \log_2 N + 2\lambda)$$

where λ is the security level in bits, N is the number of parties involved in the underlying MPC protocol, τ is the number of repetitions and (q, m, k) are RSD parameters. For the RSD instance used in Section 7.4.3, we get a lower bound of 3.6 KB for 128-bit security with $N = 256$, while our scheme achieves sizes of 5.9 KB.

The most promising direction to improve the schemes proposed in this thesis consists in proposing new MPCitH transformations. In Chapter 8, we showed that we can use low-threshold linear secret sharings to build faster schemes. The approach leads to the MPCitH-based schemes with the fastest verification. However, it suffers from an increase of the signature size, which can be important depending on the used MPC protocol. In the state of the art, there are only two MPCitH works dealing with Shamir's secret sharings: [AHIV17] and the work described in Chapter 8. We can observe that there is a gap between them. [AHIV17] considers a more general setting, but it does not achieve competitive performance

when building signature schemes, and Chapter 8 considers a restricted setting (it allows only linear operations on sharings) preventing to use more efficient MPC protocols to build signatures. A natural question would thus be:

Question 10.2. *Could we improve [AHIV17] to get small sizes even when considering small arithmetic circuits (as those in signature schemes)? Could we extend the setting of Chapter 8 while keeping small sizes?*

Finally, as mentioned in Chapter 3, the state of the art about schemes with advanced functionalities (as ring/group signatures, multi-signatures, threshold signatures, ...) are sparse. Now that we have efficient signature schemes, an interesting research direction would consist in adding features to them.

Question 10.3. *How can we build efficient schemes with advanced functionalities thanks to the MPC-in-the-Head paradigm?*

Appendices

A. Proof of Privacy

This appendix provides a proof for the following theorem:

Theorem 8.3.4. *Let us consider an MPC protocol Π_{add} complying with the protocol format described in Protocol 10. If Π_{add} is well-defined and $(N-1)$-private, then the protocol Π_{LSSS} corresponding to Π_{add} with an $(\ell+1, N)$-threshold linear secret sharing scheme (see Protocol 13) is well-defined and ℓ-private.*

Proof. In the MPC protocol Π_{LSSS} using an $(\ell+1, N)$-threshold LSSS, all the values computed by the parties are shares from the underlying LSSS. The parties take such sharings as input and the latters are stable by the operations performed in Π_{LSSS} (since the computation over the shares is linear). Therefore, the MPC protocol Π_{LSSS} is well-defined.

By assumption, we have that the MPC protocol Π_{add} is $(N-1)$-private when using an additive sharing. This implies that there exists a simulator $\mathsf{Sim}_{\text{add}}$ which takes as inputs a set $I' \subset [N]$ of size $N-1$, $[\![w']\!]_{I'}$, $[\![\vec{\beta}']\!]_{I'}$ and the outcome (Accept or Reject) of the real-world execution and which outputs views for parties in I' whose joint distribution is perfectly indistinguishable from the joint views of the same parties in a real-world execution of the MPC protocol (with the same outcome).

We first describe the simulator $\mathsf{Sim}_{\text{LSSS}}(I, [\![w]\!]_I, [\![\vec{\beta}]\!]_I, \vec{\varepsilon}, y)$:

1. Sample w' and $\vec{\beta}'$ randomly.
2. Compute $[\![w']\!] \leftarrow \mathsf{Share}_{\text{add}}(w'; r_{w'})$ for some fresh randomness $r_{w'}$.
3. Compute $[\![\vec{\beta}']\!] \leftarrow \mathsf{Share}_{\text{add}}(\vec{\beta}'; r_{\vec{\beta}'})$ for some fresh randomness $r_{\vec{\beta}'}$.
4. Choose a set $I' \subset [N]$ such that $|I'| = N-1$.
5. Call the simulator $\mathsf{view}_{\text{add}} \leftarrow \mathsf{Sim}_{\text{add}}(I', [\![w']\!]_{I'}, [\![\vec{\beta}']\!]_{I'}, \vec{\varepsilon}, y)$.
6. Extract $\vec{\alpha}'$ from $\mathsf{view}_{\text{add}}$ and let $\vec{\alpha} := \vec{\alpha}'$.
7. For $j = 1$ to t:
   - Compute $[\![\alpha^j]\!]_I = \varphi^j_{(\varepsilon^i)_{i\leq j},(\alpha^i)_{i<j}}([\![w]\!]_I, ([\![\beta^i]\!]_I)_{i\leq j})$,
   - Deduce $[\![\alpha^j]\!]$ from $[\![\alpha^j]\!]_I$ and α^j.
8. Output $[\![\vec{\alpha}]\!]$, $[\![w]\!]_I$, $[\![\vec{\beta}]\!]_I$.

In what follows, we show that the above simulator outputs a distribution which is perfectly indistinguishable from the views of the same parties in a real execution setting, which proves the ℓ-privacy of the protocol Π_{LSSS}.

Let us fix a set $I \subset [N]$ such that $|I| = \ell$. Let us also take w and $\vec{\beta}$ and share them as

$$[\![w]\!] = \mathsf{Share}_{\text{LSSS}}(w; r_1)$$
$$[\![\vec{\beta}]\!] = \mathsf{Share}_{\text{LSSS}}(\vec{\beta}; r_2)$$

using some random tapes r_1, r_2. Finally, let us sample a random $\vec{\varepsilon}$.

We propose below three experiments. We denote Exp_1, Exp_2 and Exp_3 the distribution of their respective outputs. The first experiment corresponds to the simulation of the joint view of the parties in I.

Experiment 1. We compute the output $y \in \{\textsc{Accept}, \textsc{Reject}\}$ of the protocol using $[\![w]\!]$, $[\![\vec{\beta}]\!]$ and $\vec{\varepsilon}$:

$$y = \begin{cases} \textsc{Accept} & \text{if } g(\vec{\alpha}) = 0, \\ \textsc{Reject} & \text{otherwise,} \end{cases} \qquad \text{with} \quad \vec{\alpha} = \Phi(w, \vec{\varepsilon}, \vec{\beta}) \ .$$

Then, we output the views returned by $\mathsf{Sim}_{\text{LSSS}}(I, [\![w]\!]_I, [\![\vec{\beta}]\!]_I, \vec{\varepsilon}, y)$ together with y.

Experiment 2. The difference with the previous experiment is that w' and $\vec{\beta}'$ are respectively defined as w and $\vec{\beta}$ (instead of being random). Thanks to the privacy of the additive sharing we get that $[\![w']\!]_I$ and $[\![\vec{\beta}']\!]_I$ are perfectly indistinguishable from $[\![w]\!]_I$ and $[\![\vec{\beta}]\!]_I$ which implies

$$\text{Exp}_1 \equiv \text{Exp}_2 \ .$$

(Here $\text{Exp}_1 \equiv \text{Exp}_2$ means that the two distributions are perfectly indistinguishable).

Experiment 3. The difference with the previous experiment is that $\vec{\alpha}$ is defined as $\vec{\alpha} = \Phi(w, \vec{\varepsilon}, \vec{\beta})$ (where Φ is defined as in Equation (8.1)) instead of $\vec{\alpha} = \vec{\alpha}'$. Thanks to the $(N-1)$-privacy of the protocol Π_{add}, we get that the joint views $\mathsf{view}_{\text{add}}$ returned by the simulator are perfectly indistinguishable from the same views in real-world protocol execution. In particular, $[\![\vec{\alpha}']\!]$ is identically distributed in the two experiments, which implies

$$\text{Exp}_2 \equiv \text{Exp}_3 \ .$$

Finally, it is not hard to check that the output of Experiment 3 corresponds to the real-world joint views of the parties in I, which concludes the proof. □

B. Proof of Soundness

This appendix provides a proof for the soundness property in the following theorem:

Theorem 8.3.5. *Let us consider an MPC protocol Π_{LSSS} complying with the protocol format described in Protocol 13 using an $(\ell+1, N)$-threshold LSSS, such that Π_{LSSS} is ℓ-private in the semi-honest model and of false positive rate p. Then, Protocol 14 built from Π_{LSSS} satisfies the three following properties:*

- ***Completeness.*** *A prover $\mathcal{P}$ who knows a witness w such that $(x, w) \in \mathcal{R}$ and who follows the steps of the protocol always succeeds in convincing the verifier $\mathcal{V}$.*

- ***Soundness.*** *Suppose that there is an efficient prover $\tilde{\mathcal{P}}$ that, on input x, convinces the honest verifier $\mathcal{V}$ to accept with probability*

$$\tilde{\varepsilon} := \Pr[\langle \tilde{\mathcal{P}}, \mathcal{V} \rangle(x) \to 1] \ > \ \varepsilon$$

where the soundness error ε is defined as

$$\varepsilon := \frac{1}{\binom{N}{\ell}} + p \cdot \frac{\ell \cdot (N - \ell)}{\ell + 1} \ .$$

Then, there exists an efficient probabilistic extraction algorithm $\mathcal{E}$ that, given rewindable black-box access to $\tilde{\mathcal{P}}$, outputs either a witness w satisfying $(x, w) \in \mathcal{R}$, or a commitment/hash collision, by making an average number of calls to $\tilde{\mathcal{P}}$ which is upper bounded by

$$\frac{4}{\tilde{\varepsilon} - \varepsilon} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{8 \cdot (N - \ell)}{\tilde{\varepsilon} - \varepsilon}\right).$$

- ***Honest-Verifier Zero-Knowledge.*** *There exists an efficient simulator $\mathcal{S}$ which, given the random challenge I outputs a transcript which is indistinguishable from a real transcript of Protocol 14.*

For any set $\mathcal{T}$ of successful transcripts corresponding to the same commitment,

- either the revealed shares of $[\![w]\!]$ are not unique, and then we find a commitment collision or a hash collision,
- or the openings are unique.

Along this proof, we consider that the extractor only gets transcripts with unique revealed shares since otherwise the extractor would find directly a commitment collision or a hash collision.

We shall denote by R_h the randomness of $\tilde{\mathcal{P}}$ which is used to generate the initial commitment h_1 (which determines the witness sharing $[\![w]\!]$), and we denote r_h a possible realization of R_h. Throughout the proof, we denote $\mathsf{succ}_{\tilde{\mathcal{P}}}$ the event that $\tilde{\mathcal{P}}$ succeeds in convincing $\mathcal{V}$. By hypothesis, we have $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}] = \tilde{\varepsilon}$.

B.1. Technical Lemmas

In our proof, we shall make use of the following lemmas:

Lemma B.1 (Splitting Lemma [PS00])**.** *Let X and Y be two finite sets, and let $A \subseteq X \times Y$ such that*

$$\Pr\left[(x, y) \in A \mid (x, y) \leftarrow X \times Y\right] \geq \varepsilon \ .$$

For any $\alpha \in [0, 1)$, let

$$B = \left\{(x, y) \in X \times Y \ \middle| \ \Pr\left[(x, y') \in A \mid y' \leftarrow Y\right] \geq (1 - \alpha) \cdot \varepsilon\right\} \ .$$

We have:

1. $\Pr\left[(x, y) \in B \mid (x, y) \leftarrow X \times Y\right] \geq \alpha \cdot \varepsilon$

2. $\Pr\left[(x,y) \in B \mid (x,y) \leftarrow A\right] \geq \alpha$.

Proof. See [PS00] for the proof. □

Lemma B.2. *Let $E_1, \ldots, E_M$ be arbitrary events. For $i \in \{1, \ldots, M\}$ and a bit $b \in \{0,1\}$, let us define*

$$A_i^b := \begin{cases} E_i & \text{if } b = 1, \\ \bar{E}_i & \text{if } b = 0. \end{cases}$$

We have those following relations, where $\mathrm{wt}_H(x)$ denotes the Hamming weight of x (i.e. the number of non-zero coordinates of x):

$$\sum_{x \in \{0,1\}^M} \Pr[A_1^{x_1}, A_2^{x_2}, \ldots, A_M^{x_M}] = 1$$

$$\sum_{x \in \{0,1\}^M} \mathrm{wt}_H(x) \cdot \Pr[A_1^{x_1}, A_2^{x_2}, \ldots, A_M^{x_M}] = \Pr[E_1] + \ldots + \Pr[E_M]$$

Proof. The first equality comes from the fact the underlying events form a partition. The second equality can be proved by induction on the integer M. The case $M = 1$ is trivial:

$$\sum_{x \in \{0,1\}} \mathrm{wt}_H(x) \cdot \Pr[A_1^{x_1}] = 0 \cdot \Pr[\bar{E}_1] + 1 \cdot \Pr[E_1] = \Pr[E_1]$$

Let assume the relation for a positive integer M. By noting for some $x \in \{0,1\}^M$ $A^{\mathbf{x}} := \{A_1^{x_1}, \ldots, A_M^{x_M}\}$, we have

$$\begin{aligned}
&\sum_{x \in \{0,1\}^{M+1}} \mathrm{wt}_H(x) \cdot \Pr[A_1^{x_1}, A_2^{x_2}, \ldots, A_{M+1}^{x_{M+1}}] \\
&= \sum_{x \in \{0,1\}^M} \mathrm{wt}_H(x) \cdot \Pr[A^{\mathbf{x}}, \bar{E}_{M+1}] \\
&\qquad + \sum_{x \in \{0,1\}^M} (\mathrm{wt}_H(x) + 1) \cdot \Pr[A^{\mathbf{x}}, E_{M+1}] \\
&= \sum_{x \in \{0,1\}^M} \mathrm{wt}_H(x) \cdot (\Pr[A^{\mathbf{x}}, \bar{E}_{M+1}] + \Pr[A^{\mathbf{x}}, E_{M+1}]) \\
&\qquad + \sum_{x \in \{0,1\}^M} \Pr[A^{\mathbf{x}}, E_{M+1}] \\
&= \sum_{x \in \{0,1\}^M} \mathrm{wt}_H(x) \cdot \Pr[A^{\mathbf{x}}] + \Pr[E_{M+1}] \\
&= (\Pr[E_1] + \ldots + \Pr[E_M]) + \Pr[E_{M+1}].
\end{aligned}$$

□

B.2. When restraining to only bad witnesses

Let r_h be a possible realization of R_h. Given $R_h = r_h$, we have a unique hash commitment h_1 in the transcript. This hash commitment uniquely defines the shares of the witness $[\![w]\!]_1$, $\ldots$, $[\![w]\!]_N$ (by assumption on the absence of hash/commitment collisions). In the following, we shall denote $w^{(J)}$ the witness corresponding to the shares $\{[\![w]\!]_i\}_{i \in J}$, for $|J| = \ell + 1$. We

have a total of $\binom{N}{\ell+1}$ possibly-distinct witnesses $w^{(J)}$. We shall say that $w^{(J)}$ is a *good witness* whenever $(x, w^{(J)}) \in \mathcal{R}$, otherwise we call $w^{(J)}$ a *bad witness*.

For any transcript produced by $\tilde{\mathcal{P}}$ (with h_1 as first hash commitment), the hash commitments $h_2, \ldots, h_t$ uniquely define the shares $\{[\![\alpha^j]\!]_i\}_{i\in S}$, for $j \in [t]$. In the following, we shall denote by $[\![\alpha^j]\!] = ([\![\alpha^j]\!]_1, \ldots, [\![\alpha^j]\!]_N)$ the full $(\ell+1, N)$-sharing consistent with the shares $\{[\![\alpha^j]\!]_i\}_{i\in S}$. The hash commitments $h_2, \ldots, h_t$ also uniquely define the shares $\{[\![\beta^j]\!]_i\}_{i\in[N]}$, for $j \in [t]$. We shall denote $\mathcal{H}$ the set of *honest parties*, *i.e.* the set of the parties for which the committed shares $[\![\alpha^1]\!]_i, \ldots, [\![\alpha^t]\!]_i$, are consistent with the committed input shares $[\![w]\!]_i$ and $[\![\beta^1]\!]_i, \ldots, [\![\beta^t]\!]_i$. More formally,

$$\mathcal{H} = \left\{ i : \forall j, [\![\alpha^j]\!]_i = \varphi^j_{(\varepsilon^i)_{i\le j},(\alpha^i)_{i<j}}([\![w]\!]_i, ([\![\beta^k]\!]_i)_{k\le j}) \right\} .$$

We further denote Y the random variable which corresponds to the number of honest parties, *i.e.* $Y = |\mathcal{H}|$. We stress that $[\![\alpha^1]\!], \ldots, [\![\alpha^t]\!]$, $\mathcal{H}$ and Y depend on the randomness of the (malicious) prover and the randomness of the verifier used before Step 3 of Protocol 14.

For every $i \in [N]$ and $j \in [t]$, we shall further denote $[\![\bar{\alpha}^j]\!]_i$ the share obtained through an honest computation from the committed input shares, that is:

$$[\![\bar{\alpha}^j]\!]_i = \varphi^j_{(\varepsilon^i)_{i\le j},(\alpha^i)_{i<j}}([\![w]\!]_i, ([\![\beta^k]\!]_i)_{k\le j}) .$$

We stress that $[\![\bar{\alpha}^j]\!]_i$ might not be equal to $[\![\alpha^j]\!]_i$. We actually have $i \in \mathcal{H}$ if and only if $[\![\bar{\alpha}^j]\!]_i = [\![\alpha^j]\!]_i$ for every $j \in [t]$. In the following, we shall say that witness $w^{(J)}$ gives rise to a false positive in the MPC protocol Π, and denote this probability event E_J, whenever

$$g(\bar{\alpha}^1_J, \ldots, \bar{\alpha}^t_J) = 0$$

where $\bar{\alpha}^1_J, \ldots, \bar{\alpha}^t_J$ are the plain values corresponding to the sharings $\{[\![\bar{\alpha}^1]\!]_i\}_{i\in J}, \ldots, \{[\![\bar{\alpha}^t]\!]_i\}_{i\in J}$. By definition of the MPC protocol Π, we have:

$$\forall J \text{ s.t. } |J| = \ell+1, \ \Pr[E_J] \le p.$$

For the first step of the proof, we shall consider a subset D of parties, *i.e.* $D \subset \{1, \ldots, N\}$, and we denote

$$N' := |D| \qquad \text{and} \qquad \mathcal{J} = \{J \subset D : |J| = \ell+1\}.$$

We shall further denote

$$\Pr'[\,\cdot\,] := \Pr[\ \cdot\ \mid R_h = r_h, I \subset D].$$

We will show that, if $\{w^{(J)}\}_{J\in\mathcal{J}}$ are all *bad* witnesses, then the probability $\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}]$ is upper bounded by

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}] \le \frac{\binom{N}{\ell}}{\binom{N'}{\ell}} \cdot \varepsilon \tag{1}$$

where ε is the soundness error defined in the theorem statement, which is

$$\varepsilon := \frac{1}{\binom{N}{\ell}} + p \cdot \frac{\ell \cdot (N-\ell)}{\ell+1} .$$

The rest of this subsection is devoted to the demonstration of Equation (1).

For $J \in \mathcal{J}$ and $b \in \{0,1\}$, let us introduce the notation

$$A_J^b = \begin{cases} E_J & \text{if } b = 1, \\ \bar{E}_J & \text{if } b = 0. \end{cases}$$

Let $x = (x_J)_{J \in \mathcal{J}} \in \{0,1\}^{|\mathcal{J}|}$ and let $y \in \{0, \ldots, N\}$. Let us assume that $\mathsf{succ}_{\tilde{\mathcal{P}}}$, $Y = y$ and $\{A_J^{x_J}\}_{J \in \mathcal{J}}$ jointly occur. Because $\mathsf{succ}_{\tilde{\mathcal{P}}}$ occurs, we have that

$$g(\alpha^1, \ldots, \alpha^t) = 0$$

where $\alpha^1, \ldots, \alpha^t$ are the values corresponding to the sharings $[\![\alpha^1]\!], \ldots, [\![\alpha^t]\!]$. Then for each set $J \in \mathcal{J}$ ($w^{(J)}$ is a bad witness) such that $J \subset \mathcal{H}$ (the parties in J are honest), we have that $[\![\bar{\alpha}^j]\!]_i = [\![\alpha^j]\!]_i$ for every $i \in J$ and every $j \in [t]$, which implies

$$g(\bar{\alpha}_J^1, \ldots, \bar{\alpha}_J^t) = g(\alpha^1, \ldots, \alpha^t) = 0 \ .$$

Namely, a false positive necessarily occurs for $w^{(J)}$, *i.e.* $x_J = 1$, whenever $J \in \mathcal{J}$ with $J \subset \mathcal{H}$. Thus

$$\mathrm{wt}_H(x) \geq \sum_{J \in \mathcal{J} : J \subset \mathcal{H}} x_J = \binom{y}{\ell + 1} \ .$$

By defining

$$y_{\max} := \max\{y : \mathrm{wt}_H(x) \geq \binom{y}{\ell + 1}\} \ ,$$

we get that

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}, Y = y \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] = 0 \qquad \text{if } y > y_{\max}$$

and so

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] = \sum_{y=0}^{y_{\max}} \Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}, Y = y \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \ .$$

The only way for the transcript to be successful is that the set I of challenged opened parties only contains honest parties, *i.e.* $I \subset \mathcal{H}$. Thus,

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}, Y = y] \leq \Pr[I \subset \mathcal{H} \mid I \subset D, Y = y] = \frac{\binom{y}{\ell}}{\binom{N'}{\ell}} \ .$$

We deduce

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \ \leq \ \sum_{y=0}^{y_{\max}} \frac{\binom{y}{\ell}}{\binom{N'}{\ell}} \cdot \Pr'[Y = y \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \ \leq \ \frac{\binom{y_{\max}}{\ell}}{\binom{N'}{\ell}} \ .$$

Since $\mathrm{wt}(x)$ is a non-negative integer, we have $y_{\max} \geq \ell$. Let us consider three cases:

Case 1: $y_{\max} = \ell$, it means that $\mathrm{wt}(x) = 0$, then

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \leq \frac{\binom{y_{\max}}{\ell}}{\binom{N'}{\ell}} = \frac{1}{\binom{N'}{\ell}} = \frac{\mathrm{wt}_H(x) \cdot \ell + 1}{\binom{N'}{\ell}}$$

Case 2: $y_{\max} = \ell + 1$, it means that $\mathrm{wt}(x) \geq 1$, then

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \leq \frac{\binom{y_{\max}}{\ell}}{\binom{N'}{\ell}} = \frac{\ell+1}{\binom{N'}{\ell}} \leq \frac{\mathrm{wt}_H(x) \cdot \ell + 1}{\binom{N'}{\ell}}$$

Case 3: $y_{\max} \geq \ell + 2$, then

$$\begin{aligned}\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] &\leq \frac{\binom{y_{\max}}{\ell}}{\binom{N'}{\ell}} = \frac{\frac{\ell+1}{y_{\max}-\ell}\binom{y_{\max}}{\ell+1}}{\binom{N'}{\ell}} \\ &\leq \frac{\ell+1}{2} \cdot \frac{\mathrm{wt}_H(x)}{\binom{N'}{\ell}} \leq \frac{\mathrm{wt}_H(x) \cdot \ell + 1}{\binom{N'}{\ell}}\end{aligned}$$

In any case, we have the relation

$$\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \leq \frac{\mathrm{wt}_H(x) \cdot \ell + 1}{\binom{N'}{\ell}}. \tag{2}$$

for any $x \in \{0,1\}^{|\mathcal{J}|}$. And so, we have

$$\begin{aligned}\Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}}] &= \sum_{x \in \{0,1\}^{\mathcal{J}}} \Pr'[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid \{A_J^{x_J}\}_{J \in \mathcal{J}}] \cdot \Pr'[\{A_J^{x_J}\}_{J \in \mathcal{J}}] \\ &\leq \sum_{x \in \{0,1\}^{\mathcal{J}}} \frac{\mathrm{wt}_H(x) \cdot \ell + 1}{\binom{N'}{\ell}} \cdot \Pr'[\{A_J^{x_J}\}_{J \in \mathcal{J}}] && \text{using (2)} \\ &= \frac{1}{\binom{N'}{\ell}} + \frac{\ell}{\binom{N'}{\ell}} \cdot \sum_{x \in \{0,1\}^{\mathcal{J}}} \mathrm{wt}_H(x) \cdot \Pr'[\{A_J^{x_J}\}_{J \in \mathcal{J}}] && \text{using Lemma } B.2 \\ &\leq \frac{1}{\binom{N'}{\ell}} + \frac{\ell}{\binom{N'}{\ell}} \cdot \sum_{J \in J} \Pr'[E_J] && \text{using Lemma } B.2 \\ &\leq \frac{1}{\binom{N'}{\ell}} + \frac{\ell}{\binom{N'}{\ell}} \cdot |\mathcal{J}| \cdot p \\ &= \frac{\binom{N}{\ell}}{\binom{N'}{\ell}} \cdot \left(\frac{1}{\binom{N}{\ell}} + \frac{\ell}{\binom{N}{\ell}} \cdot \underbrace{\binom{N'}{\ell+1}}_{\leq \binom{N}{\ell+1}} p \right) \leq \frac{\binom{N}{\ell}}{\binom{N'}{\ell}} \cdot \varepsilon \ .\end{aligned}$$

Thus we obtain the desired inequality (1).

B.3. Building of the extractor

In the previous subsection, we proved that the probability that a malicious prover $\tilde{\mathcal{P}}$ produces a valid transcript when the opened parties are restricted to a set of N' parties for which the shares only encode bad witnesses is upper-bound by

$$\frac{\binom{N}{\ell}}{\binom{N'}{\ell}} \cdot \varepsilon \ .$$

We now show how to build an extractor which outputs a witness w satisfying $(x, w) \in \mathcal{R}$ (if not a hash or commitment collision) when giving rewindable black-box access to a malicious prover $\tilde{\mathcal{P}}$ which produces successful transcripts with a probability $\tilde{\varepsilon} > \varepsilon$.

Let us fix an arbitrary value $\delta \in (0, 1)$ such that $(1 - \delta)\tilde{\varepsilon} > \varepsilon$ (such δ exists since $\tilde{\varepsilon} > \varepsilon$). Let r_h be a possible realization of R_h. We will say that r_h is *good* if it is such that

$$\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h = r_h] \geq (1 - \delta) \cdot \tilde{\varepsilon} \ . \tag{3}$$

By the Splitting Lemma B.1, we have

$$\Pr[R_h \text{ good} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}] \geq \delta \ . \tag{4}$$

Our extractor first runs the $\tilde{\mathcal{P}}$ with honest verifier requests until obtaining a successful transcript T_0 by running. If this T_0 corresponds to a good r_h, then we can obtain further successful transcripts with "high" probability (*i.e.* probability greater than $(1 - \delta) \cdot \tilde{\varepsilon}$) by rewinding the protocol just after the initial commitment h_1. Based on the assumption that r_h is good, a sub-extractor $\mathcal{E}_0$ will build a list of *successful* transcripts $\mathcal{T}$, all with same initial commitment. We denote $P(\mathcal{T})$ the set of the parties which have been open in at least one transcript of $\mathcal{T}$, *i.e.* $P(\mathcal{T}) := \bigcup_{T \in \mathcal{T}} I_T$ where I_T is the set of opened parties of the transcript T.

For a certain number N_1 of iterations, the sub-extractor $\mathcal{E}_0$ tries to feed the list $\mathcal{T}$ until there exist a good witness among the open input shares. We formally describe the sub-extractor routine in the following pseudocode:

Sub-extractor $\mathcal{E}_0$ (on input a successful transcript T_0):

1. $\mathcal{T} = \{T_0\}$
2. Do N_1 times:
3. Run $\tilde{\mathcal{P}}$ with honest $\mathcal{V}$ and same r_h as T_0 to get transcript T
4. If T is a successful transcript,
5. $\mathcal{T} \leftarrow \mathcal{T} \cup \{T\}$.
6. If $\mathcal{T}$ contains a good witness w, return w.
7. Return $\emptyset$.

Let us evaluate the probability that the stop condition is reached in a given number of iteration N_1. Consider a loop iteration in $\mathcal{E}_0$ at the beginning of which we have a list $\mathcal{T}$ of successful transcripts (which does not contain a good witness since the stop condition has not been reached) and a transcript T sampled at Step 3. We denote Z the event that a new party is open (a party which is not in $P(\mathcal{T})$) in the transcript T. This event is defined with respect to the randomness of the verifier challenges in T.

Let us lower bound the probability to have a successful transcript T and the event Z occurring in the presence of a good R_h:

$$p_g := \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \cap Z \mid R_h \text{ good}] \ .$$

We have:

$$\begin{aligned} p_g &= \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h \text{ good}] - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \cap \bar{Z} \mid R_h \text{ good}] \\ &= \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h \text{ good}] - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h \text{ good}, \bar{Z}] \cdot \Pr[\bar{Z} \mid R_h \text{ good}] \\ &\geq (1 - \delta) \cdot \tilde{\varepsilon} - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h \text{ good}, \bar{Z}] \cdot \Pr[\bar{Z} \mid R_h \text{ good}] \end{aligned}$$

where the last inequality holds by (3).

The probability that a new party is not opened corresponds to the probability that the set I of opened parties is a subset of $P(\mathcal{T})$, *i.e.*

$$\Pr[\bar{Z} \mid R_h \text{ good}] = \Pr[\bar{Z}] = \frac{\binom{|P(\mathcal{T})|}{\ell}}{\binom{N}{\ell}} \ .$$

The success probability knowing that that no new party is open corresponds to the success probability when restricting to the $|P(\mathcal{T})|$ parties which have been already open. By assumption ($\mathcal{T}$ does not contain a good witness), the shares of those parties only correspond to bad witnesses. Thus, this probability can be upper bounded using the inequality (1) of Section B.2 with $N' = |P(\mathcal{T})|$:

$$\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}} \mid R_h \text{ good}, \bar{Z}] \leq \frac{\binom{N}{\ell}}{\binom{|P(\mathcal{T})|}{\ell}} \cdot \varepsilon \ .$$

Thus, we get

$$p_g \geq (1-\delta) \cdot \tilde{\varepsilon} - \varepsilon \ .$$

To summarize, in the presence of a good R_h, the probability of the event $\mathsf{succ}_{\tilde{\mathcal{P}}} \cap Z$ (*i.e.* getting a successful transcript T which opens a new party) is lower bounded by $(1-\delta)\cdot\tilde{\varepsilon}-\varepsilon > 0$. Moreover, the event $\mathsf{succ}_{\tilde{\mathcal{P}}} \cap Z$ can occur at most $N-\ell$ times, because T_0 already opens ℓ parties and there are N parties in total. We deduce that after $N-\ell$ occurrences of $\mathsf{succ}_{\tilde{\mathcal{P}}} \cap Z$, the list $\mathcal{T}$ contains a good witness.

Let us now define

$$N_1 = \frac{4(N-\ell)}{p_0} \quad \text{with} \quad p_0 := (1-\delta) \cdot \tilde{\varepsilon} - \varepsilon \ . \tag{5}$$

And let $X \sim \mathcal{B}(N_1, p_0)$ a binomial distributed random variable with parameters (N_1, p_0). The probability that $\mathcal{E}_0$ reaches the stop condition and returns a (good) witness for a successful transcript T_0 with good R_h satisfies:

$$\begin{aligned}
\Pr[\mathcal{E}_0(T_0) \neq \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0} \cap R_h \text{ good}] &\geq \Pr[X > N-\ell] \\
&= \Pr\left[\frac{X}{N_1} - p_0 > \frac{N-\ell}{N_1} - p_0\right] \\
&= 1 - \Pr\left[\frac{X}{N_1} - p_0 \leq \frac{N-\ell}{N_1} - p_0\right] \\
&= 1 - \Pr\left[\frac{X}{N_1} - p_0 \leq -\frac{3}{4}p_0\right] \\
&\geq 1 - \Pr\left[|\frac{X}{N_1} - p_0| \geq \frac{3}{4}p_0\right] \\
&\geq 1 - \frac{p_0 \cdot (1-p_0)}{N_1 \cdot p_0^2 \cdot \left(\frac{3}{4}\right)^2} \qquad (6) \\
&= 1 - \frac{16}{9} \cdot \frac{1-p_0}{4 \cdot (N-\ell)} = 1 - \frac{4}{9} \cdot \frac{1-p_0}{N-\ell} \\
&\geq 1 - \frac{4}{9} \geq \frac{1}{2}
\end{aligned}$$

The inequality (6) holds from the Bienaymé-Techbychev inequality. Thus, using $N_1 = \frac{4(N-\ell)}{p_0}$, the probability to reach the stop condition assuming a good R_h is at least 1/2. Without assumption on R_h, the probability to reach the stop condition satisfies:

$$\Pr[\mathcal{E}_0(T_0) \neq \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}] \geq \Pr[R_h \text{ good} \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}] \cdot \Pr[\mathcal{E}_0(T_0) \neq \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0} \cap R_h \text{ good}] \geq \frac{\delta}{2} \ .$$

Let us now describe the complete extractor procedure:

<u>Extractor $\mathcal{E}$</u>:

1. Repeat $+\infty$ times:
2. Run $\tilde{\mathcal{P}}$ with honest $\mathcal{V}$ to get transcript T_0
3. If T_0 is not a successful transcript, go to the next iteration
4. Call $\mathcal{E}_0$ on T_0 to get list of transcripts $\mathcal{T}$
5. If $\mathcal{T} \neq \emptyset$, return $\mathcal{T}$

Let C denotes the number of calls to $\tilde{\mathcal{P}}$ made by the extractor before ending. While entering a new iteration:

- the extractor makes one call to $\tilde{\mathcal{P}}$ to obtain T_0,
- if T_0 is not successful, which occurs with probability $(1 - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}])$,
  - the extractor continues to the next iteration and makes an average of $\mathbb{E}[C]$ calls to $\tilde{\mathcal{P}}$,
- if T_0 is successful, which occurs with probability $\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}]$,
  - the extractor makes at most N_1 calls to $\tilde{\mathcal{P}}$ in the loop of $\mathcal{E}_0$,
  - then $\mathcal{E}_0$ returns an empty list (the stop condition is not reached), which occurs with probability $\Pr[\mathcal{E}_0(T_0) = \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}]$, the extractor continues to the next iteration and makes an average of $\mathbb{E}[C]$ calls to $\tilde{\mathcal{P}}$,
  - otherwise, if $\mathcal{E}_0(T_0)$ returns a non-empty list, the extractor stops and no more calls to $\tilde{\mathcal{P}}$ are necessary.

The mean number of calls to $\tilde{\mathcal{P}}$ hence satisfies the following equality:

$$\mathbb{E}[C] = 1 + \underbrace{(1 - \Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}]) \cdot \mathbb{E}[C]}_{T_0 \text{ unsuccessful}} + \underbrace{\Pr[\mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}] \cdot (N_1 + \Pr[\mathcal{E}_0(T_0) = \emptyset \mid \mathsf{succ}_{\tilde{\mathcal{P}}}^{T_0}] \cdot \mathbb{E}[C])}_{T_0 \text{ successful}}$$

which gives

$$\begin{aligned}
\mathbb{E}[C] &\leq 1 + (1 - \tilde{\varepsilon}) \cdot \mathbb{E}[C] + \tilde{\varepsilon} \cdot \Big(N_1 + (1 - \frac{\delta}{2}) \cdot \mathbb{E}[C]\Big) \\
&\leq 1 + \tilde{\varepsilon} \cdot N_1 + \mathbb{E}[C]\Big(1 - \frac{\tilde{\varepsilon} \cdot \delta}{2}\Big) \\
&\leq \frac{2}{\delta \cdot \tilde{\varepsilon}} \cdot (1 + \tilde{\varepsilon} \cdot N_1) \\
&= \frac{2}{\delta \cdot \tilde{\varepsilon}} \cdot \left(1 + \tilde{\varepsilon} \cdot \frac{4 \cdot (N - \ell)}{(1 - \delta) \cdot \tilde{\varepsilon} - \varepsilon}\right)
\end{aligned}$$

To obtain an δ-free formula, let us take δ such that $(1-\delta)\cdot\tilde{\varepsilon} = \frac{1}{2}(\tilde{\varepsilon}+\varepsilon)$. We have $\delta = \frac{1}{2}\left(1-\frac{\varepsilon}{\tilde{\varepsilon}}\right)$ and the average number of calls to $\tilde{\mathcal{P}}$ is upper bounded as

$$\mathbb{E}[C] \leq \frac{4}{\tilde{\varepsilon}-\varepsilon}\cdot\left(1+\tilde{\varepsilon}\cdot\frac{8\cdot(N-\ell)}{\tilde{\varepsilon}-\varepsilon}\right)$$

which concludes the proof.

C. Signature Scheme and Proof of Unforgeability

We can transform the zero-knowledge proofs of knowledge described in Section 8.3 into signature schemes using the Fiat-Shamir heuristic [FS87]. Protocols 15 and 16 describe the signing and verification algorithms obtained when following this approach for the 5-round case (*i.e.* for $t=1$ iteration in the MPC protocol).

When applying the Fiat-Shamir transform, we compute the verifier challenges $(\varepsilon^{[e]})_{e\in[1:\tau]}$ and $(I^{[e]})_{e\in[1:\tau]}$ as:

$$\begin{aligned} &h_1 = \mathrm{Hash}_1(m, \mathsf{salt}, \tilde{h}^{[1]}, \ldots, \tilde{h}^{[\tau]}) \\ &(\varepsilon^{[e]})_{e\in[1:\tau]} \leftarrow \mathrm{Expand}(h_1) \end{aligned}$$

and

$$\begin{aligned} &h_2 = \mathrm{Hash}_2(m, \mathsf{salt}, h_1, [\![\alpha^{[1]}]\!]_S, \ldots, [\![\alpha^{[N]}]\!]_S) \\ &(I^{[e]})_{e\in[1:\tau]} \leftarrow \mathrm{Expand}(h_2) \end{aligned}$$

where m is the input message, Hash_1 and Hash_2 are cryptographic hash functions, Expand is an extendable output hash function, and $(\tilde{h}^{[e]}, [\![\alpha^{[e]}]\!]_S)_{e\in[1:\tau]}$ are the commitments and the broadcast shares merged for the τ repetitions. We introduce a value salt called *salt* which is sampled from $\{0,1\}^{2\lambda}$ at the beginning of the signing process. This value is then used for each commitment to the parties' states. Since the signature security relies on the random oracle model, we can safely replace the commitment scheme Com of Protocol 14 by a single hash function Hash_0. Moreover, we derive all the randomness used in the scheme from a root seed for performance reason and to make the scheme easily turnable into a deterministic signature scheme. In particular, the randomness used for the sharings is derived from this root seed using a pseudo-random generator PRG which is made explicit in the description (while it was implicit in the description of the zero knowledge protocol). Finally, we denote Hash_m the hash function involved for the Merkle trees.

In this signature scheme, a secret key is a witness w and a public key is a statement x, with $(x,w)\in\mathcal{R}$ for the considered relation $\mathcal{R}$. We assume the existence of a function F which maps every witness to the corresponding statement:

$$F : w \mapsto x \quad \text{s.t. } (x,w)\in\mathcal{R}.$$

We further assume that F is an $(t_{\mathrm{OWF}}, \epsilon_{\mathrm{OWF}})$-*hard one-way function*, namely an adversary $\mathcal{A}$ receiving a random statement x has a probability at most ϵ_{OWF} to output the corresponding witness w in time at most t_{OWF}.

The zero-knowledge protocol for relation $\mathcal{R}$ which we transform into the present signature scheme depends on two functions: ψ which computes the hints and φ which corresponds to the party computation. In practice, those two functions depend on the statement x (*i.e.*, on the public key in the case of a signature scheme), so for the sake of completeness, we will index them with x in this section: ψ_x and φ_x.

Inputs: A secret key w, a public key $x := F(w)$ and a message $m \in \{0,1\}^*$.

Phase 0: Initialization.

1. Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$.
2. Sample a root seed $\rho \leftarrow \{0,1\}^{2\lambda}$.

Phase 1: Preparation of the MPC-in-the-Head inputs. For each iteration $e \in [1:\tau]$,

1. Derive randomness $r_w^{[e]}$, $r_\beta^{[e]}$ and $r_\psi^{[e]}$ from the root seed ρ:
$$r_w^{[e]}, r_\beta^{[e]}, r_\psi^{[e]} \leftarrow \mathrm{PRG}(\mathsf{salt}, e, \rho).$$
2. Share the witness w into an $(\ell+1, N)$-threshold linear secret sharing $[\![w^{[e]}]\!]$:
$$[\![w^{[e]}]\!] \leftarrow \mathsf{Share}(w; r_w^{[e]}).$$
3. Compute
$$\beta^{[e]} \leftarrow \psi_x(w; r_\psi^{[e]})$$
and share it:
$$[\![\beta^{[e]}]\!] \leftarrow \mathsf{Share}(\beta^{[e]}; r_\beta^{[e]}).$$
4. Compute the commitments
$$\mathsf{com}_i^{[e]} := \mathrm{Hash}_0(\mathsf{salt}, e, i, [\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)$$
for all $i \in [1:N]$, and compute the Merkle root
$$\tilde{h}^{[e]} := \mathrm{MerkleTree}(\mathsf{com}_1^{[e]}, \ldots, \mathsf{com}_N^{[e]}).$$

Phase 2: First challenge (randomness for the MPC protocol).

1. Compute $h_1 = \mathrm{Hash}_1(m, \mathsf{salt}, \tilde{h}^{[1]}, \ldots, \tilde{h}^{[\tau]})$.
2. Expand h_1 as $(\varepsilon^{[e]})_{e\in[1:\tau]} \leftarrow \mathrm{Expand}(h_1)$.

Phase 3: Simulation of the MPC protocol. For each iteration $e \in [1:\tau]$,

1. Computes, for $i \in S$,
$$[\![\alpha^{[e]}]\!]_i := \varphi_{x,\varepsilon^{[e]}}([\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)$$
and recomposes $\alpha^{[e]}$.

   This step is repeated as many times as in the MPC protocol (cf Protocol 11).

Phase 4: Second challenge (parties to be opened).

1. Compute $h_2 = \mathrm{Hash}_2(\mathsf{salt}, h_1, [\![\alpha^{[1]}]\!]_S, \ldots, [\![\alpha^{[N]}]\!]_S)$.
2. Expand h_2 as $(I^{[e]})_{e\in[1:\tau]} \leftarrow \mathrm{Expand}(h_2)$ where, for every e, $I^{[e]} \subset [1:N]$ is a subset of ℓ parties (*i.e.* $|I^{[e]}| = \ell$).

Phase 5: Building of the signature. Output the signature σ built as
$$\mathsf{salt} \mid h_1 \mid h_2 \mid \left(([\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)_{i\in I}, \mathsf{auth}^{[e]}, [\![\alpha^{[e]}]\!]_{i^{*[e]}} \right)_{e\in[1:\tau]}$$
where $\mathsf{auth}^{[e]}$ is the authentication path for $\{\mathsf{com}_i^{[e]}\}_{i\in I}$ w.r.t. Merkle root $\tilde{h}^{[e]}$ and $i^{*[e]} \in S \setminus I^{[e]}$.

Protocol 15: Signature Scheme – Signing algorithm

Inputs: A public key x, a signature σ and a message $m \in \{0,1\}^*$.

1. Parse the signature σ as

$$\mathsf{salt} \mid h_1 \mid h_2 \mid \Big((\llbracket w^{[e]} \rrbracket_i, \llbracket \beta^{[e]} \rrbracket_i)_{i \in I}, \mathsf{auth}^{[e]}, \llbracket \alpha^{[e]} \rrbracket_{i^{*[e]}} \Big)_{e \in [1:\tau]} \leftarrow \sigma$$

2. Expand h_1 as $(\varepsilon^{[e]})_{e \in [1:\tau]} \leftarrow \mathrm{Expand}(h_1)$.
3. Expand h_2 as $(I^{[e]})_{e \in [1:\tau]} \leftarrow \mathrm{Expand}(h_2)$ where, for every e, $I^{[e]} \subset [1:N]$ is a subset of ℓ parties (*i.e.* $|I^{[e]}| = \ell$).
4. For each iteration $e \in [1:\tau]$,
   - Computes the commitments $\mathsf{com}_i^{[e]}$ and the broadcast values $\llbracket \alpha^{[e]} \rrbracket_i$ for $i \in I^{[e]}$ from $(\llbracket w^{[e]} \rrbracket_i, \llbracket \beta^{[e]} \rrbracket_i)_{i \in I}$: for all $i \in I^{[e]}$,

$$\mathsf{com}_i^{[e]} = \mathrm{Hash}_0(\mathsf{salt}, e, i, \llbracket w^{[e]} \rrbracket_i, \llbracket \beta^{[e]} \rrbracket_i)$$
$$\llbracket \alpha^{[e]} \rrbracket_i = \varphi_{x,\varepsilon^{[e]}}(\llbracket w^{[e]} \rrbracket_i, \llbracket \beta^{[e]} \rrbracket_i)$$

   - Recover $\alpha^{[e]}$, by

$$\alpha^{[e]} = \mathsf{Reconstruct}_{I^{[e]} \cup \{i^{*[e]}\}}(\llbracket \alpha^{[e]} \rrbracket_{I \cup \{i^{*[e]}\}}).$$

   - Compute the Merkle root $\tilde{h}^{[e]}$ using $(\{\mathsf{com}_i^{[e]}\}_{i \in I}, \mathsf{auth}^{[e]})$.
   - Compute the shares $\llbracket \alpha^{[e]} \rrbracket_S$ using $\llbracket \alpha^{[e]} \rrbracket = \mathsf{Expand}_{I \cup \{i^*\}}(\llbracket \alpha^{[e]} \rrbracket_{I \cup \{i^*\}})$;
5. Compute $h_1' = \mathrm{Hash}_1(m, \mathsf{salt}, \tilde{h}^{[1]}, \ldots, \tilde{h}^{[h]})$.
6. Compute $h_2' = \mathrm{Hash}_2(\mathsf{salt}, h_1, \llbracket \alpha^{[1]} \rrbracket_S, \ldots, \llbracket \alpha^{[N]} \rrbracket_S)$.
7. Output Accept iff $h_1' \stackrel{?}{=} h_1$, $h_2' \stackrel{?}{=} h_2$ and $\forall e \in [1:\tau], g(\alpha^{[e]}) \stackrel{?}{=} 0$.

Protocol 16: Signature Scheme – Verification algorithm

We first consider the notion of *unforgeability against key-only attacks* (EUF-KO). In this setting, the adversary is only given a public key x and she attempts to generate a pair (m, σ) such that σ is a valid signature of m with respect to x. The following lemma shows the EUF-KO of the signature scheme in the random oracle model and with respect to the OWF security of the function F.

Lemma C.1. *Let* Hash_0, Hash_1, Hash_2, Hash_m *and* Expand *be modeled as random oracles, and let* (N, τ, λ, p) *be parameters of the signature scheme. Let* $\mathcal{A}$ *be an adversary against the EUF-KO security of the scheme running in time* $t_{\mathcal{A}}$ *and making a total of* Q *random oracle queries. Assuming that* F *is an* $(t_{\mathcal{A}}, \epsilon_{OWF})$*-hard one-way function, then* $\mathcal{A}$*'s advantage in the EUF-KO game is*

$$\epsilon_{EUF\text{-}KO} \leq \epsilon_{OWF} + \frac{(\tau N + 1)Q^2}{2^{2\lambda}} + \Pr[X + Y = \tau],$$

with

- $X = \max_{q_1 \in \mathcal{Q}_1}\{X_{q_1}\}$ *with* $X_{q_1} \sim \mathfrak{B}\left(\tau, \binom{N}{\ell+1}p\right)$ *and*
- $Y = \max_{q_2 \in \mathcal{Q}_2}\{Y_{q_2}\}$ *with* $Y_{q_2} \sim \mathfrak{B}\left(\tau - X, 1/\binom{N}{\ell}\right)$,

where $\mathfrak{B}(n_0, p_0)$ *denotes the binomial distribution with* n_0 *the number of trials and* p_0 *the success probability of each trial.*

The following proof is highly inspired (and carbon copied where relevant) from the proofs of [BDK+21b, Lemma 2] and [KZ22, Lemma 5].

Proof. We give an algorithm $\mathcal{B}$ (the reduction) which uses the EUF-KO adversary $\mathcal{A}$ to compute a pre-image for the key generation function F.

Algorithm $\mathcal{B}$ simulates the EUF-KO game using the random oracles Hash_0, Hash_1, Hash_2 and Hash_m and query lists Q_c, Q_1, Q_2 and Q_m. In addition, $\mathcal{B}$ maintains two tables $\mathcal{T}_{\mathsf{sh}}$ and $\mathcal{T}_{\mathsf{wit}}$ which respectively store the shares of the parties and the corresponding witnesses (for a given set of $\ell + 1$ shares J) that $\mathcal{B}$ recovers from $\mathcal{A}$'s RO queries. $\mathcal{B}$ also maintains a set Bad to keep track of the outputs of all four random oracles. We also ignore calls to Expand in our analysis, since they are used to expand outputs from Hash_1 and Hash_2: when Expand is a random function this is equivalent to increasing the output lengths of Hash_1 and Hash_2.

Behavior of $\mathcal{B}$. On input x, a OWF challenge, algorithm $\mathcal{B}$ forwards it to $\mathcal{A}$ as a signature public key for the EUF-KO game. It lets $\mathcal{A}$ run and answer its random oracle queries in the following way. We assume (wlog.) that Algorithm 1, Algorithm 2, Algorithm 3 and Algorithm 4 only consider queries that are correctly formed, and ignore duplicate queries. In these algorithms, the notation $v \to \mathcal{Q}$ is used to mean $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{v\}$.

- Hash_0: When $\mathcal{A}$ queries the commitment random oracle, $\mathcal{B}$ records the query to learn which commitment corresponds to which input share. See Algorithm 1.
- Hash_m: As Hash_0, it records the query to link the commitment with the Merkle roots. See Algorithm 2.

- Hash_1: When $\mathcal{A}$ sends the Merkle roots for the share commitments, $\mathcal{B}$ checks whether these roots were output by a right use of a Merkle tree simulated by Hash_m and for which the leaves were output by its simulation of Hash_0. If any were for some e and i, then $\mathcal{B}$ is able to reconstruct the shares for party i in repetition e. If $\mathcal{B}$ is able to reconstruct the shares for a subset J of $\ell + 1$ parties for an execution e, then it can extract the corresponding witness value $w^{[e](J)}$ used by $\mathcal{A}$ (for this execution e and this subset of parties J). See Algorithm 3. *Note: The algorithm description also include the computation of further values that are part of the protocol which are useless to Algorithm 3 and only included for notation purpose in order to make the analysis of the case* $\Pr[\mathcal{A}$ *wins* $\mid \mathcal{B}$ *outputs* $\perp]$ *easier to follow.*

- Hash_2: No extraction takes place during this random oracle simulation. See Algorithm 4.

Algorithm 1 Hash function Hash_0

$\underline{\text{Hash}_0(q_0 = (\mathsf{salt}, e, i, [\![w]\!]_i, [\![\beta]\!]_i))}$:
- $\mathsf{com} \xleftarrow{\$} \{0,1\}^{2\lambda}$.
- If $\mathsf{com} \in \mathsf{Bad}$, then abort.
- $\mathsf{com} \to \mathsf{Bad}$.
- $(q_0, \mathsf{com}) \to \mathcal{Q}_0$.
- Return com.

Algorithm 2 Hash function Hash_m

$\underline{\text{Hash}_m(q_m = (h_1, h_2))}$:
- $h_1 \to \mathsf{Bad}$
- $h_2 \to \mathsf{Bad}$
- $h_m \xleftarrow{\$} \{0,1\}^{2\lambda}$.
- If $h_m \in \mathsf{Bad}$, then abort.
- $h_m \to \mathsf{Bad}$.
- $(q_m, h_m) \to \mathcal{Q}_m$.
- Return h_m.

In the rest of the proof, we assume that $\mathcal{A}$ returns a pair (m, σ) if and only if it is valid (with probability 1). This is wlog. since $\mathcal{A}$ can check that (m, σ) passes the verification before returning it without any degradation of her success probability. As a consequence, the hash h_1 in the returned (valid) signature has necessarily been obtained through a query q_1 to Hash_1 of the form $q_1 = (m, \mathsf{salt}, \tilde{h}^{[1]}, \ldots, \tilde{h}^{[1:\tau]})$. Moreover, all the hash computations from the commitments $\mathsf{com}_i^{[e]}$, with $i \in I^{[e]}$, to the Merkle root $\tilde{h}^{[e]}$ must have been obtained through valid requests to Hash_m (otherwise the verification of $\mathsf{auth}^{[e]}$ would fail with overwhelming probability). Similarly, all the commitments $\mathsf{com}_i^{[e]}$, with $i \in I^{[e]}$, must have been obtained through valid calls to Hash_0. This notably implies that the table $\mathcal{T}_{\mathsf{sh}}$ filled by Algorithm 3 satisfies $\mathcal{T}_{\mathsf{sh}}[q_1, e, i] \neq \emptyset$ for every (e, i) such that $i \in I^{[e]}$.

When $\mathcal{A}$ terminates, $\mathcal{B}$ checks the $\mathcal{T}_{\mathsf{wit}}$ table for any entry where the extracted $w^{[e](J)}$ is consistent with x. If a match is found, $\mathcal{B}$ outputs $w^{[e](J)}$ as a pre-image for the OWF, otherwise $\mathcal{B}$ outputs $\perp$.

Algorithm 3 Hash function Hash_1

$\underline{\mathrm{Hash}_1(q_1)}$:
Parse q_1 as $(m, \mathsf{salt}, \tilde{h}^{[1]}, \ldots, \tilde{h}^{[1:\tau]})$
For $e \in [1:\tau], i \in [1:N]$, do $\tilde{h}^{[e]} \to \mathsf{Bad}$.

For $(e,i) \in [1:\tau] \times [1:N]$ such that $\exists \mathsf{com}_i^{[e]}$: $\mathsf{com}_i^{[e]}$ is the ith leaf of the Merkle tree with root $\tilde{h}^{[e]}$ where nodes are in $\mathcal{Q}_m$, do
If $\exists([\![w]\!]_i, [\![\beta]\!]_i)$ s.t. $((\mathsf{salt}, e, i, [\![w]\!]_i, [\![\beta]\!]_i), \mathsf{com}_i^{[e]}) \in \mathcal{Q}_0$, then
$([\![w]\!]_i, [\![\beta]\!]_i) \to \mathcal{T}_{\mathsf{sh}}[q_1, e, i]$.

For each $e \in [1:\tau]$ and $J \subset [1:N]$, do
If $\mathcal{T}_{\mathsf{sh}}[q_1, e, i] \neq \emptyset$, $\forall i \in J$, then
$w^{[e](J)} \leftarrow \mathsf{Reconstruct}_J([\![w]\!]_J)$.
$\beta^{[e](J)} \leftarrow \mathsf{Reconstruct}_J([\![\beta]\!]_J)$.
$w^{[e](J)} \to \mathcal{T}_{\mathsf{wit}}[q_1, e, J]$.

$h_1 \xleftarrow{\$} \{0,1\}^{2\lambda}$.
If $h_1 \in \mathsf{Bad}$, then abort.
$h_1 \to \mathsf{Bad}$.
$(q_1, h_1) \to \mathcal{Q}_1$.

▷ This gray block is for notation purpose only
$(\varepsilon^{[e]})_{e \in [1:\tau]} \leftarrow \mathrm{Expand}(h_1)$
For each $e \in [1:\tau]$ and $J \subset [1:N]$:$\mathcal{T}_{\mathsf{wit}}[q_1, e, J] \neq \emptyset$, do
$\alpha^{[e](J)} = \varphi_{\varepsilon^{[e]}}(w^{[e](J)}, \beta^{[e](J)})$
For each $(e,i) \in [1:\tau] \times [1:N]$:$\mathcal{T}_{\mathsf{sh}}[q_1, e, i] \neq \emptyset$, do
$[\![\bar{\alpha}^{[e]}]\!]_i = \varphi_{\varepsilon^{[e]}}([\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!])$

Return h_1.

Algorithm 4 Hash function Hash_2

$\underline{\mathrm{Hash}_2(q_2)}$:
Parse q_2 as $(\mathsf{salt}, h_1, ([\![\alpha^{[e]}]\!]_S)_{e \in [1:\tau]})$.
$h_1 \to \mathsf{Bad}$.
$h_2 \xleftarrow{\$} \{0,1\}^{2\lambda}$.
If $h_2 \in \mathsf{Bad}$, then abort.
$h_2 \to \mathsf{Bad}$.
$(q_2, h_2) \to \mathcal{Q}_2$.
Return h_2.

Advantage of the reduction. Given the behavior presented above, we have the following by the law of total probability:

$$\begin{aligned}\Pr[\mathcal{A} \text{ wins}] &= \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{B} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{B} \text{ ouputs } \perp] \\ &\quad + \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{B} \text{ ouputs } w] \\ &\leq \Pr[\mathcal{B} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \mid \mathcal{B} \text{ ouputs } \perp] \\ &\quad + \Pr[\mathcal{B} \text{ ouputs } w]. \end{aligned} \tag{7}$$

Let Q_{com}, Q_m, Q_1 and Q_2 denote the number of queries made by $\mathcal{A}$ to each respective random oracle. Given the way in which values are added to Bad, we have:

$$\begin{aligned}\Pr[\mathcal{B} \text{ aborts}] &= (\#\text{times a digest is sampled}) \cdot \Pr[\mathcal{B} \text{ aborts at that digest}] \\ &\leq (Q_{\mathsf{com}} + Q_m + Q_1 + Q_2) \cdot \frac{\max|\mathsf{Bad}|}{2^{2\lambda}} \\ &= (Q_{\mathsf{com}} + Q_m + Q_1 + Q_2) \cdot \frac{Q_{\mathsf{com}} + 3Q_m + (\tau N + 1)Q_1 + 2Q_2}{2^{2\lambda}} \\ &\leq \frac{(\tau N + 1)(Q_{\mathsf{com}} + Q_m + Q_1 + Q_2)^2}{2^{2\lambda}} \end{aligned} \tag{8}$$

By definition, we also have

$$\Pr[\mathcal{B} \text{ ouputs } w] \leq \epsilon_{\mathrm{OWF}} \ .$$

It remains to deal with the term $\Pr[\mathcal{A} \text{ wins} \mid \mathcal{B} \text{ ouputs } \perp]$. Namely, we now analyze the probability of $\mathcal{A}$ winning the EUF-KO experiment conditioned on the event that $\mathcal{B}$ outputs $\perp$, i.e., no pre-image to x was found on the query lists. For the rest of the proof, we assume that $\mathcal{B}$ outputs $\perp$.

Cheating in the first round. For any query $(q_1, h_1) \in \mathcal{Q}_1$, and its corresponding expanded answer $(\varepsilon^{[e]})_{e\in[1:\tau]}$, let $G_1(q_1, h_1)$ be the set of indices $e \in [1 : \tau]$ of "good executions" where there exists J such that both $\mathcal{T}_{\mathsf{wit}}[q_1, e, J]$ is non-empty and $g(\alpha^{[e](J)}) = 0$, namely a false positive occurs for at least one set J for execution e (since $w^{[e](J)}$ cannot satisfy $(x, w^{[e](J)}) \in \mathcal{R}$ since $\mathcal{B}$ outputs $\perp$). We then have, for every $e \in [1 : \tau]$,

$$\Pr[e \in G_1(q_1, h_1) \mid \mathcal{B} \text{ outputs } \perp] \leq \binom{N}{\ell + 1} p$$

where p is the false-positive rate of the underlying MPC protocol, given that h_1 is distributed uniformly at random (which holds since Hash_1 and Expand are random functions).

As the response h_1 is uniform, each $e \in [1 : \tau]$ has the same independent probability of being in $G_1(q_1, h_1)$. We therefore have that $\#G_1(q_1, h_1) \sim X_{q_1}$ where $X_{q_1} = \mathfrak{B}\left(\tau, \binom{N}{\ell+1}p\right)$, the binomial distribution with τ trials, each with success probability $\binom{N}{\ell+1}p$. Letting $(q_{1\mathsf{best}}, h_{1\mathsf{best}})$ denote the query-response pair which maximizes $\#G_1(q_1, h_1)$, we then have that

$$\#G_1(q_{1\mathsf{best}}, h_{1\mathsf{best}}) \sim X = \max_{q_1 \in \mathcal{Q}_1} \{X_{q_1}\}.$$

Cheating in the second round. Let $(q_2, h_2) \in \mathcal{Q}_2$ be the query such that h_2 is used in the valid signature returned by the adversary. Since the returned signature is valid (with

probability 1), there must also exist $(q_1, h_1) \in \mathcal{Q}_1$ such that h_1 is used in the signature and q_2 is of the form $q_2 = (h_1, \ldots)$. Then for each "bad" first-round execution $e \in [1:\tau] \backslash G_1(q_1, h_1)$, either the verification failed, in which case $\mathcal{A}$ couldn't have won, or the verification passed, despite $\forall J, g(\alpha^{[e](J)}) \neq 0$. We shall denote $\mathcal{H}^{[e]}(q_2)$ the set of *honest parties*, *i.e.* the set of the parties for which the committed shares $[\![\alpha^{[e]}]\!]_i$, are consistent with the committed input shares $[\![w^{[e]}]\!]_i$ and $[\![\beta^{[e]}]\!]_i$. More formally,

$$\mathcal{H}^{[e]}(q_2) = \left\{ i : [\![\alpha^{[e]}]\!]_i = \varphi_{\varepsilon^{[e]}}([\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i) \right\} .$$

Let us consider three cases:

- there is strictly less than ℓ honest parties, *i.e.* $|\mathcal{H}^{[e]}(q_2)| < \ell$, but as ℓ parties are opened, verification would fail;
- there is strictly more than ℓ honest parties, *i.e.* $|\mathcal{H}^{[e]}(q_2)| > \ell$, but it would imply that there exists a set J of $\ell + 1$ honest parties ($\mathcal{J} \subset \mathcal{H}^{[e]}(q_2), |J| = \ell + 1$). In that case, we get that $\alpha^{[e](J)} = \alpha^{[e]}$ where $\alpha^{[e]}$ is the value encoded by $[\![\alpha^{[e]}]\!]_S$ in q_2. However, since $e \in [1:\tau] \backslash G_1(q_1, h_1)$, $g(\alpha^{[e](J)}) \neq 0$ implies that $g(\alpha^{[e]}) \neq 0$, and so verification would fail;
- there is exactly ℓ honest parties, *i.e.* $|\mathcal{H}^{[e]}(q_2)| = \ell$, which is the only possible case given that $\mathcal{A}$ wins with q_2.

Since the expanded $h_2 = (I^{[e]})_{e \in [1:\tau]} \in \{I \subset [1:N] : |I| = \ell\}^\tau$ is distributed uniformly at random, the probability that the verification passes while cheating for all such "bad" first-round executions e is

$$\left(\frac{1}{\binom{N}{\ell}} \right)^{\tau - \#G_1(q_1, h_1)} \leq \left(\frac{1}{\binom{N}{\ell}} \right)^{\tau - \#G_1(q_{1\mathsf{best}}, h_{1\mathsf{best}})} .$$

The probability that this happens for at least one of the $\mathcal{Q}_2$ queries made to Hash_2 is

$$\Pr\left[\mathcal{A} \text{ wins} \mid \begin{array}{c} \mathcal{B} \text{ outputs } \perp \\ \#G_1(q_{1\mathsf{best}}, h_{1\mathsf{best}}) = \tau_1 \end{array} \right] \leq 1 - \left(1 - \left(\frac{1}{\binom{N}{\ell}} \right)^{\tau - \tau_1} \right)^{Q_2} .$$

Finally conditioning on $\mathcal{B}$ outputting $\perp$ and summing over all values of τ_1, we have that

$$\Pr[A \text{ wins} \mid \mathcal{B} \text{ outputs } \perp] \leq \Pr[X + Y = \tau] \tag{9}$$

where X is as before, and $Y = \max_{q_2 \in \mathcal{Q}_2}\{Y_{q_2}\}$ where the Y_{q_2} variables are independently and identically distributed as $\mathcal{B}(\tau - X, 1/\binom{N}{\ell})$.

Conclusion. Bringing Equation (7), Equation (8) and Equation (9) together, we obtain the following:

$$\Pr[\mathcal{A} \text{ wins}] \leq \frac{(\tau N + 1)(Q_{\mathsf{com}} + Q_m + Q_1 + Q_2)^2}{2^{2\lambda}} + \Pr[X + Y = \tau] + \Pr[\mathcal{B} \text{ outputs } w].$$

Assuming KeyGen is an ϵ_{OWF}-secure OWF and setting $Q = Q_{\mathsf{com}} + Q_m + Q_1 + Q_2$ gives the required bound and concludes the proof. □

We now consider the notion of *unforgeability against chosen message attacks* (EUF-CMA). In this setting, the adversary is given a public key x and she can ask an oracle (called the *signature oracle*) to sign messages $(m_1, \ldots, m_r)$ that she can select at will. The goal of the adversary is to generate a pair (m, σ) such that m is not one of requests to the signature oracle and such that σ is a valid signature of m with respect to x. The following theorem shows the EUF-CMA of the signature scheme in the random oracle model.

Theorem C.2. *Let* Hash_0, Hash_1, Hash_2, Hash_m *and* Expand *be modeled as random oracles, and let* (N, τ, λ, p) *be parameters of the signature scheme. Let* $\mathcal{A}$ *be an adversary against the EUF-CMA security of the scheme running in time* $t_{\mathcal{A}}$ *and making a total of* Q_{RO} *random oracle queries and* Q_{sign} *signing queries. Assuming that* F *is an* $(t_{\mathcal{A}}, \epsilon_{OWF})$*-hard one-way function and that* PRG *is a* $(t_{\mathcal{A}}, \epsilon_{PRG})$*-secure pseudorandom generator then* $\mathcal{A}$*'s advantage in the EUF-CMA game is*

$$\epsilon_{EUF\text{-}CMA} \leq \epsilon_{OWF} + \epsilon_{PRG} + \frac{(\tau N + 2)Q^2}{2^{2\lambda}} + \Pr[X + Y = \tau],$$

where $Q = Q_{RO} + N_{Hash} \cdot Q_{Sig}$ *with* $N_{Hash} = 2 + \tau(2N - 1)$ *the number of hash computations in a signature generation, and where* X, Y *are defined as in Lemma C.1.*

Proof. We consider the reduction algorithm $\mathcal{B}$ described in the proof of Lemma C.1 which we extend to answer to the signing queries of the adversary $\mathcal{A}$. We consider three different games:

- Game 0: $\mathcal{B}$ uses a signature oracle $\mathcal{O}_{\mathrm{Sig}}(w, x, \cdot)$ which perfectly answers signing queries from $\mathcal{A}$;
- Game 1: the signature oracle is replaced by $\mathcal{O}'_{\mathrm{Sig}}(w, x, \cdot)$ which perfectly answers signing queries from $\mathcal{A}$ except that the calls to the PRG are replaced with true randomness;
- Game 2: the signature oracle is replaced by a simulator $\mathcal{S}_{\mathrm{Sig}}(x, \cdot)$ answering signing queries from $\mathcal{A}$ without being given the secret witness as input.

In Game 0, $\mathcal{B}$ behaves exactly as in the proof of Lemma C.1 while additionally answering the signing queries from $\mathcal{A}$ using $\mathcal{O}_{\mathrm{Sig}}(w, x, \cdot)$. The signing oracle $\mathcal{O}_{\mathrm{Sig}}(w, x, \cdot)$ makes queries to the random oracles Hash_0, Hash_1, Hash_2 and Hash_m which are answered by $\mathcal{B}$ as in the proof of Lemma C.1, and it computes the signature from the input message and the key pair (w, x) as described in Protocol 15. The total number of random oracle queries is hence of $Q = Q_{\mathrm{RO}} + N_{\mathrm{Hash}} \cdot Q_{\mathrm{Sig}}$. The signing queries being perfectly answered, $\mathcal{A}$ produces a valid signature with probability $\epsilon_{\text{EUF-CMA}}$. Then, by the proof of Lemma C.1, $\mathcal{B}$ interacting with $\mathcal{A}$ and $\mathcal{O}_{\mathrm{Sig}}(w, x, \cdot)$ recovers w with probability $\epsilon_{\mathrm{Game0}}$ such that

$$\epsilon_{\text{EUF-CMA}} \leq \epsilon_{\mathrm{Game0}} + \frac{(\tau N + 1)Q^2}{2^{2\lambda}} + \Pr[X + Y = \tau] \ .$$

We note that this is insufficient to prove our security statement since the reduction $\mathcal{B}$ should not have access to the oracle $\mathcal{O}_{\mathrm{Sig}}(w, x, \cdot)$, which is why we need to transit to Game 2.

Game 1 is similar to Game 0 but the signing oracle $\mathcal{O}_{\mathrm{Sig}}(w, x, \cdot)$ is replaced by an oracle $\mathcal{O}'_{\mathrm{Sig}}(w, x, \cdot)$. The latter works in the exact same way as the original signing oracle except that the the pseudo-randomness $(r_w^{[e]}, r_\beta^{[e]}, r_\psi^{[e]})$ of Phase 1, *i.e.* the outputs of $\mathrm{PRG}(\mathsf{salt}, e, \rho)$, is

replaced by true randomness (independent of the root seed ρ). This is indistinguishable from the previous reduction given that PRG is a secure pseudorandom generator. We deduce that the success probability ϵ_{Game1} of $\mathcal{B}$ to recover w while interacting with $\mathcal{A}$ and $\mathcal{O}'_{\text{Sig}}(w, x, \cdot)$ satisfies

$$|\epsilon_{\text{Game0}} - \epsilon_{\text{Game1}}| \leq \epsilon_{\text{PRG}} \ .$$

Finally, Game 2 is similar to the previous games but the signing oracle is replaced by a simulator $\mathcal{S}_{\text{Sig}}(x, \cdot)$ which does not take the secret key w as input. Additionally, $\mathcal{B}$ keeps a list $\mathcal{S}l$ of all the salts appearing in random oracle queries. Namely, Algorithm 1 (Hash_0), Algorithm 3 (Hash_1) and Algorithm 4 (Hash_2) further perform $\mathsf{salt} \to \mathcal{S}l$ on any query with salt as salt. This simulator is depicted in Algorithm 5. Whenever $\mathcal{S}_{\text{Sig}}$ aborts, $\mathcal{B}$ also aborts.

Algorithm 5 Signing simulator.

$\underline{\mathcal{S}_{\text{Sig}}(x, m)}$:

Sample a random salt $\mathsf{salt} \leftarrow \{0,1\}^{2\lambda}$
If $\mathsf{salt} \in \mathcal{S}l$, then abort
Sample random hashes $h_1 \leftarrow \{0,1\}^{2\lambda}$ and $h_2 \leftarrow \{0,1\}^{2\lambda}$
If $h_1 \in \mathsf{Bad}$ or $h_2 \in \mathsf{Bad}$ then abort
Expand h_1 as $(\varepsilon^{[e]})_{e\in[1:\tau]} \leftarrow \text{Expand}(h_1)$
Expand h_2 as $(I^{[e]})_{e\in[1:\tau]} \leftarrow \text{Expand}(h_2)$
Randomly generate $\alpha^{[e]}$ for every $e \in [1:\tau]$ s.t. $g(\alpha^{[e]}) = 0$
Randomly generate the shares $[\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i$ for every $e \in [1:\tau]$ and $i \in I^{[e]}$
Compute $[\![\alpha^{[e]}]\!]_i := \varphi_{x,\varepsilon^{[e]}}([\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)$ for every $e \in [1:\tau]$ and $i \in I^{[e]}$
From $[\![\alpha^{[e]}]\!]_{I^{[e]}}$ and $\alpha^{[e]}$, reconstruct the shares $[\![\alpha^{[e]}]\!]_S$ for every $e \in [1:\tau]$
Compute $\mathsf{com}_i^{[e]} := \text{Hash}_0(\mathsf{salt}, e, i, [\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)$ for every $e \in [1:\tau]$ and $i \in I^{[e]}$
Sample random commitments $\mathsf{com}_i^{[e]} \leftarrow \{0,1\}^{2\lambda}$ for every $e \in [1:\tau]$ and $i \notin I^{[e]}$
If $\mathsf{com}_i^{[e]} \in \mathsf{Bad}$ for some $e \in [1:\tau]$ and $i \notin I^{[e]}$, then abort
Compute $\tilde{h}^{[e]} := \text{MerkleTree}(\mathsf{com}_1^{[e]}, \ldots, \mathsf{com}_N^{[e]})$ for every $e \in [1:\tau]$
For $q_1 = (m, \mathsf{salt}, \tilde{h}^{[1]}, \ldots, \tilde{h}^{[1:\tau]})$, let $(q_1, h_1) \to Q_1$
For $q_2 = (\mathsf{salt}, h_1, ([\![\alpha^{[e]}]\!]_S)_{e\in[1:\tau]})$, let $(q_2, h_2) \to Q_2$
Let $\mathsf{auth}^{[e]}$ the authentication path for $\{\mathsf{com}_i^{[e]}\}_{i\in I}$ w.r.t. Merkle root $\tilde{h}^{[e]}$
Let $i^{*[e]} \in S \setminus I^{[e]}$
Return $\sigma := [\mathsf{salt} \mid h_1 \mid h_2 \mid (([\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)_{i\in I}, \mathsf{auth}^{[e]}, [\![\alpha^{[e]}]\!]_{i^{*[e]}})_{e\in[1:\tau]}]$

Let us stress that the random generation of $\alpha^{[e]}$ such that $g(\alpha^{[e]}) = 0$ is done according to the real distribution of $\alpha^{[e]}$ in a valid signature. This distribution depends on the MPC protocol and is independent of w (by the zero-knowledge property).

The signing simulator follows the same principle as the zero-knowledge simulator of the proof-of-knowledge protocol. By knowing the challenges beforehand, it can generate a signature with perfect distribution without knowing the secret witness. In the random oracle model, this simply means randomly generating the answers h_1 and h_2 of the oracles Hash_1 and Hash_2 before they are actually queried. In the absence of aborts, we thus get that an answer of $\mathcal{S}_{\text{Sig}}$ on input (x, m) is identically distributed to an answer of $\mathcal{O}'_{\text{Sig}}$ on input $(w, x, \cdot)$.

We have to deal with an additional subtlety. By randomly generating the commitments $\mathsf{com}_i^{[e]}$ for every $e \in [1:\tau]$ and $i \notin I^{[e]}$, the simulator implicitly define some outputs of the Hash_0 oracle for which she does not know the input. Since the sharing $[\![w^{[e]}]\!]$ is fully defined by the shares $[\![w^{[e]}]\!]_{I^{[e]}}$ and the witness w, and the sharing $[\![\beta^{[e]}]\!]$ is fully defined by the shares $[\![\beta^{[e]}]\!]_{I^{[e]}}$ and the plain value $\beta^{[e]} = \psi_x(w; r_\psi^{[e]})$, Hash_0 should be constrained to answer a future request $q_0 = (\mathsf{salt}, e, i, [\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)$ by the $\mathsf{com}_i^{[e]}$ randomly sampled by the simulator whenever a match occurs, *i.e.*, whenever

1. (salt, e, i) are such that salt corresponds to a previous signing request for which $i \neq I^{[e]}$,
2. the shares $[\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i$ in the request q_0 are consistent with the full sharings $[\![w^{[e]}]\!]$ and $[\![\beta^{[e]}]\!]$ defined by the shares $[\![w^{[e]}]\!]_{I^{[e]}}$ and $[\![\beta^{[e]}]\!]_{I^{[e]}}$ of this previous signing request together with the witness w.

To deal with this, we simply modify Algorithm 1 (Hash_0) in the following way. If a new request $q_0 = (\mathsf{salt}, e, i, [\![w^{[e]}]\!]_i, [\![\beta^{[e]}]\!]_i)$ is made for which condition 1 above is satisfied, the algorithm reconstructs a candidate witness w^* from the shares $[\![w^{[e]}]\!]_{I^{[e]}}$ from the previous signing query and the share $[\![w^{[e]}]\!]_i$ from the current q_0 query. In case w^* is a valid witness, *i.e.* $F(w^*) = x$, $\mathcal{B}$ returns w^*. We thus have that such event can only occur if $\mathcal{B}$ outputs w and does not affect the event ($\mathcal{A}$ wins $\mid \mathcal{B}$ ouputs $\perp$) that we are considering here.

The probability to abort due to collisions in Bad is the same for $\mathcal{O}'_{\mathrm{Sig}}$ and $\mathcal{S}_{\mathrm{Sig}}$. Indeed, they both do the same amounts of queries to the random oracles, the simulator just do them in a different order and handle the queries for h_1 and h_2 directly. $\mathcal{S}_{\mathrm{Sig}}$ may further aborts in case of salt collision, which happens with probability at most $(Q_{\mathrm{RO}} + Q_{\mathrm{Sig}})/2^{2\lambda}$. We deduce that the success probability $\epsilon_{\mathrm{Game2}} = \epsilon_{\text{EUF-CMA}}$ of $\mathcal{B}$ to recover w while interacting with $\mathcal{A}$ and simulating signing queries with $\mathcal{S}_{\mathrm{Sig}}(x, \cdot)$ satisfies

$$|\epsilon_{\mathrm{Game1}} - \epsilon_{\mathrm{Game2}}| \leq \frac{Q_{\mathrm{Sig}}(Q_{\mathrm{RO}} + Q_{\mathrm{Sig}})}{2^{2\lambda}} \leq \frac{Q^2}{2^{2\lambda}}$$

which concludes the proof. □

Bibliography

[AABN02] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. "From Identification to Signatures via the Fiat-Shamir Transform: Minimizing Assumptions for Security and Forward-Security". In: *EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. LNCS. Springer, Heidelberg, Apr. 2002, pp. 418–433. DOI: 10.1007/3-540-46035-7_28 (cit. on pp. 36, 48, 71).

[ABB+22] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. *SPHINCS+ – Submission to the NIST post-quantum project*. Version 3.1 – 10 June 2022. https://sphincs.org/data/sphincs+-r3.1-specification.pdf. 2022 (cit. on pp. 2, 131, 157).

[ABB+23a] Nicolas Aragon, Magali Bardet, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Victor Dyseryn, Thibauld Feneuil, Philippe Gaborit, Antoine Joux, Matthieu Rivain, Jean-Pierre Tillich, and Adrien Vinçotte. *RYDE Specifications*. June 2023. 2023 (cit. on p. 169).

[ABB+23b] Nicolas Aragon, Magali Bardet, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Victor Dyseryn, Thibauld Feneuil, Philippe Gaborit, Romaric Neveu, Matthieu Rivain, and Jean-Pierre Tillich. *MIRA Specifications*. June 2023. 2023 (cit. on p. 168).

[ABD+21] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. *CRYSTALS-Kyber – Algorithm Specifications and Supporting Documentation*. Version 3.02 – 4 August 2021. https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf. 2021 (cit. on p. 2).

[ABG+19] Nicolas Aragon, Olivier Blazy, Philippe Gaborit, Adrien Hauteville, and Gilles Zémor. "Durandal: A Rank Metric Based Signature Scheme". In: *EUROCRYPT 2019, Part III*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. LNCS. Springer, Heidelberg, May 2019, pp. 728–758. DOI: 10.1007/978-3-030-17659-4_25 (cit. on pp. 37, 55, 60).

[ACBH13] Sidi Mohamed El Yousfi Alaoui, Pierre-Louis Cayrel, Rachid El Bansarkhani, and Gerhard Hoffmann. "Code-Based Identification and Signature Schemes in Software". In: *Security Engineering and Intelligence Informatics - CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regensburg, Germany, September 2-6, 2013. Proceedings*. 2013, pp. 122–136 (cit. on p. 36).

[AD97] Miklós Ajtai and Cynthia Dwork. "A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence". In: *29th ACM STOC*. ACM Press, May 1997, pp. 284–293. DOI: 10.1145/258533.258604 (cit. on p. 82).

[AFG+23] Carlos Aguilar Melchor, Thibauld Feneuil, Nicolas Gama, Shay Gueron, James Howe, David Joseph, Antoine Joux, Edoardo Persichetti, Tovohery H. Randrianarisoa, Matthieu Rivain, and Dongze Yue. *The Syndrome Decoding in the Head (SD-in-the-Head) Signature Scheme – Algorithm Specifications and Supporting Documentation*. Version 1.0 – 31 May 2023. 2023 (cit. on p. 168).

[AFS03] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. *A Fast Provably Secure Cryptographic Hash Function*. Cryptology ePrint Archive, Report 2003/230. https://eprint.iacr.org/2003/230. 2003 (cit. on p. 61).

[AGH+23] Carlos Aguilar Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. "The Return of the SDitH". In: *EUROCRYPT 2023, Part V*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14008. LNCS. Springer, Heidelberg, Apr. 2023, pp. 564–596. DOI: 10.1007/978-3-031-30589-4_20 (cit. on pp. 25, 80, 126, 163, 167, 168, 173).

[AGS11] Carlos Aguilar, Philippe Gaborit, and Julien Schrek. "A new zero-knowledge code based identification scheme with reduced communication". In: *2011 IEEE Information Theory Workshop*. 2011, pp. 648–652. DOI: 10.1109/ITW.2011.6089577 (cit. on pp. 36, 70).

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 2087–2104. DOI: 10.1145/3133956.3134104 (cit. on pp. 22–24, 26, 30, 130, 132, 135, 154, 175, 176).

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. "Ciphers for MPC and FHE". In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 430–454. DOI: 10.1007/978-3-662-46800-5_17 (cit. on pp. 56, 79, 108).

[ARV22] Gora Adj, Luis Rivera-Zamarripa, and Javier Verbel. *MinRank in the Head: Short Signatures from Zero-Knowledge Proofs*. Cryptology ePrint Archive, Report 2022/1501. https://eprint.iacr.org/2022/1501. 2022 (cit. on pp. 33, 121, 122).

[BBC+19a] Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. "A Finite Regime Analysis of Information Set Decoding Algorithms". In: *Algorithms* 12.10 (2019), p. 209 (cit. on p. 52).

[BBC+19b] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. "Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs". In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 67–97. DOI: 10.1007/978-3-030-26954-8_3 (cit. on pp. 30, 31).

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Scalable Zero Knowledge with No Trusted Setup". In: *CRYPTO 2019, Part III*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11694. LNCS. Springer, Heidelberg, Aug. 2019, pp. 701–732. DOI: 10.1007/978-3-030-26954-8_23 (cit. on p. 130).

[BBP+23] Marco Baldi, Sebastian Bitzer, Alessio Pavoni, Paolo Santini, Antonia Wachter-Zeh, and Violetta Weger. *Zero Knowledge Protocols and Signatures from the Restricted Syndrome Decoding Problem*. Cryptology ePrint Archive, Paper 2023/385. https://eprint.iacr.org/2023/385. 2023. URL: https://eprint.iacr.org/2023/385 (cit. on p. 57).

[BBPS21] Alessandro Barenghi, Jean-François Biasse, Edoardo Persichetti, and Paolo Santini. "LESS-FM: Fine-Tuning Signatures from the Code Equivalence Problem". In: *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*. Ed. by Jung Hee Cheon and Jean-Pierre Tillich. Springer, Heidelberg, 2021, pp. 23–43. DOI: 10.1007/978-3-030-81293-5_2 (cit. on p. 54).

[BBSS20] Xavier Bonnetain, Rémi Bricout, André Schrottenloher, and Yixin Shen. "Improved Classical and Quantum Algorithms for Subset-Sum". In: *ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. LNCS. Springer, Heidelberg, Dec. 2020, pp. 633–666. DOI: 10.1007/978-3-030-64834-3_22 (cit. on pp. 82, 97, 98).

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. "Aurora: Transparent Succinct Arguments for R1CS". In: *EUROCRYPT 2019, Part I*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11476. LNCS. Springer, Heidelberg, May 2019, pp. 103–128. DOI: 10.1007/978-3-030-17653-2_4 (cit. on p. 130).

[BD20] Ward Beullens and Cyprien Delpech de Saint Guilhem. "LegRoast: Efficient Post-quantum Signatures from the Legendre PRF". In: *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*. Ed. by Jintai Ding and Jean-Pierre Tillich. Springer, Heidelberg, 2020, pp. 130–150. DOI: 10.1007/978-3-030-44223-1_8 (cit. on pp. 32, 102, 133, 136, 139, 154).

[BDK+21a] Shi Bai, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. *CRYSTALS-Dilithium – Algorithm Specifications and Supporting Documentation*. Version 3.1 – 8 February 2021. https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf. 2021 (cit. on pp. 2, 78).

[BDK+21b] Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. "Banquet: Short and Fast Signatures from AES". In: *PKC 2021, Part I*. Ed. by Juan Garay. Vol. 12710. LNCS. Springer, Heidelberg, May 2021, pp. 266–297. DOI: 10.1007/978-3-030-75245-3_11 (cit. on pp. 27, 30–32, 56, 60, 79, 88, 108, 133, 136, 151, 154, 156, 190).

[Bea92] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. LNCS. Springer, Heidelberg, Aug. 1992, pp. 420–432. DOI: 10.1007/3-540-46766-1_34 (cit. on p. 28).

[BESV22] Emanuele Bellini, Andre Esser, Carlo Sanna, and Javier Verbel. "MR-DSS – Smaller MinRank-Based (Ring-)Signatures". In: *Post-Quantum Cryptography*. Ed. by Jung Hee Cheon and Thomas Johansson. Cham: Springer International Publishing, 2022, pp. 144–169. ISBN: 978-3-031-17234-2 (cit. on pp. 108, 120–122).

[Beu20] Ward Beullens. "Sigma Protocols for MQ, PKP and SIS, and Fishy Signature Schemes". In: *EUROCRYPT 2020, Part III*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12107. LNCS. Springer, Heidelberg, May 2020, pp. 183–211. DOI: 10.1007/978-3-030-45727-3_7 (cit. on pp. 27, 42, 55, 83, 98, 108, 110, 115, 121).

[BFH+20] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. "Ligero++: A New Optimized Sublinear IOP". In: *ACM CCS 2020*. Ed. by Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna. ACM Press, Nov. 2020, pp. 2025–2038. DOI: 10.1145/3372297.3417893 (cit. on p. 154).

[BG22] Loïc Bidoux and Philippe Gaborit. *Compact Post-Quantum Signatures from Proofs of Knowledge leveraging Structure for the PKP, SD and RSD Problems*. 2022. arXiv: 2204.02915 [cs.CR] (cit. on pp. 57, 108, 109, 121, 122, 125, 126).

[BGKM22] Loïc Bidoux, Philippe Gaborit, Mukul Kulkarni, and Victor Mateu. *Code-based Signatures from New Proofs of Knowledge for the Syndrome Decoding Problem*. 2022. arXiv: 2201.05403 [cs.CR] (cit. on pp. 77, 78).

[BGKW90] Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. "Efficient Identification Schemes Using Two Prover Interactive Proofs". In: *CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. LNCS. Springer, Heidelberg, Aug. 1990, pp. 498–506. DOI: 10.1007/0-387-34805-0_44 (cit. on p. 82).

[BHH01] Dan Boneh, Shai Halevi, and Nick Howgrave-Graham. "The Modular Inversion Hidden Number Problem". In: *ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. LNCS. Springer, Heidelberg, Dec. 2001, pp. 36–51. DOI: 10.1007/3-540-45682-1_3 (cit. on pp. 4, 33, 81, 84, 100, 102, 104, 105, 154).

[BHK+19] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. "The SPHINCS$^+$ Signature Framework". In: *ACM CCS 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 2129–2146. DOI: 10.1145/3319535.3363229 (cit. on pp. 56, 60, 79).

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. "Decoding Random Binary Linear Codes in $2^{n/20}$: How $1 + 1 = 0$ Improves Information Set Decoding". In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 520–536. DOI: 10.1007/978-3-642-29011-4_31 (cit. on p. 52).

[Blo09] Jeremiah Blocki. *Direct zero-knowledge proofs*. Senior Research Thesis, B.S. in Computer Science, Carnegie Mellon University. 2009 (cit. on p. 82).

[BMPS20] Jean-François Biasse, Giacomo Micheli, Edoardo Persichetti, and Paolo Santini. "LESS is More: Code-Based Signatures Without Syndromes". In: *AFRICACRYPT 20*. Ed. by Abderrahmane Nitaj and Amr M. Youssef. Vol. 12174. LNCS. Springer, Heidelberg, July 2020, pp. 45–65. DOI: 10.1007/978-3-030-51938-4_3 (cit. on p. 54).

[BMSV22] Emanuele Bellini, Rusydi H. Makarim, Carlo Sanna, and Javier A. Verbel. "An Estimator for the Hardness of the MQ Problem". In: *AFRICACRYPT 22*. Ed. by Lejla Batina and Joan Daemen. Vol. 2022. LNCS. Springer Nature, July 2022, pp. 323–347. DOI: 10.1007/978-3-031-17433-9_14 (cit. on p. 114).

[BN20] Carsten Baum and Ariel Nof. "Concretely-Efficient Zero-Knowledge Arguments for Arithmetic Circuits and Their Application to Lattice-Based Cryptography". In: *PKC 2020, Part I*. Ed. by Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas. Vol. 12110. LNCS. Springer, Heidelberg, May 2020, pp. 495–526. DOI: 10.1007/978-3-030-45374-9_17 (cit. on pp. 25, 28–31, 60, 83, 88, 101, 111, 130, 133, 136, 144, 145, 151, 154, 157).

[BVZ12] Aurélie Bauer, Damien Vergnaud, and Jean-Christophe Zapalowicz. "Inferring Sequences Produced by Nonlinear Pseudorandom Number Generators Using Coppersmith's Methods". In: *PKC 2012*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Vol. 7293. LNCS. Springer, Heidelberg, May 2012, pp. 609–626. DOI: 10.1007/978-3-642-30057-8_36 (cit. on pp. 102, 104).

[Can89] David G. Cantor. "On arithmetical algorithms over finite fields". In: *Journal of Combinatorial Theory, Series A* 50 (1989), pp. 285–300 (cit. on p. 76).

[CC06] Hao Chen and Ronald Cramer. "Algebraic Geometric Secret Sharing Schemes and Secure Multi-Party Computations over Small Fields". In: *CRYPTO 2006*. Ed. by Cynthia Dwork. Vol. 4117. LNCS. Springer, Heidelberg, Aug. 2006, pp. 521–536. DOI: 10.1007/11818175_31 (cit. on pp. 131, 149).

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. "Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 1825–1842. DOI: 10.1145/3133956.3133997 (cit. on pp. 22, 30, 100, 154).

[CDN15] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015. DOI: 10.1017/CBO9781107337756 (cit. on pp. 140, 149).

[CGH00] Dario Catalano, Rosario Gennaro, and Shai Halevi. "Computing Inverses over a Shared Secret Modulus". In: *EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. LNCS. Springer, Heidelberg, May 2000, pp. 190–206. DOI: 10.1007/3-540-45539-6_14 (cit. on p. 83).

[Cha22] André Chailloux. *On the (In)security of optimized Stern-like signature schemes.* WCC 2022: The Twelfth International Workshop on Coding and Cryptography. https://www.wcc2022.uni-rostock.de/storages/uni-rostock/Tagungen/WCC2022/Papers/WCC_2022_paper_54.pdf. 2022 (cit. on p. 49).

[CHR+16] Ming-Shing Chen, Andreas Hülsing, Joost Rijneveld, Simona Samardjiska, and Peter Schwabe. "From 5-Pass MQ-Based Identification to MQ-Based Signatures". In: *ASIACRYPT 2016, Part II.* Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10032. LNCS. Springer, Heidelberg, Dec. 2016, pp. 135–165. DOI: 10.1007/978-3-662-53890-6_5 (cit. on pp. 71, 114, 115).

[CJL+92] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. "Improved Low-Density Subset Sum Algorithms". In: *Comput. Complex.* 2 (1992), pp. 111–128. DOI: 10.1007/BF01201999. URL: https://doi.org/10.1007/BF01201999 (cit. on p. 97).

[Cou01] Nicolas Courtois. "Efficient Zero-Knowledge Authentication Based on a Linear Algebra Problem MinRank". In: *ASIACRYPT 2001.* Ed. by Colin Boyd. Vol. 2248. LNCS. Springer, Heidelberg, Dec. 2001, pp. 402–421. DOI: 10.1007/3-540-45682-1_24 (cit. on pp. 120–122).

[CVE11] Pierre-Louis Cayrel, Pascal Véron, and Sidi Mohamed El Yousfi Alaoui. "A Zero-Knowledge Identification Scheme Based on the q-ary Syndrome Decoding Problem". In: *SAC 2010.* Ed. by Alex Biryukov, Guang Gong, and Douglas R. Stinson. Vol. 6544. LNCS. Springer, Heidelberg, Aug. 2011, pp. 171–186. DOI: 10.1007/978-3-642-19574-7_12 (cit. on pp. 70, 75).

[DDOS19] Cyprien Delpech de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P. Smart. "BBQ: Using AES in Picnic Signatures". In: *SAC 2019.* Ed. by Kenneth G. Paterson and Douglas Stebila. Vol. 11959. LNCS. Springer, Heidelberg, Aug. 2019, pp. 669–692. DOI: 10.1007/978-3-030-38471-5_27 (cit. on pp. 56, 108, 136, 154).

[DKR+21] Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, and Greg Zaverucha. *Shorter Signatures Based on Tailor-Made Minimalist Symmetric-Key Crypto.* Cryptology ePrint Archive, Report 2021/692. https://eprint.iacr.org/2021/692. 2021 (cit. on pp. 27, 102, 108, 154).

[DLO+18] Ivan Damgård, Ji Luo, Sabine Oechsner, Peter Scholl, and Mark Simkin. "Compact Zero-Knowledge Proofs of Small Hamming Weight". In: *PKC 2018, Part II.* Ed. by Michel Abdalla and Ricardo Dahab. Vol. 10770. LNCS. Springer, Heidelberg, Mar. 2018, pp. 530–560. DOI: 10.1007/978-3-319-76581-5_18 (cit. on p. 60).

[DOT21] Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. "Limbo: Efficient Zero-knowledge MPCitH-based Arguments". In: *ACM CCS 2021.* Ed. by Giovanni Vigna and Elaine Shi. ACM Press, Nov. 2021, pp. 3022–3036. DOI: 10.1145/3460120.3484595 (cit. on pp. 23–25, 30, 32, 108, 130–132, 136, 151, 153–156, 160, 162).

[DST19] Thomas Debris-Alazard, Nicolas Sendrier, and Jean-Pierre Tillich. "Wave: A New Family of Trapdoor One-Way Preimage Sampleable Functions Based on Codes". In: *ASIACRYPT 2019, Part I*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11921. LNCS. Springer, Heidelberg, Dec. 2019, pp. 21–51. DOI: 10.1007/978-3-030-34578-5_2 (cit. on pp. 37, 55, 60).

[EDV+12] Sidi Mohamed El Yousfi Alaoui, Özgür Dagdelen, Pascal Véron, David Galindo, and Pierre-Louis Cayrel. "Extended Security Arguments for Signature Schemes". In: *AFRICACRYPT 12*. Ed. by Aikaterini Mitrokotsa and Serge Vaudenay. Vol. 7374. LNCS. Springer, Heidelberg, July 2012, pp. 19–34 (cit. on p. 71).

[Fen22] Thibauld Feneuil. *Building MPCitH-based Signatures from MQ, MinRank, Rank SD and PKP*. Cryptology ePrint Archive, Report 2022/1512. https://eprint.iacr.org/2022/1512. 2022 (cit. on pp. 5, 31, 33, 107, 169).

[FHK+20] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. *Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU*. Version 1.2 – 1 October 2020. https://falcon-sign.info/falcon.pdf. 2020 (cit. on pp. 2, 78, 79).

[FJR21] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. *Shared Permutation for Syndrome Decoding: New Zero-Knowledge Protocol and Code-Based Signature*. Cryptology ePrint Archive, Report 2021/1576. https://eprint.iacr.org/2021/1576. 2021 (cit. on pp. 77, 78, 108, 125, 126).

[FJR22a] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. *Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs*. Cryptology ePrint Archive, Report 2022/188. https://eprint.iacr.org/2022/188. 2022 (cit. on p. 67).

[FJR22b] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. "Syndrome Decoding in the Head: Shorter Signatures from Zero-Knowledge Proofs". In: *CRYPTO 2022, Part II*. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13508. LNCS. Springer, Heidelberg, Aug. 2022, pp. 541–572. DOI: 10.1007/978-3-031-15979-4_19 (cit. on pp. 5, 32, 59, 70, 88, 102, 108, 123, 133, 136, 139, 152–154, 156).

[FJR23] Thibauld Feneuil, Antoine Joux, and Matthieu Rivain. "Shared permutation for syndrome decoding: new zero-knowledge protocol and code-based signature". In: *Des. Codes Cryptogr.* 91.2 (2023), pp. 563–608. DOI: 10.1007/s10623-022-01116-1. URL: https://doi.org/10.1007/s10623-022-01116-1 (cit. on pp. 5, 35, 39, 46, 51, 99).

[FMRV22a] Thibauld Feneuil, Jules Maire, Matthieu Rivain, and Damien Vergnaud. *Zero-Knowledge Protocols for the Subset Sum Problem from MPC-in-the-Head with Rejection*. Cryptology ePrint Archive, Report 2022/223. https://eprint.iacr.org/2022/223. 2022 (cit. on pp. 92, 94, 108).

[FMRV22b] Thibauld Feneuil, Jules Maire, Matthieu Rivain, and Damien Vergnaud. "Zero-Knowledge Protocols for the Subset Sum Problem from MPC-in-the-Head with Rejection". In: *ASIACRYPT 2022, Part II*. Ed. by Shweta Agrawal and Dongdai Lin. Vol. 13792. LNCS. Springer, Heidelberg, Dec. 2022, pp. 371–402. DOI: 10.1007/978-3-031-22966-4_13 (cit. on pp. 5, 27, 29, 32, 33, 81, 105, 136, 153, 154).

[FR22] Thibauld Feneuil and Matthieu Rivain. *Threshold Linear Secret Sharing to the Rescue of MPC-in-the-Head*. Cryptology ePrint Archive, Report 2022/1407. https://eprint.iacr.org/2022/1407. 2022 (cit. on pp. 5, 25, 26, 80, 129, 149).

[FR23] Thibauld Feneuil and Matthieu Rivain. *MQOM: MQ on my Mind – Algorithm Specifications and Supporting Documentation*. Version 1.0 – 31 May 2023. 2023 (cit. on p. 169).

[FS87] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. LNCS. Springer, Heidelberg, Aug. 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12 (cit. on pp. 18, 36, 48, 60, 71, 99, 109, 113, 119, 123, 153, 187).

[GG03] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2003. ISBN: 9780521826464. URL: https://books.google.fr/books?id=NuEHj0wPwgIC (cit. on p. 76).

[GG07] Philippe Gaborit and Marc Girault. "Lightweight code-based identification and signature". In: *IEEE International Symposium on Information Theory, ISIT 2007, Nice, France, June 24-29, 2007*. 2007, pp. 191–195 (cit. on p. 36).

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. "How to Construct Random Functions". In: *Journal of the ACM* 33.4 (Oct. 1986), pp. 792–807. DOI: 10.1145/6490.6503 (cit. on p. 10).

[GM10] Shuhong Gao and Todd Mateer. "Additive Fast Fourier Transforms Over Finite Fields". In: *IEEE Transactions on Information Theory* 56.12 (2010), pp. 6265–6272. DOI: 10.1109/TIT.2010.2079016 (cit. on p. 76).

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. "ZKBoo: Faster Zero-Knowledge for Boolean Circuits". In: *USENIX Security 2016*. Ed. by Thorsten Holz and Stefan Savage. USENIX Association, Aug. 2016, pp. 1069–1083 (cit. on pp. 21, 22, 29, 30, 131, 132, 135, 154).

[GPS22] Shay Gueron, Edoardo Persichetti, and Paolo Santini. "Designing a Practical Code-Based Signature Scheme from Zero-Knowledge Proofs with Trusted Setup". In: *Cryptography* 6.1 (2022). ISSN: 2410-387X. DOI: 10.3390/cryptography6010005. URL: https://www.mdpi.com/2410-387X/6/1/5 (cit. on pp. 55, 70, 75, 78).

[GSV21] Yaron Gvili, Sarah Scheffler, and Mayank Varia. "BooLigero: Improved Sublinear Zero Knowledge Proofs for Boolean Circuits". In: *FC 2021, Part I*. Ed. by Nikita Borisov and Claudia Díaz. Vol. 12674. LNCS. Springer, Heidelberg, Mar. 2021, pp. 476–496. DOI: 10.1007/978-3-662-64322-8_23 (cit. on pp. 24, 130).

[HJ10] Nick Howgrave-Graham and Antoine Joux. "New Generic Algorithms for Hard Knapsacks". In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 235–256. DOI: 10.1007/978-3-642-13190-5_12 (cit. on p. 97).

[HS74] Ellis Horowitz and Sartaj Sahni. "Computing Partitions with Applications to the Knapsack Problem". In: *J. ACM* 21.2 (1974), pp. 277–292. DOI: 10.1145/321812.321823. URL: https://doi.org/10.1145/321812.321823 (cit. on p. 97).

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. "Zero-knowledge from secure multiparty computation". In: *39th ACM STOC*. Ed. by David S. Johnson and Uriel Feige. ACM Press, June 2007, pp. 21–30. DOI: 10.1145/1250790.1250794 (cit. on pp. 19–22, 36, 131, 132).

[IN96] Russell Impagliazzo and Moni Naor. "Efficient Cryptographic Schemes Provably as Secure as Subset Sum". In: *Journal of Cryptology* 9.4 (Sept. 1996), pp. 199–216. DOI: 10.1007/BF00189260 (cit. on pp. 82, 97).

[Kar72] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. Ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Boston, MA: Springer US, 1972, pp. 85–103. ISBN: 978-1-4684-2001-2. DOI: 10.1007/978-1-4684-2001-2_9. URL: https://doi.org/10.1007/978-1-4684-2001-2_9 (cit. on p. 82).

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures". In: *ACM CCS 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM Press, Oct. 2018, pp. 525–537. DOI: 10.1145/3243734.3243805 (cit. on pp. 24, 25, 27, 30, 36, 43, 44, 56, 94, 99, 112, 130, 136, 139, 154).

[KZ20a] Daniel Kales and Greg Zaverucha. "An Attack on Some Signature Schemes Constructed from Five-Pass Identification Schemes". In: *CANS 20*. Ed. by Stephan Krenn, Haya Shulman, and Serge Vaudenay. Vol. 12579. LNCS. Springer, Heidelberg, Dec. 2020, pp. 3–22. DOI: 10.1007/978-3-030-65411-5_1 (cit. on pp. 48, 71, 99, 109, 110, 113, 119, 123, 155, 158, 174).

[KZ20b] Daniel Kales and Greg Zaverucha. "Improving the Performance of the Picnic Signature Scheme". In: *IACR TCHES* 2020.4 (2020). https://tches.iacr.org/index.php/TCHES/article/view/8680, pp. 154–188. ISSN: 2569-2925. DOI: 10.13154/tches.v2020.i4.154-188 (cit. on pp. 56, 60, 78, 79, 133, 154).

[KZ22] Daniel Kales and Greg Zaverucha. *Efficient Lifting for Shorter Zero-Knowledge Proofs and Post-Quantum Signatures*. Cryptology ePrint Archive, Report 2022/588. https://eprint.iacr.org/2022/588. 2022 (cit. on pp. 25, 31, 32, 60, 66, 78, 88, 102, 111, 112, 130, 136, 139, 154, 156, 157, 190).

[LN17] Yehuda Lindell and Ariel Nof. "A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority". In: *ACM CCS 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 259–276. DOI: 10.1145/3133956.3133999 (cit. on pp. 31, 60, 88).

[LN96] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. 2nd ed. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1996. DOI: 10.1017/CBO9780511525926 (cit. on pp. 117, 118).

[LNSW13] San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. "Improved Zero-Knowledge Proofs of Knowledge for the ISIS Problem, and Applications". In: *PKC 2013*. Ed. by Kaoru Kurosawa and Goichiro Hanaoka. Vol. 7778. LNCS. Springer, Heidelberg, Feb. 2013, pp. 107–124. DOI: 10.1007/978-3-642-36362-7_8 (cit. on pp. 82, 84, 98).

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. "Public-Key Cryptographic Primitives Provably as Secure as Subset Sum". In: *TCC 2010*. Ed. by Daniele Micciancio. Vol. 5978. LNCS. Springer, Heidelberg, Feb. 2010, pp. 382–400. DOI: 10.1007/978-3-642-11799-2_23 (cit. on p. 82).

[LSSW12] San Ling, Igor E. Shparlinski, Ron Steinfeld, and Huaxiong Wang. "On the modular inversion hidden number problem". In: *J. Symb. Comput.* 47.4 (2012), pp. 358–367. DOI: 10.1016/j.jsc.2011.09.002. URL: https://doi.org/10.1016/j.jsc.2011.09.002 (cit. on pp. 102, 104).

[Lyu08] Vadim Lyubashevsky. "Lattice-Based Identification Schemes Secure Under Active Attacks". In: *PKC 2008*. Ed. by Ronald Cramer. Vol. 4939. LNCS. Springer, Heidelberg, Mar. 2008, pp. 162–179. DOI: 10.1007/978-3-540-78440-1_10 (cit. on p. 83).

[Lyu09] Vadim Lyubashevsky. "Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures". In: *ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. LNCS. Springer, Heidelberg, Dec. 2009, pp. 598–616. DOI: 10.1007/978-3-642-10366-7_35 (cit. on pp. 37, 83).

[Mer88] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". In: *CRYPTO'87*. Ed. by Carl Pomerance. Vol. 293. LNCS. Springer, Heidelberg, Aug. 1988, pp. 369–378. DOI: 10.1007/3-540-48184-2_32 (cit. on p. 11).

[MH78] R. Merkle and M. Hellman. "Hiding information and signatures in trapdoor knapsacks". In: *IEEE Transactions on Information Theory* 24.5 (1978), pp. 525–530. DOI: 10.1109/TIT.1978.1055927 (cit. on p. 82).

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. "Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$". In: *ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. LNCS. Springer, Heidelberg, Dec. 2011, pp. 107–124. DOI: 10.1007/978-3-642-25385-0_6 (cit. on pp. 52, 74).

[MS78] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. Discrete Mathematics and its Applications. Elsevier Science, 1978. ISBN: 9780444851932 (cit. on p. 140).

[NIS16] National Institute of Standards and Technology (NIST). *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf. 2016 (cit. on pp. 2, 22).

[NIS22] National Institute of Standards and Technology (NIST). *Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process*. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf. 2022 (cit. on pp. 3, 5, 6, 166).

[Odl90] A. M. Odlyzko. *The rise and fall of knapsack cryptosystems*. English. Cryptology and computational number theory, Lect. Notes AMS Short Course, Boulder/CO (USA) 1989, Proc. Symp. Appl. Math. 42, 75-88 (1990). 1990 (cit. on p. 82).

[Pet10] Christiane Peters. "Information-Set Decoding for Linear Codes over F_q". In: *The Third International Workshop on Post-Quantum Cryptography, PQCRYPTO 2010*. Ed. by Nicolas Sendrier. Springer, Heidelberg, May 2010, pp. 81–94. DOI: 10.1007/978-3-642-12929-2_7 (cit. on p. 75).

[PS00] David Pointcheval and Jacques Stern. "Security Arguments for Digital Signatures and Blind Signatures". In: *Journal of Cryptology* 13.3 (June 2000), pp. 361–396. DOI: 10.1007/s001450010003 (cit. on pp. 179, 180).

[Sch90] Claus-Peter Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. LNCS. Springer, Heidelberg, Aug. 1990, pp. 239–252. DOI: 10.1007/0-387-34805-0_22 (cit. on p. 37).

[Sha79] Adi Shamir. "How to Share a Secret". In: *Communications of the Association for Computing Machinery* 22.11 (Nov. 1979), pp. 612–613. DOI: 10.1145/359168.359176 (cit. on p. 131).

[Sha86] Adi Shamir. *A Zero-Knowledge Proof for Knapsacks*. presented at a workshop on Probabilistic Algorithms, Marseille. Mar. 1986 (cit. on pp. 82, 84, 98).

[Sho94] Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700 (cit. on p. 2).

[Sim91] G.J. Simmons. "Identification of data, devices, documents and individuals". In: *Proceedings. 25th Annual 1991 IEEE International Carnahan Conference on Security Technology*. 1991, pp. 197–218. DOI: 10.1109/CCST.1991.202215 (cit. on p. 82).

[SINY22] Bagus Santoso, Yasuhiko Ikematsu, Shuhei Nakamura, and Takanori Yasuda. *Three-Pass Identification Scheme Based on MinRank Problem with Half Cheating Probability*. 2022. arXiv: 2205.03255 [cs.CR] (cit. on pp. 120–122).

[SS81] Richard Schroeppel and Adi Shamir. "A T=O($2^{n/2}$), S=O($2^{n/4}$) Algorithm for Certain NP-Complete Problems". In: *SIAM J. Comput.* 10.3 (1981), pp. 456–464. DOI: 10.1137/0210033. URL: https://doi.org/10.1137/0210033 (cit. on p. 97).

[SSH11] Koichi Sakumoto, Taizo Shirai, and Harunaga Hiwatari. "Public-Key Identification Schemes Based on Multivariate Quadratic Polynomials". In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 706–723. DOI: 10.1007/978-3-642-22792-9_40 (cit. on pp. 114, 115).

[Ste94] Jacques Stern. "A New Identification Scheme Based on Syndrome Decoding". In: *CRYPTO'93*. Ed. by Douglas R. Stinson. Vol. 773. LNCS. Springer, Heidelberg, Aug. 1994, pp. 13–21. DOI: 10.1007/3-540-48329-2_2 (cit. on pp. 36, 37, 70, 82, 125, 126).

[TS16] Rodolfo Canto Torres and Nicolas Sendrier. "Analysis of Information Set Decoding for a Sub-linear Error Weight". In: *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*. Ed. by Tsuyoshi Takagi. Springer, Heidelberg, 2016, pp. 144–161. DOI: 10.1007/978-3-319-29360-8_10 (cit. on p. 52).

[Vér96] Pascal Véron. "Improved identification schemes based on error-correcting codes". In: *Appl. Algebra Eng. Commun. Comput.* 8.1 (1996), pp. 57–69 (cit. on pp. 36, 70, 125, 126).

[Wan22] William Wang. *Shorter Signatures from MQ*. Cryptology ePrint Archive, Report 2022/344. https://eprint.iacr.org/2022/344. 2022 (cit. on pp. 108, 114, 115).

[Was08] Lawrence C. Washington. *Elliptic Curves. Number Theory and Cryptography. 2nd Edition*. Discrete Mathematics and its Applications. Chapman & Hall/CRC Press, 2008. ISBN: 978-1-4200-7146-7 (cit. on p. 149).

[WZ88] Y. Wang and X. Zhu. "A fast algorithm for the Fourier transform over finite fields and its VLSI implementation". In: *IEEE Journal on Selected Areas in Communications* 6.3 (1988), pp. 572–577. DOI: 10.1109/49.1926 (cit. on p. 76).

[XSH+19] Jun Xu, Santanu Sarkar, Lei Hu, Huaxiong Wang, and Yanbin Pan. "New Results on Modular Inversion Hidden Number Problem and Inversive Congruential Generator". In: *CRYPTO 2019, Part I*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. LNCS. Springer, Heidelberg, Aug. 2019, pp. 297–321. DOI: 10.1007/978-3-030-26948-7_11 (cit. on pp. 102, 104).