Interleaved Ladders: One more Step on Generalizing the Montgomery Ladder

Yoann Marquer, Tania Richmond, Pascal Véron

Interleaved Ladders: One more Step on Generalizing the Montgomery Ladder

Yoann Marquer[1,4], Tania Richmond[2,4], and Pascal Véron[3]

[1]SnT, SVV, University of Luxembourg, Luxembourg, yoann.marquer@uni.lu
[2]Institut de Sciences Exactes et Appliquées, Université de la Nouvelle-Calédonie, France, tania.richmond@unc.nc
[3]Laboratoire IMath, Université de Toulon, France, pascal.veron@univ-tln.fr
[4]Most of this work was done while Y. Marquer and T. Richmond were affiliated with Inria, Univ. Rennes, CNRS, IRISA, France

Abstract

Iterative conditional branchings appear in various sensitive algorithms, like the modular exponentiation in the RSA cryptosystem or the scalar multiplication in elliptic-curve cryptography. The Montgomery ladder is a common example of such algorithm with desirable security properties against some side-channel and fault-injection attacks. In this paper, we abstract away and generalize these security properties by formalizing, using systems of equations, what we call semi-interleaved and fully-interleaved ladders. This fruitful approach allows us to design novel fault-injection attacks, able to obtain some/all bits of the secret against different ladders, breaking the common Montgomery ladder. We also demonstrate the generality of our approach by applying our ladder equations to the modular exponentiation and the scalar multiplication, both in the semi- and fully-interleaved cases, to propose novel and more secure algorithms.

1 Introduction

1.1 Context

An *iterative conditional branching* is a loop containing a conditional branching. Such iterative conditional branching is said *sensitive* if the condition of the conditional branching depends on a sensitive value, like the value of the secret key in a cryptosystem. Sensitive iterative conditional branchings are frequent (Subsection 3.1) and, unfortunately, usually prone to vulnerability to side-channel (Subsection 2.1) or fault-injection (Subsection 2.2) attacks.

The square-and-multiply (Algorithm 1 p.4) and the Montgomery ladder (Algorithm 3 p.6) are common examples of sensitive iterative conditional branchings. As opposed to the square-and-multiply algorithm, the Montgomery ladder demonstrates desirable security properties against side-channel and fault-injection attacks (Subsection 2.3). We call *ladderization* the refactoring process transforming, for instance, the vulnerable square-and-multiply algorithm into the more protected Montgomery ladder.

1.2 Contribution

The purpose of this paper is to solve the computational problem of ladderizing sensitive iterative conditional branchings in order to reduce their security vulnerabilities. This is done by abstracting away the desirable security properties from the Montgomery ladder and by generalizing ladderization to a more general class: the ladderizable algorithms obtained in Theorem 1 p.10.

To our knowledge, this ladderization approach is the first systematic attempt to generalize these security properties, by using systems of equations called the *ladder equations*. While investigating this novel problem, we distinguish between *semi-*

interleaved ladders (Subsection 3.2), including the usual Montgomery ladder, and *fully-interleaved ladders* (Subsection 3.3) with a more complex pattern.

Based on the code symmetry of the semi- and fully-interleaved ladders, the same operations are performed regardless the sensitive value, so these algorithms are secure against simple side-channel attacks depending on these operations, like timing side-channel attacks [32] or SPA (Simple Power Analysis) [31].

Based on fault-propagation patterns that prevent an attacker to compare outputs with or without fault injection (Section 4), we demonstrate in Table 1 p.14 that fully-interleaved ladders are more secure against fault-injection attacks than semi-interleaved ladders, which are more secure than algorithms without interleaving. This fruitful investigation leads to the discovery of novel fault-injection attacks applicable to the common Montgomery ladder (Section 4). Moreover, a violation of the ladder invariant $y = \ell(x)$ between intermediate variables x and y allows the protected system to detect a fault-injection, and thus to trigger countermeasures (Subsection 2.3).

Solving the ladder equations was feasible by a solver for the semi-interleaved ladders with `if-then` branchings (i.e., without `else` branch). Thus, the ladderization refactoring was semi-automatized in this simple case [11]. But, because of the complexity of the ladder equations in the general case, we focus in this paper on manually solving the ladder equations for common cryptosystems.

We validate our approach and demonstrate its generality by obtaining novel and more secure algorithms for the modular exponentiation in the RSA [45] and DSA [40] cryptosystems (Section 5), and for the scalar multiplication in elliptic-curve cryptography [5, 40] (Section 6). In most of the cases, solving the ladder equations leads to a degree of freedom, that we exploit by blinding intermediate computations using a randomly generated integer, in order to prevent advanced side-channel attacks like CPA (Correlation Power Analysis) [10]. We also discuss the feasibility of our solutions. Tables 2 and 4 summarize their cost compared to the Montgomery ladder, demonstrating a trade-off between security and performance.

The ladder equations, the first two fault-injection attacks and the solutions for the modular exponentiation have already been published [37], but they are included here so that the paper is entirely self-contained. This paper extends and complete these results with the following contributions:

- a third fault-injection attack (for the second attacker model) against all (including fully-interleaved) ladders (Subsection 4.2),
- in the case of the modular exponentiation, a) more performant algorithms for the solutions than initially published [37] and b) a proof that the fully-interleaved ladder solution is applicable, more precisely that the probability to randomly obtain a suitable ladder constant (which was an open problem in [37]) is almost 1 when used in concrete applications like RSA or DSA cryptosystems (Subsections 5.5 and 5.6),
- and novel semi- and fully-interleaved ladder algorithms for the scalar multiplication over elliptic curves (Section 6).

1.3 Organization of the Paper

We introduce in Section 2 related works on side-channel and fault-injection attacks, then we use the case of the modular exponentiation to introduce security concerns and we present the Montgomery ladder both the in this case and in the case of scalar multiplication in elliptic-curve cryptography. We conclude this section by introducing related works on parallel computation, in particular in the context of elliptic-curve cryptography. In Section 3, we formalize the iterative conditional branchings to deduce the equations satisfied by the semi-interleaved and fully-interleaved ladders, and thus the requirements for ladderizable algorithms. We state our research questions in Subsection 3.5. In Section 4, we introduce two attacker models using fault-injection techniques, and we compare the vulnerability of the non-, semi- and fully-interleaved ladders to these attacks. We then detail how to produce examples of the semi- and fully-interleaved ladders in Section 5 for the modular exponentiation and in Section 6 for the scalar multiplication. Subsections 5.7 and 6.6 answer the research questions, then Section 7 concludes.

2 Related Works

In this section, we first introduce context on side-channel (Subsection 2.1) and fault-injection attacks (Subsection 2.2). Then, we explain step-by-step in Subsection 2.3 the rationale behind known algorithms for the modular exponentiation, and their respective relevance regarding desirable security properties. In particular, we introduce in this subsection the Montgomery ladder for modular exponentiation, then its variant in Subsection 2.4 for scalar multiplication. Finally, we briefly describe in Subsection 2.5 works aiming at improving performance using parallel computations, to better compare our solutions with the Montgomery ladder.

2.1 Side-Channel Attacks

A *side-channel* is a way of transmitting information (purposely or not) to another system out of the intended communication channels. Side-channel attacks rely on the relationship between information leaked through a side-channel and the secret data to obtain confidential information. In particular, they can be used to break cryptographic protocols by exploiting information that is observed during execution of the implementation of an algorithm.

The ability to extract information on the secret key of a cryptosystem through side-channel observations has been achieved in practice using:

- *Timing side-channel attacks* [32] rely on the correlation between execution time and the sensitive information. For instance, we illustrate in Subsection 2.3 how timing attacks can be used to obtain the number of 1s in the binary representation of a secret key.
- *Simple Power Analysis* (SPA) [31] consists in observing the power profile of a chip during execution to retrieve sensitive information. For instance, in Figure 1 it is possible to read directly on the screen of an oscilloscope the first bits of the secret key.
- More advanced techniques can be applied to a power side-channel, like *Correlation Power Analysis* (CPA) [10] based on the linear correlation between the number of bits flipping at an iteration and the evolution of the power consumption.

Countermeasures against simple side-channel attacks like timing attacks or SPA involve *code symmetry*: if the same operations are computed in any case then the attacker cannot leverage execution time or a single power trace to gain sensitive information.

But code symmetry is not sufficient against more advanced attacks like CPA. Against such attacks, a more advanced countermeasure like *blinding* is required. The idea is to hide the correlation between intermediate values and the sensitive information by using a random value during intermediate computations.

For instance, in the context of the RSA [45] cryptosystem, let k be the private key and e be the public key such that, for every integer x, $x^{ke} = x \bmod n$. A message m is encrypted using the public key e into a ciphertext $c = m^e \bmod n$ and publicly sent to the chip. Then, without countermeasure, the chip uses the private key k to decrypt the ciphertext into the original message $c^k = m^{ke} = m \bmod n$. But in the case of a blinding countermeasure the chip picks a random integer r invertible modulo n and decrypt cr^e instead of c. Indeed, $(cr^e)^k = (mr)^{ke} = mr \bmod n$. Then, because r is invertible, from mr the message m can be computed by the chip. So, during the computation of $(cr^e)^k$, the intermediate values depend on the random integer r, and the attacker is not able to distinguish bit flips due to the message or due to the random variable. In this article, we call such a random integer used in blinding techniques a *blinding integer*.

2.2 Fault Injection Attacks

A *fault* is a physical defect, imperfection or flaw that occurs within some hardware or software component, while an *error* is a deviation from accuracy or correctness, and is the manifestation of a fault [50]. Hardware (also called physical) faults can be permanent, transient or intermittent, while software faults are the consequence of incorrect design, at specification or at coding time. *Fault injection* is defined as the validation technique of the dependability of fault tolerant systems, which consists in performing controlled experiments where the observation of the system's behavior in presence of faults is induced explicitly by the writing introduction (called injection) of faults in the sys-

tem [2]. Fault injection techniques can be grouped into *invasive* and *non-invasive* techniques, the latter being able to hide their presence so as to have no effect on the system other than the faults they inject [50].

We illustrate in Subsection 2.3 a non-invasive technique called *safe-error attack* [24], which consists in 1) running the source code with a given input to get the corresponding output, then 2) run it again with the same input but while introducing a transient fault when the execution reaches a particular location to get another output, and finally 3) compare both outputs to see if the fault injection altered the execution, and thus retrieve some information about the execution.

In this paper, we do not assume a specific mean (hardware, software, simulation, emulation, etc.) for fault injection and we consider only two types of fault injections: *random fault* (the affected variable is set to a random value) and *zeroing/one-ing fault* (all bits are set to 0 or 1). We detail attacker models and fault-propagation patterns in Section 4.

2.3 Modular Exponentiation

Modular exponentiation algorithms compute $a^k \bmod n$, where k is a secret key and a, n are public values. They are commonly used in cryptosystems like RSA [45], and we use them as running example. We denote by $k = \sum_{0 \leq i \leq d} k[i]\, 2^i$ the binary expansion of size $d+1$ of the secret key, where $k[i]$ is the bit i of k.

The *square-and-multiply* Algorithm 1 computes the left-to-right modular exponentiation using $a^{\sum_{0 \leq i \leq d} k[i]2^i} = \prod_{0 \leq i \leq d} (a^{2^i})^{k[i]}$. For every iteration, the squaring x^2 is computed at Line 3 in every case while the multiplication ax is computed at Line 5 only if the current secret bit is $k[i] = 1$. This dependency on the secret key can be detected by observing side-channels.

Observing the execution time [32] leaks the number of multiplications performed during the execution, hence the Hamming weight (i.e., the number of non-zero bits) of the secret key. This narrows down the search space, compared to brute-force attacks.

Observing the power profile during the execution leaks even more information than observing execution time, as demonstrated in Simple Power Analysis (SPA) [31]. We illustrate in Figure 1 a profile obtained in the screen of an oscilloscope. The x-axis corresponds to timestamps, while the y-axis corresponds to positions in the screen of the oscilloscope, linearly correlated with the differential voltage from a baseline, measured on the chip during execution. An iteration requires only one operation if the secret bit is $k[i] = 0$ and two operations[1] if the secret bit is $k[i] = 1$. Hence, based on the length of each period of power consumption, an attacker can read the first bits of the secret key on the screen of the oscilloscope[2].

To prevent SPA, regularity of the modular exponentiation algorithms is required, which means that both branches of the sensitive conditional bran-

Algorithm 1 Square and Multiply

input public a, n; secret k

1: $x \leftarrow 1$
2: **for** $i = d$ **to** 0 **do**
3: $\quad x \leftarrow x^2 \bmod n$
4: $\quad$ **if** $k[i] = 1$ **then**
5: $\quad\quad x \leftarrow ax \bmod n$
6: $\quad$ **end if**
7: **end for**
8: **return** x

output $x = a^k \bmod n$

Algorithm 2 Square and Multiply Always

input public a, n; secret k

1: $x \leftarrow 1$
2: **for** $i = d$ **to** 0 **do**
3: $\quad x \leftarrow x^2 \bmod n$
4: $\quad$ **if** $k[i] = 1$ **then**
5: $\quad\quad x \leftarrow ax \bmod n$
6: $\quad$ **else**
7: $\quad\quad y \leftarrow ax \bmod n$
8: $\quad$ **end if**
9: **end for**
10: **return** x

output $x = a^k \bmod n$

[1] Actually, a multiplication can be distinguished from a squaring, so variants with squaring only have been proposed in [12] and improved in [20], but this is out of the scope of this paper.

[2] To improve readability of the plot, we padded the end of iterations so that each one has the same execution time, making easier to distinguish long from shot periods of power consumption.

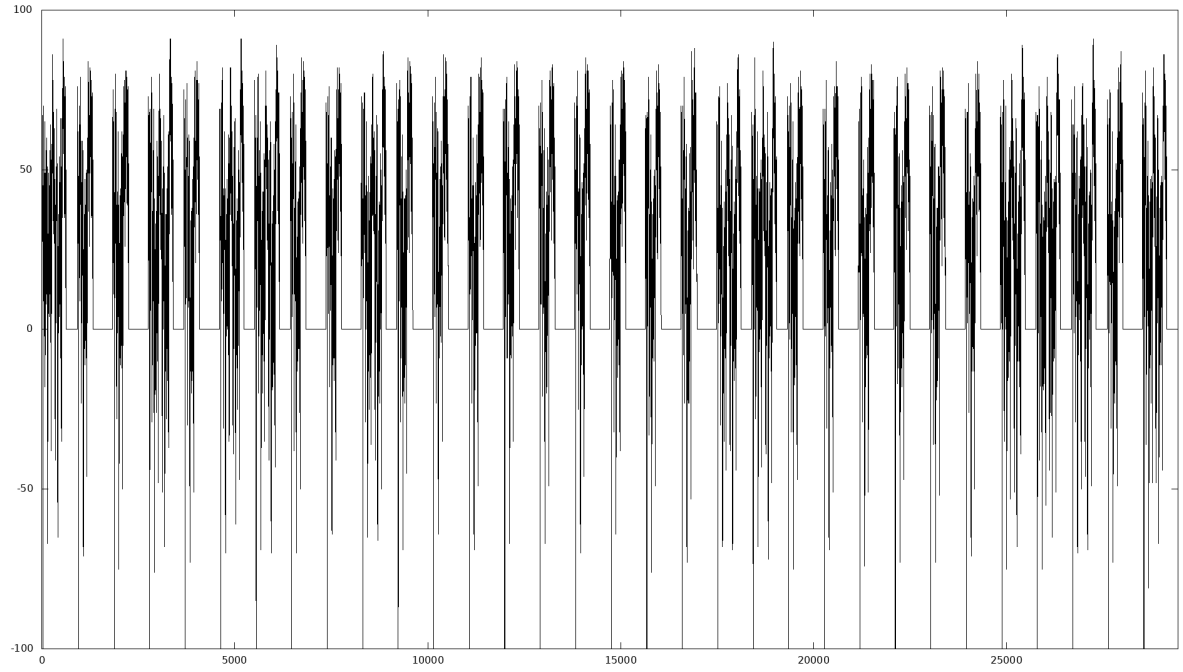

Figure 1: Power profile for the first 32 iterations of Algorithm 1. Based on the length of periods of power consumption (long for 1 and short for 0), an attacker can read the 32 first bits 10010110010000000001100000011101 of the secret key on the screen of the oscilloscope.

ching perform the same operations, independently from the value of the secret key. Thus, an obvious solution is to add an `else` branch containing a dummy instruction [14] at Line 7 of the *square-and-multiply-always* Algorithm 2.

But power profiles also depend on values in the considered registers, so computing a multiplication in every case is better against SPA but not against more advanced side-channel attacks like CPA [10]. Fortunately, standard blinding techniques can prevent differential [14, 38] or correlation power analysis [10]. We use such a blinding technique in Subsection 5.2 and Subsection 6.2 to improve security in most solutions.

Moreover, countermeasures developed against a given attack may benefit another one [48]. The multiplication in the `else` branch of the square-and-multiply-always algorithm is a dummy operation in the sense that if the current secret bit is $k[i] = 0$ then the result is not used by the rest of the computation. Thus, a random fault injected [50] in the register containing ax will propagate through successive iterations and alter the final result only if the current secret bit is $k[i] = 1$. So, an attacker (see the attacker models in Section 4) able to inject a fault in a register at a given iteration can apply such a process called a *safe-error attack* to obtain any digit of the secret key by comparing the final output with or without fault.

Such simple safe-error attack cannot happen in the case of the Algorithm 3 proposed by Montgomery [33], where a fault injected in a register will eventually propagate to the other one. Thus, the fault alters the final result in any case, preventing the attacker to obtain information. But, as we detail in Section 4, some information on the last digits may still leak, weakening the protection obtained from this ladder.

The Montgomery ladder is algorithmically equivalent [36] to the square-and-multiply(-always) algorithm(s), in the sense that x has the same value for every iteration. Actually, some variants [9] of the

Algorithm 3 Montgomery Ladder for the Modular Exponentiation

input public a, n; secret k
1: $x \leftarrow 1$
2: $y \leftarrow a \bmod n$
3: **for** $i = d$ **to** 0 **do**
4: **if** $k[i] = 1$ **then**
5: $x \leftarrow xy \bmod n$
6: $y \leftarrow y^2 \bmod n$
7: **else**
8: $y \leftarrow xy \bmod n$
9: $x \leftarrow x^2 \bmod n$
10: **end if**
11: **end for**
12: **return** x
output $x = a^k \bmod n$

square-and-multiply-always algorithm[3] may be as resistant as the Montgomery ladder [24], by checking invariants [28] violated if a fault is injected. For instance, in the case of the Montgomery ladder, the invariant $y = ax$ is satisfied for every iteration. These invariants are important because their violation in case of fault injection can be detected by the algorithm itself, which then is able to trigger self-secure countermeasures [18], as discussed briefly in Subsection 4.3.

Note that the `else` branch in Algorithm 3 is identical to the `then` branch except that x and y are swapped which, because of code symmetry, provides also (partial[4]) protection against timing and power leakage. Moreover, the variable dependency makes these variables interleaved, so this exponentiation is algorithmically (but partially, as we demonstrate in Section 4) protected against safe-error attacks. Finally, as opposed to square-and-multiply-always in Algorithm 2, the code in the `else` branch of Algorithm 3 is not dead, so will not be removed by compiler optimizations.

Thus, such algorithmic properties provide some protection against side-channel and fault-injection attacks, and are therefore security properties which are desirable in more algorithms. We formalize these properties in Section 3, then we propose secure algorithms for the modular exponentiation in Section 5.

2.4 Elliptic-curve cryptography (ECC)

ECC was independently introduced in 1985 by Victor Miller [39] and Neal Koblitz [30]. It is nowadays considered as an excellent choice for key exchange or digital signatures, especially when these mechanisms run on resource-constrained devices. The security of most cryptocurrencies is based on ECC, which has been standardized by the NIST [5, 40].

Definition 1 (Elliptic Curve)**.** Let p be a prime. An *elliptic curve* in short Weierstrass form over a finite field $\mathbb{F}_p$ is the set $E(\mathbb{F}_p) = \left\{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p \mid y^2 = x^3 + ax + b\right\} \cup \{\mathcal{O}\}$ with parameters $a, b \in \mathbb{F}_p$ satisfying $4a^3 + 27b^2 \neq 0$ and $\mathcal{O}$ being called the point at infinity.

The set $E(\mathbb{F}_p)$ is an additive Abelian group with an efficiently computable group law. The point $\mathcal{O}$ is the identity element of the group law. Point addition $P + Q$ or point doubling $2P$ involves additions and multiplications over $\mathbb{F}_p$. Depending on the parameters a and b of the curve, there exists many formulas[5] to optimize these two operations.

The main operation in ECC is scalar multiplication $kA = A + \cdots + A$, where A is a point on a curve and k is an integer. It can be performed by using the *double-and-add* Algorithm 4, similar to the square-and-multiply Algorithm 2. The initialization $x \leftarrow 1$ in square-and-multiply is replaced by $P \leftarrow \mathcal{O}$[6] in double-and-add, the squaring $x \leftarrow x^2$ is replaced by a doubling $P \leftarrow 2P$ and the multiplication $x \leftarrow ax$ is replaced by an addition $P \leftarrow A + P$, producing the output $P = kA$.

As the double-and-add Algorithm 4 for the scalar multiplication is the counterpart of the square-and-

[3] See [22] for highly regular right-to-left variants, [23] for a generalization to any basis and left-to-right/right-to-left variants, and [49] for their duality.

[4] For instance, the variables x and y may have different access time, which would hinder protection against cache-timing leakage.

[5] A large and updated survey on the elliptic curves explicit formulas database web site [46].

[6] Definition 1 corresponds to the common representation of the elliptic curves in affine coordinates, but the point $\mathcal{O}$ at infinity cannot be represented in this coordinate system. Nevertheless, for efficiency reasons, operations on elliptic curves are computed in practice using projective coordinates [13], where $\mathcal{O}$ can be represented. Hence, Algorithm 4 is practical. Describing the various projective systems is out of the scope of this paper, hence we presented only the affine coordinates for the sake of simplicity.

Algorithm 4 Double-and-Add

input public A; secret k

1: $P \leftarrow \mathcal{O}$
2: **for** $i = d$ **to** 0 **do**
3: $\quad P \leftarrow 2P$
4: $\quad$ **if** $k[i] = 1$ **then**
5: $\quad\quad P \leftarrow A + P$
6: $\quad$ **end if**
7: **end for**
8: **return** P

output $P = kA$

Algorithm 5 Montgomery Ladder for the Scalar Multiplication

input public A; secret k

1: $P \leftarrow \mathcal{O}$
2: $Q \leftarrow A$
3: **for** $i = d$ **to** 0 **do**
4: $\quad$ **if** $k[i] = 1$ **then**
5: $\quad\quad P \leftarrow P + Q$
6: $\quad\quad Q \leftarrow 2Q$
7: $\quad$ **else**
8: $\quad\quad Q \leftarrow Q + P$
9: $\quad\quad P \leftarrow 2P$
10: $\quad$ **end if**
11: **end for**
12: **return** P

output $P = kA$

multiply Algorithm 1 for the modular exponentiation, a variant of the Montgomery ladder is also available in Algorithm 5 for the scalar multiplication, providing another example of ladderization.

We propose secure algorithms for the scalar multiplication in Section 6.

2.5 Parallel Computations

Finally, we compare the secure algorithms we obtain in this paper with the Montgomery ladder, respectively in Section 5 for the modular exponentiation and in Section 6 for the scalar multiplication. Restricting such a comparison in terms of performance to sequential computations only would not reflect the current state of the art.

Indeed, parallel implementations of algorithms on multicore systems was investigated for scalar multiplication [25, 15]. Parallelism was leveraged in Graphical Processing Units (GPUs) to drastically increase the number of operations per second performed for the elliptic-curve factorization method [7]. Moreover, efficient elliptic curve point multiplications were obtained using an algorithm where both the pre-computation and post-computation stages are performed in parallel, on machines with up to eight cores [6]. Finally, leveraging an approach based on the residue number system, a parallel algorithm was proposed for elliptic curve point multiplication, efficient on more than four cores [1].

Therefore, when comparing our solutions to the Montgomery ladder in Subsections 5.4 and 6.5, we consider both sequential and parallel computations.

3 Ladder Equations

In this section, we formalize the desirable security properties occurring in algorithms like the Montgomery ladder (Algorithm 3) by using systems of equations we call *ladder equations*. These constraints are sufficient to refactor iterative conditional branchings (Subsection 3.1) as in the square-and-multiply algorithm (Algorithm 1 p.4) into semi- (Subsection 3.2) or fully-interleaved (Subsection 3.3) ladders, to make them more secure. The algorithms satisfying these constraints are called *ladderizable* and are characterized in Theorem 1 (Subsection 3.4). Finally, in Subsection 3.5 we state our research questions and we provide first answers.

3.1 Iterative Conditional Branchings

In this paper, we focus on code patterns like Algorithm 6, which consists in a conditional branching in (potentially nested) loop branching(s) and which we call an *iterative conditional branching*. Moreover, if the condition involves a sensitive value, i.e., which depends on confidential information, we call it a *sensitive iterative conditional branching*. Iterative conditional branching appear in algorithms like the square-and-multiply Algorithm 1 p.4 for the modular exponentiation, its counterpart the double-and-add Algorithm 4 p.7 for the elliptic curve point multiplication [39, 30], or the secure bit permutation in the McEliece cryptosystem [47],

Algorithm 6 Iterative Conditional Branching

1: $x \leftarrow \text{init}$
2: **for** $i = 1$ **to** n **do**
3: $\quad \ddots$
4: $\quad$ **if** secret **then**
5: $\quad\quad x \leftarrow \theta(x)$
6: $\quad$ **else**
7: $\quad\quad x \leftarrow \varepsilon(x)$
8: $\quad$ **end if**
9: $\quad \ddots$
10: **end for**

Algorithm 7 Loop bounded by a sensitive variable

1: assert(secret $\leq$ bound)
2: **for** $i = 0$ **to** secret **do**
3: $\quad \vdots$
4: **end for**

attacked in [43]. It also appears naturally when one tries to turn a loop on sensitive information as in Algorithm 7 into a safe loop containing a sensitive conditional branching as in Algorithm 8, that can itself be balanced to obfuscate the dependency on the secret. So, iterative conditional branchings are frequent in cryptosystems. Hence, making them at least as secure as the Montgomery ladder is desirable.

Note that our approach does not depend on the number or depth of the considered loops, hence the dots in Algorithm 6. We assume only that the conditional branching uses only one variable x, the multivariate case being left for future work.

Definition 2 (Iterative Conditional Branching)**.** An algorithm like Algorithm 6 is said with a (univariate) *iterative conditional branching* with two (unary) functions θ and ε.

3.2 Semi-Interleaved Ladders

To prevent information leakage from side-channel (Subsection 2.1) or fault injection (Subsection 2.2) attacks, we use a fresh variable y in the *semi-interleaved ladder* presented in Algorithm 9. As in the Montgomery ladder in Algorithm 3 p.6, we look for two functions ℓ and f such that for each iteration:

- the invariant $y = \ell(x)$ holds, and

Algorithm 8 Equivalent iterative conditional branching

1: **for** $i = 0$ **to** bound **do**
2: $\quad$ **if** $i \leq$ secret **then**
3: $\quad\quad \vdots$
4: $\quad$ **end if**
5: **end for**

Algorithm 9 Semi-Interleaved ladders

1: $x \leftarrow \text{init}$
2: $y \leftarrow \ell(\text{init})$
3: **for** $i = 1$ **to** n **do**
4: $\quad \ldots$
5: $\quad$ **if** secret **then**
6: $\quad\quad x \leftarrow f(x, y)$
7: $\quad\quad y \leftarrow \varepsilon(y)$
8: $\quad$ **else**
9: $\quad\quad y \leftarrow f(y, x)$
10: $\quad\quad x \leftarrow \varepsilon(x)$
11: $\quad$ **end if**
12: $\quad \ldots$
13: **end for**

- x has the same value for every iteration as in Algorithm 6.

the Montgomery ladder being a particular case, where $\ell(x) = ax$ and, depending on the secret bit, x is updated by either $\theta(x) = ax^2$ or $\varepsilon(x) = x^2$.

We determine by induction the constraints that ℓ and f should satisfy. In Algorithm 9, $y = \ell(x)$ holds at the initialization. Then, we assume by induction that $y = \ell(x)$ holds at the beginning of an iteration.

In the `then` branch[7] we have $x \leftarrow f(x, y)$ then $y \leftarrow \varepsilon(y)$, thus in order to satisfy the invariant $y = \ell(x)$ at the end of an iteration, the following equation must hold:

$$\forall x, \varepsilon(\ell(x)) = \ell(f(x, \ell(x)))$$

where x is quantified over the considered values e.g., modular integers for the modular exponentiation or points in elliptic-curve cryptography. In order to

[7]The condition depends on the secret, hence exploiting the condition to determine ℓ and f could lead to data dependencies that could be attacked. Thus, we do not assume in the `then` branch that the condition is satisfied and we do not assume in the `else` branch that the negation of the condition is satisfied.

Algorithm 10 Fully-Interleaved ladders

```
1: x ← init
2: y ← ℓ(init)
3: for i = 1 to n do
4:     ...
5:     if secret then
6:         x ← f(x, y)
7:         y ← g(x, y)
8:     else
9:         y ← f(y, x)
10:        x ← g(y, x)
11:    end if
12:    ...
13: end for
```

have x updated to $\theta(x)$ during the iteration, the following equation must hold:

$$\forall x, f(x, \ell(x)) = \theta(x)$$

In the `else` branch we have $y \leftarrow f(y, x)$ then $x \leftarrow \varepsilon(x)$, thus in order to satisfy $y = \ell(x)$ at the end of an iteration the following equation must hold:

$$\forall x, f(\ell(x), x) = \ell(\varepsilon(x))$$

and x is already updated to $\varepsilon(x)$ during the iteration. Therefore, we obtained the *ladder equations* for the semi-interleaved ladders:

Definition 3 (Semi-Ladderizable)**.** Let A be an algorithm with a univariate iterative conditional branching with two unary functions denoted θ and ε. A is *semi-ladderizable* if there exists a unary function ℓ and a binary function f such that, for every considered input value x:

$$\begin{cases} \varepsilon(\ell(x)) = \ell(\theta(x)) & \text{(1a)} \\ f(x, \ell(x)) = \theta(x) & \text{(1b)} \\ f(\ell(x), x) = \ell(\varepsilon(x)) & \text{(1c)} \end{cases}$$

For instance, in the square-and-multiply Algorithm 1, we have $\theta(x) = ax^2$ and $\varepsilon(x) = x^2$, and we know that this iterative conditional branching can be semi-ladderized by using the Montgomery ladder Algorithm 3 with $\ell(x) = ax$ and $f(x, y) = xy$, but we demonstrate in Section 5 that there are other solutions.

Note that, to respect the form of the semi-interleaved ladder, we should have written $y \leftarrow yx$ instead of $y \leftarrow xy$ in the `else` branch of the Montgomery ladder Algorithm 3. The former is actually better regarding vulnerability to the M safe-error [24] or collision [27] attacks, demonstrating that the ladderization approach provides good practice.

Note also that the invariant function $\ell(x) = ax$ of the Montgomery ladder can directly be read in the `then` branch of Algorithm 1. This was leveraged in a previous study [11] to automatically infer $f(x, y) = xy$ and thus to automatically refactor the code to make it more secure. But this was done only in the simplest case of the semi-interleaved ladders with `if-then` branchings (so without `else` branch). In this paper we address the general case by manually solving these ladders equations for common cryptographic examples: the modular exponentiation in the RSA and DSA cryptosystems in Section 5, and the scalar multiplication in elliptic-curve cryptography in Section 6.

3.3 Fully-Interleaved Ladders

Unfortunately, the semi-interleaved ladder (Algorithm 9) is vulnerable to fault injection attacks, as we demonstrate in Section 4, because in every branch at least one variable (x or y) depends only on its previous value and not on the previous value of both variables. Moreover, an attacker able to determine whether the output of one operation is used as the input to another can apply collision attacks[8] to infer whether two following bits are the same [21].

To tackle these challenges, we propose in Algorithm 10 a *fully-interleaved ladder* using three functions ℓ, f, and g. As in the previous subsection, we look for functions satisfying that for each iteration:

- the invariant $y = \ell(x)$ holds, and
- x has the same value for every iteration as in Algorithm 6.

Similarly to the semi-interleaved ladders, we determine by induction the constraints that the functions ℓ, f, and g have to satisfy. Again, $y = \ell(x)$ is satisfied at the initialization. Then, we assume

[8] A possible countermeasure is to randomly blend variants of the ladder, or compute the exponentiation by taking a random (bounded) walk [35].

by induction that $y = \ell(x)$ at the beginning of an iteration.

In the `then` branch we have $x \leftarrow f(x, y)$ then $y \leftarrow g(x, y)$, thus in order to have $y = \ell(x)$ satisfied at the end of any iteration, the following equation must hold:

$$\forall x, g(f(x, \ell(x)), \ell(x)) = \ell(f(x, \ell(x)))$$

and in order to have x updated to $\theta(x)$ during the iteration, the following equation must hold:

$$\forall x, f(x, \ell(x)) = \theta(x)$$

In the `else` branch we have $y \leftarrow f(y, x)$ then $x \leftarrow g(y, x)$, thus, in order to have $y = \ell(x)$ satisfied at the end of an iteration, the following equation must hold:

$$\forall x, f(\ell(x), x) = \ell(g(f(\ell(x), x), x))$$

and in order to have x updated to $\varepsilon(x)$ during the iteration, the following equation must hold:

$$\forall x, g(f(\ell(x), x), x) = \varepsilon(x)$$

Therefore, we obtain the *ladder equations* for the fully-interleaved ladders:

Definition 4 (Fully-Ladderizable)**.** Let A be an algorithm with a univariate iterative conditional branching with two unary functions denoted θ and ε. A is *fully-ladderizable* if there exists a unary function ℓ and two binary functions f and g such that, for every considered input value x:

$$\begin{cases} g(\theta(x), \ell(x)) = \ell(\theta(x)) & \text{(2a)} \\ f(x, \ell(x)) = \theta(x) & \text{(2b)} \\ f(\ell(x), x) = \ell(\varepsilon(x)) & \text{(2c)} \\ g(f(\ell(x), x), x) = \varepsilon(x) & \text{(2d)} \end{cases}$$

Note that Equation (1b) is Equation (2b) and Equation (1c) is Equation (2c). Without surprise, if g is chosen such that $g(x, y) = \varepsilon(y)$ then Equation (1a) is a special case of Equation (2a), and Equation (2d) is satisfied. Thus, semi-interleaved ladders are subcases of fully-interleaved ladders for $g(x, y) = \varepsilon(y)$.

3.4 Ladderizable Algorithms

From Subsections 3.2 and 3.3 we obtained ladder equations, sufficient to guarantee that, for each iteration, the invariant holds and the considered variable x is correctly updated. We use these constraints to define the class of the *ladderizable* algorithms, i.e., the algorithms that can be refactored to be more secure using ladderization:

Theorem 1. *Let A be an algorithm with an iterative conditional branching with two unary functions denoted θ and ε. If A is semi-ladderizable with ℓ and f, or fully-ladderizable with ℓ, f and g, then for every iteration of the ladderized variant:*

- $y = \ell(x)$
- *x is updated as in A and takes on the value:*

$$x \leftarrow \begin{cases} \theta(x) & \text{if secret} \\ \varepsilon(x) & \text{otherwise} \end{cases}$$

Thus, for every algorithm with an iterative conditional branching, as in Algorithm 6, if there exists ℓ and f satisfying the ladder equations in Definition 3 then the conditional branching can be refactored as in Algorithm 9 as a semi-interleaved ladder. Even better, if there exists ℓ, f and g satisfying the ladder equations in Definition 4 then it can be refactored as in Algorithm 10 as a fully-interleaved ladder. In any case, the obtained algorithm is algorithmically equivalent [36] in the sense that for every iteration the considered variable x has the same value for both original and ladderized variants.

3.5 Research Questions

In this paper we aim to answer the following research questions regarding the ladderization approach:

- **RQ1**: To which extent can the algorithmic strength of the Montgomery ladder be generalized to other algorithms?
- **RQ2**: How secure are interleaved ladders compared to the Montgomery ladder?
- **RQ3**: How feasible are interleaved ladders in practice and, if so, how performant are they compared to the Montgomery ladder?

Theorem 1 provides a theoretical answer to **RQ1**. We abstracted away the Montgomery ladder to the largest class of algorithms possible while preserving, through the ladder equations, its desirable security properties.

The protection against side-channel attacks comes from the code symmetry. In semi- or fully-interleaved ladders the `else` branch is identical to the `then` branch, except that x and y are swapped. So, the same operations are performed regardless the secret value, and both ladder types are secure against side-channel attacks depending on these operations, like simple timing attacks [32] or SPA [31]. Therefore, the ladderization refactoring is an algorithmic countermeasure against these side-channel attacks. Moreover, Subsections 5.2 and 6.2 demonstrate solutions with blinding integers which are randomly generated and thus provide protection against more complex side-channel attacks like CPA [10].

The protection against fault-injection attacks is investigated in Section 4. Then, to provide a more concrete answer to **RQ1**, we construct examples of semi- and fully-interleaved ladders in Section 5 for the modular exponentiation and in Section 6 for the scalar multiplication.

4 Vulnerability against Fault Injection Attacks

To investigate the vulnerability protection of the interleaved ladders (Section 3), we introduce in Subsections 4.1 and 4.2 two attacker models using fault injection techniques. As described in the background (Subsection 2.2), we consider only two types of fault injections: *random fault* (the affected variable is set to a random value) and *zeroing/one-ing fault* (some/all bits are set to 0 or 1). Injecting a random fault requires dedicated material and knowledge, and zeroing/one-ing bits is even more costly. But the feasibility of these (attacker) techniques does not impact the feasibility of the (defender) ladderization approach (**RQ3**), hence we do not consider in this paper the cost for the attacker.

We describe the fault-propagation patterns of the non-, semi- and fully-interleaved ladders in Subsubsection 4.1.1, then we describe the first attack in Subsubsection 4.1.2, the second attack in Subsubsection 4.2.1, and the third attack in Subsubsection 4.2.2. Finally, we compare the vulnerability of the non-, semi- and fully-interleaved ladders in Subsection 4.3 and we provide there more answers to our research questions.

4.1 First Attacker Model

For ease of presentation, we assume as in the Montgomery ladder (Algorithm 3) that "secret" in the iterative conditional branching (Algorithm 6) and the semi- and fully-interleaved ladders (Algorithms 9 and 10) is the condition $k[i] = 1$, where $k[i]$ is the bit i of the secret key k. But the approach would be similar for any other sensitive condition.

We start by the first attacker model, which uses only random faults.

Definition 5 (First Attacker Model)**.** We assume that:

- The attacker wants to obtain the secret key stored in the chip and copied in the register k.
- The attacker can run the program any number of times:
  - inputting x_{init} and y_{init}, the initial values in the registers x and y,
  - obtaining x_{final} and/or y_{final}, the final value(s) returned by the program.
- A run consists of iterations i over[9] $1, \ldots, n$, where:
  - $k[i]$ is the bit i of k,
  - x_i (resp. y_i) denotes the value in the register x (resp. y) between iterations[10] $i-1$ and i.
- The attacker can $\lightning x_i$ (resp. $\lightning y_i$) inject a random fault (i.e., set the affected variable to a random value) in the register x (resp. y) between iterations $i-1$ and i.

[9] To simplify the notations, we assume in this section that the counter is incremented (from 1 to n with a step 1), but the approach is similar for other initial or final values, a decremented counter, and/or other step values.

[10] Iteration 0 being the initialization.

After injecting a random fault in a register, the attacker might be unlucky and obtain the same value as before in this register. But this is unlikely (especially considering the size of modern cryptographic keys and messages) and can be mitigated by performing several tries. So, we assume for the sake of simplicity that the new value is always different. We also assume that this new value leads to a different final value. Otherwise, the attacker should try different x_{init} and y_{init} inputs.

4.1.1 Fault-Propagation Patterns

As described in Subsection 2.3, the attacker can perform a *safe-error attack*:

- run a program for the square-and-multiply(-always) Algorithm 1 (resp. Algorithm 2) for a given input x_{init}, obtaining a value x_{final},
- then run the program again with the same input while injecting a fault $\lightning y_i$ in the register $y = ax$ between iterations $i-1$ and i, obtaining a value x_{fault}.

If $x_{\text{fault}} = x_{\text{final}}$ then $k[i] = 0$, otherwise $k[i] = 1$. Hence, this safe-error attack leaks the value of this secret bit. The same attack can be repeated for every iteration, thus the attacker can obtain that way all the bits of the secret key.

In the square-and-multiply(-always) algorithm, because the current value of x determines the next value of x, a faulted value $\lightning x_i$ for an iteration i always propagates to the next iteration $\lightning x_{i+1}$, which we denote by $\lightning x_i \Rightarrow \lightning x_{i+1}$. To obtain the bit $k[i]$, the attacker has exploited the fact that a fault in register $y = ax$ propagates to x, denoted by $\lightning y_i \Rightarrow \lightning x_{i+1}$, only if $k[i] = 1$.

As opposed to this non-ladderized variant, in Algorithms 9 and 10 the values of x and y are interleaved and the fault-propagation patterns are the following:

1. For the semi-interleaved ladder:

$$\lightning x_i \Rightarrow (\lightning x_{i+1} \text{ and } (\lightning y_{i+1} \text{ only if } k[i] = 0))$$
$$\lightning y_i \Rightarrow (\lightning y_{i+1} \text{ and } (\lightning x_{i+1} \text{ only if } k[i] = 1))$$

   which means that a fault always propagates to the same register, but propagates to the other depending on the current bit of the secret key.

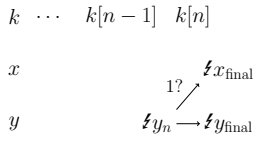

Figure 2: Fault-injection $\lightning y_n$ on the last bit $k[n]$: if the final x value has changed then $k[n] = 1$, otherwise $k[n] = 0$.

2. For the fully-interleaved ladder:

$$(\lightning x_i \text{ or } \lightning y_i) \Rightarrow (\lightning x_{i+1} \text{ and } \lightning y_{i+1})$$

   which means that any fault in one register propagates in every case to both.

The fully-interleaved ladder has a lower *fault tolerance* i.e., it is easier to disrupt the computation. This is not convenient for properties like functionality, availability, or redundancy, but this prevents an attacker from obtaining the secret key, i.e., this increases security. In other words, the ladderization refactoring reduces the information leakage from fault injection by reducing also the fault tolerance. The semi-interleaved ladder is more robust, but this comes at the price of a fault-propagation pattern depending on the secret key, which can be exploited by our first attack.

4.1.2 First Fault-Injection Attack

The attacker can $\lightning y_n$ inject a random fault in register y just before the last iteration, as in Figure 2, then compare the x output. If $x_{\text{fault}} = x_{\text{final}}$ then $k[n] = 0$, otherwise $k[n] = 1$. Thus, the attacker can obtain $k[n]$, the last bit of the secret key.

If the obtained bit was 0 as in Figure 3, the attacker can $\lightning y_{n-1}$. In that case if $x_{\text{fault}} = x_{\text{final}}$ then $k[n-1] = 0$, otherwise $k[n-1] = 1$. This process can be repeated until a 1 is found.

If the obtained bit was 1 as in Figure 4, the attacker can $\lightning x_{n-1}$. In that case if $y_{\text{fault}} = y_{\text{final}}$ then $k[n-1] = 1$, otherwise $k[n-1] = 0$. This process can be repeated until a 0 is found.

One may think that these processes can be alternated in order to recover all the bits of the secret key, but if there is a bit alternation i.e., $(k[i], k[i+1]) = (0,1)$ or $(1,0)$, then the bits before i cannot be obtained, as illustrated in Figures 5 and 6. The attacker could obtain that way all the

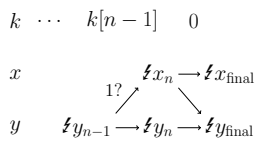

Figure 3: The obtained bit $k[n]$ was 0. Fault-injection $\lightning y_{n-1}$ on $k[n-1]$: if the final x value has changed then $k[n-1] = 1$, otherwise $k[n-1] = 0$.

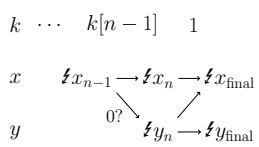

Figure 4: The obtained bit $k[n]$ was 1. Fault-injection $\lightning x_{n-1}$ on $k[n-1]$: if the final y value has changed then $k[n-1] = 0$, otherwise $k[n-1] = 1$.

bits only if the key is trivial, and it seems improbable that the attacker can obtain that way more than a small number of bits. In short, the attacker can obtain the final bits $10\ldots0$ if x_{final} can be read, and the final bits $01\ldots1$ if y_{final} can be read.

The vulnerability to fault injection is summarized in Table 1. Against the first attack, fully-interleaved ladders are more secure than semi-interleaved ladders, which are more secure than without interleaving at all. But we demonstrate in the next subsection that a stronger attacker is able to obtain all the bits of the key from both the semi- and fully-interleaved ladders.

4.2 Second Attacker Model

As opposed to the first attacker model, which uses only random faults, the second attacker model uses both random and zeroing/one-ing faults, thus allowing a wider range of attacks.

Definition 6 (Second Attacker Model)**.** We assume that:

- The second attacker has the same goal and means as the attacker in Definition 5.
- The second attacker can also $\lightning k_{>i} = 0$ (resp. $\lightning k_{>i} = 1$) *stuck-at* 0 (resp. 1) all the bits [50] of the register k between iterations i and $i+1$.

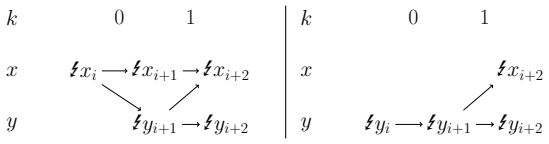

Figure 5: Fault-injection attack $\lightning x_i$ or $\lightning y_i$ before the bit alternation 01: in both cases the fault propagates to both $\lightning x_{i+2}$ and $\lightning y_{i+2}$.

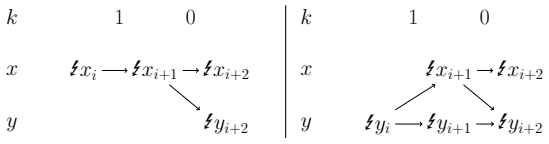

Figure 6: Fault-injection attack $\lightning x_i$ or $\lightning y_i$ before the bit alternation 10: in both cases the fault propagates to both $\lightning x_{i+2}$ and $\lightning y_{i+2}$.

4.2.1 Second Fault-Injection Attack

The second attack is similar to the first one, but it leverages zeroing/one-ing faults to circumvent the bit alternation preventing the first attack from breaking the semi-interleaved ladders. Indeed, after retrieving the last bits of the secret key and reaching a bit alternation 01 (resp. 10), the attacker can:

- run the program on the unknown bits then, when reaching the known bits, zeroing (resp. one-ing) all the secret bits to obtain the final values x_{final} and y_{final},
- run the program on the unknown bits then, just before the last unknown bits, inject a random fault and finally, when reaching the known bits, zeroing (resp. one-ing) all the secret bits to obtain the faulted values x_{fault} and y_{fault}.

Then, as in the first attack, the faulted values can be compared to the final values to determine the current secret bit. Finally, by repeating this process from the last to the first bit, the attacker can obtain all the bits of the secret key.

Therefore, this stronger attacker is able to break the semi-interleaved ladders by using the attack protocol represented[11] in Algorithm 11,

[11] Note that the iterations are reversed in the attack protocol compared to the execution of the studied device.

Table 1: Vulnerability against fault-injection attacks

	Obtained Bits		
	Model 1	Model 2	
Interleaving	Attack 1	Attack 2	Attack 3
non	all	all	all
semi	some	all	all
fully	none	none	all

where $\mathrm{EXE}(x_{\mathrm{init}}, y_{\mathrm{init}}, \lightning k_{>i} = 1)$ at Line 4 means that the studied program is executed with inputs $x_{\mathrm{init}}, y_{\mathrm{init}}$ and a stuck-at $\lightning k_{>i} = 1$, and $\mathrm{EXE}(x_{\mathrm{init}}, y_{\mathrm{init}}, \lightning k_{>i} = 1, \lightning x_i)$ at Line 5 is the same but with a random fault $\lightning x_i$. In particular, if both x_{final} and y_{final} can be read, then the Montgomery Ladder can be broken by iterating over 0 to d in Algorithm 3 p.6. This attack does not work against fully-interleaved ladders, but a fully-interleaved ladder is more difficult to obtain (when even possible), as discussed in Subsections 5.3 and 6.3. As before, the vulnerabilities against this attack are summarized in Table 1.

4.2.2 Third Fault-Injection Attack

According to Kerckhoffs principle [26], an attacker is assumed to know the architecture and the program, and thus is able to duplicate the setup of the targeted chip for a chosen value of the secret key. Thus, the attacker should be able to input k at will and even to inject random faults in the register k, as in registers x and y.

This assumption does not change the initial attacker model in Definition 5 because injecting a random fault in k between the iterations $i-1$ and i is equivalent, if the last $i-1$ bits are known, to input k with the known bits and generate randomly the beginning of the key. But this assumption changes the second attacker model in Definition 6, who is now able to test the bits of the key from left to right.

Actually, being able to write deterministically in the key register is enough to obtain with a third attack all the bits of the key, even against fully-interleaved ladders. Indeed, we consider the second attacker knows the first i bits $k[1] \dots k[i]$, where $i = 0$ for the initialization when no bit is known. By $\lightning k_{>i} = 0$ (resp. $\lightning k_{>i} = 1$) zeroing (resp. one-ing) the last i bits and $\lightning k_{>i+1} = 0$ (resp. $\lightning k_{>i+1} = 1$) zeroing (resp. one-ing) the last $i+1$ bits, the second attacker can compare $x_{\mathrm{fault}0}$ and $x_{\mathrm{final}0}$ (resp. $x_{\mathrm{fault}1}$ and $x_{\mathrm{final}1}$) obtained respectively for $k[1] \dots k[i]\, 0\, 0 \dots 0$ and $k[1] \dots k[i]\, k[i+1]\, 0 \dots 0$ (resp. with 1s instead of 0s at the end), and the same for y. Thus, the second attacker can (almost) determine whether $k[i+1] = 1$ (resp. 0):

- if $x_{\mathrm{fault}0} \neq x_{\mathrm{final}0}$ or $y_{\mathrm{fault}0} \neq y_{\mathrm{final}0}$ then $k[i+1] = 1$,
- if $x_{\mathrm{fault}1} \neq x_{\mathrm{final}1}$ or $y_{\mathrm{fault}1} \neq y_{\mathrm{final}1}$ then $k[i+1] = 0$.

If $x_{\mathrm{fault}0} = x_{\mathrm{final}0}$, $x_{\mathrm{fault}1} = x_{\mathrm{final}1}$, $y_{\mathrm{fault}0} = y_{\mathrm{final}0}$ and $y_{\mathrm{fault}1} = y_{\mathrm{final}1}$ then another inputs $x_{\mathrm{init}}, y_{\mathrm{init}}$ should be chosen in the protocol described in Algorithm 12.

That way all the bits of the secret key k can be obtained, except the ones such that for every possible x_{init}, $x_{\mathrm{fault}0} = x_{\mathrm{final}0}$ and $x_{\mathrm{fault}1} = x_{\mathrm{final}1}$. But this case is unlikely, it does not prevent the attacker from discovering the other bits, and these bits make no difference in the output of the program of interest, thus it is likely that the attacker does not care. Therefore, such attacker can obtain all the (relevant) bits of the key, even against semi- and fully-interleaved ladders. Again, the vulnerabilities against this attack are summarized in Table 1.

4.3 Comparison

Investigating the general pattern of the semi- and fully-interleaved ladders in Algorithms 9 and 10 lead to exhaustively describe the fault-propagation patterns in Subsubsection 4.1.1, which in turn lead to the discovery of the three attacks described in this section. To our knowledge, these attacks are novel, which contributes to the fruitfulness of the ladderization approach, partly answering the **RQ2** research question.

The vulnerabilities against these three fault-injection attacks are summarized in Table 1. Against the first attack, fully-interleaved ladders are more secure than semi-interleaved ladders, which are more secure than without interleaving at all. The second attack is able to break semi-interleaved ladders (for instance, the common Montgomery ladder), but not fully-interleaved ladders. Finally, the third attack is able to break all ladders.

Algorithm 11 Protocol to Attack Semi-Interleaved Ladders

FIx ← False
for $i = n$ **to** 1 **do**
  if FIx = True **then**
    $(x_{\text{final}}, y_{\text{final}}) \leftarrow \text{EXE}(x_{\text{init}}, y_{\text{init}}, \not{\text{f}} k_{>i} = 1)$
    $(x_{\text{fault}}, y_{\text{fault}}) \leftarrow \text{EXE}(x_{\text{init}}, y_{\text{init}}, \not{\text{f}} k_{>i} = 1, \not{\text{f}} x_i)$
    if $y_{\text{fault}} = y_{\text{final}}$ **then**
      $k[i] \leftarrow 1$
    else
      $k[i] \leftarrow 0$
      FIx ← False
    end if
  else
    $(x_{\text{final}}, y_{\text{final}}) \leftarrow \text{EXE}(x_{\text{init}}, y_{\text{init}}, \not{\text{f}} k_{>i} = 0)$
    $(x_{\text{fault}}, y_{\text{fault}}) \leftarrow \text{EXE}(x_{\text{init}}, y_{\text{init}}, \not{\text{f}} k_{>i} = 0, \not{\text{f}} y_i)$
    if $x_{\text{fault}} = x_{\text{final}}$ **then**
      $k[i] \leftarrow 0$
    else
      $k[i] \leftarrow 1$
      FIx ← True
    end if
  end if
end for
return k

But these results have to be mitigated for semi- and fully-interleaved ladders. Indeed, the invariant $y = \ell(x)$ holds for every iteration. So, if a random fault is injected in x or y as in the first two attacks, the violation of the invariant allows the algorithm to detect the fault [18] and thus to enhance the appropriate fault policy e.g., stopping the program or switching to a random key for the rest of the computation. Moreover, the secret key being known by the algorithm, self-secure countermeasures could also be triggered against the third attack.

Therefore, by design, semi- and fully-interleaved ladders provide partial protection against the considered fault-injection attacks, completing the answer to **RQ1** provided in Subsection 3.5. Semi-interleaved ladders are as secure as the Montgomery ladder regarding fault-injection attacks, while fully-interleaved ladders provide more protection.

5 Generalizing the Montgomery Ladder for the Modular Exponentiation

Sections 5 and 6 generalize the algorithmic strength of the Montgomery ladder by providing concrete solutions for the ladder equations in the context of various cryptosystems. In this section we continue our running example started in Subsection 2.3 by focusing on solving (Subsections 5.1 to 5.3) the ladder equations for the modular exponentiation. We compare the complexity of the novel algorithms in Subsection 5.4 with the complexity of the Montgomery ladder. Then, we demonstrate in the case of the RSA [45] (Subsection 5.5) and DSA [40] (Subsection 5.6) cryptosystems that our fully-interleaved solution is practical. Results are summarized in Subsection 5.7.

In the following, for any n we consider $\mathbb{Z}/n\mathbb{Z}$ i.e., integers modulo n, and we denote $(\mathbb{Z}/n\mathbb{Z})^\star$ the integers invertible modulo n. In this section, we assume that the functions $\theta(x)$, $\varepsilon(x)$, $f(x, y)$, $g(x, y)$, and

Algorithm 12 Protocol to Attack Both Semi- and Fully-Interleaved Ladders

for $i = 0$ **to** $n - 1$ **do**
  $(x_{\text{final}}, y_{\text{final}}) \leftarrow \text{EXE}(x_{\text{init}}, y_{\text{init}}, \lightning k_{>i} = 0)$
  $(x_{\text{fault}}, y_{\text{fault}}) \leftarrow \text{EXE}(x_{\text{init}}, y_{\text{init}}, \lightning k_{>i+1} = 0)$
  if $x_{\text{fault}} \neq x_{\text{final}}$ or $y_{\text{fault}} \neq y_{\text{final}}$ **then**
    $k[i+1] \leftarrow 1$
  else
    $(x_{\text{final}}, y_{\text{final}}) \leftarrow \text{EXE}(x_{\text{init}}, y_{\text{init}}, \lightning k_{>i} = 1)$
    $(x_{\text{fault}}, y_{\text{fault}}) \leftarrow \text{EXE}(x_{\text{init}}, y_{\text{init}}, \lightning k_{>i+1} = 1)$
    if $x_{\text{fault}} \neq x_{\text{final}}$ or $y_{\text{fault}} \neq y_{\text{final}}$ **then**
      $k[i+1] \leftarrow 0$
    end if
  end if
end for
return k

$\ell(x)$ occurring in ladder equations (Section 3) are quadratic polynomials with coefficients in $\mathbb{Z}/n\mathbb{Z}$.

Because we consider the modular exponentiation case (Subsection 2.3), the `then` branch corresponds to the function $\theta(x) = ax^2$ and the `else` branch to $\varepsilon(x) = x^2$. For the ladder function $\ell(x) = \ell_2 x^2 + \ell_1 x + \ell_0$, to simplify the analysis we assume $\ell_2 = 0$, obtaining $\ell(x) = \ell_1 x + \ell_0$, which is still a generalization of the invariant $\ell(x) = ax$ obtained in the usual Montgomery ladder. More complicated cases are left for future work.

To summarize, we consider the following coefficients:

$$\begin{aligned}
\theta(x) &= ax^2 \\
\varepsilon(x) &= x^2 \\
f(x,y) &= f_{20}x^2 + f_{11}xy + f_{02}y^2 + f_{10}x + f_{01}y + f_{00} \\
g(x,y) &= g_{20}x^2 + g_{11}xy + g_{02}y^2 + g_{10}x + g_{01}y + g_{00} \\
\ell(x) &= \ell_1 x + \ell_0
\end{aligned}$$

To ensure that $\theta(x) = ax^2$ and $\ell(x) = \ell_1 x + \ell_0$ depend on x, we assume that $a \neq 0$ and $\ell_1 \neq 0$. Moreover, in the following we focus on solutions without constraint on a to preserve the generality of the original Montgomery ladder.

Finally, note that $\mathbb{Z}/n\mathbb{Z}$ may not be an integral domain i.e., there exists $x, y \in \mathbb{Z}/n\mathbb{Z}\backslash\{0\}$ such that $xy = 0 \bmod n$, which is the case for RSA (Subsection 5.5) with p and q such that $n = pq$. Because in this case n cannot be easily factorized into p and q, and because we are looking for general solutions for any n, if $x \neq 0$ and $xy = 0$ we look for solutions $y = 0$, thus loosing some generality but simplifying the analysis.

5.1 Both Semi- and Fully-Interleaved Ladders

We start by solving the ladder equations which are common to the semi- and fully-interleaved cases. By substituting the coefficients and simplifying, Equation (1b) p.9 (resp. Equation (2b) p.10) $f(x, \ell(x)) = \theta(x)$ for the semi- (resp. fully-) interleaved cases is equivalent to:

$$\begin{cases}
f_{20} + f_{11}\ell_1 + f_{02}\ell_1^2 = a \\
f_{11}\ell_0 + 2f_{02}\ell_1\ell_0 + f_{10} + f_{01}\ell_1 = 0 \\
f_{02}\ell_0^2 + f_{01}\ell_0 + f_{00} = 0
\end{cases}$$

Similarly, Equation (1c) p.9 (resp. Equation (2c) p.10) $f(\ell(x), x) = \ell(\varepsilon(x))$ for the semi- (resp. fully-) interleaved cases is equivalent to:

$$\begin{cases}
f_{20}\ell_1^2 + f_{11}\ell_1 + f_{02} = \ell_1 \\
2f_{20}\ell_1\ell_0 + f_{11}\ell_0 + f_{10}\ell_1 + f_{01} = 0 \\
f_{20}\ell_0^2 + f_{10}\ell_0 + f_{00} = \ell_0
\end{cases}$$

5.2 Semi-Interleaved Ladder Solution

In this subsection we solve the ladder equations for the semi-interleaved case. The remaining Equation (1a) p.9 $\varepsilon(\ell(x)) = \ell(\theta(x))$ for the semi-

interleaved case is equivalent to:

$$\begin{cases} \ell_1^2 = \ell_1 a \\ 2\ell_1\ell_0 = 0 \\ \ell_0^2 = \ell_0 \end{cases}$$

So, because $\ell_1 \neq 0$, we assume $\ell_1 = a$ and $\ell_0 = 0$. Therefore Equation (1b) is equivalent to:

$$\begin{cases} f_{20} + f_{11}a + f_{02}a^2 = a \\ f_{10} + f_{01}a = 0 \\ f_{00} = 0 \end{cases}$$

and Equation (1c) is equivalent to:

$$\begin{cases} f_{20}a^2 + f_{11}a + f_{02} = a \\ f_{10}a + f_{01} = 0 \\ f_{00} = 0 \end{cases}$$

From $f_{10} + f_{01}a = f_{10}a + f_{01}$ we deduce $f_{10}(a - 1) = f_{01}(a-1)$, which is satisfied without constraint on a if $f_{10} = f_{01}$. Moreover, from $f_{10} + f_{01}a = 0$ we obtain $f_{10} = -f_{01}a$, thus from $f_{10}a + f_{01} = 0$ we obtain $f_{01}(a^2 - 1) = 0$, which is always true without constraint on a if $f_{01} = 0$. Therefore, we assume $f_{10} = 0 = f_{01}$.

From $f_{20} + f_{11}a + f_{02}a^2 = f_{20}a^2 + f_{11}a + f_{02}$ we deduce $f_{20}(a^2 - 1) = f_{02}(a^2 - 1)$, which is satisfied without constraint on a if $f_{20} = f_{02}$. The remaining constraint is $f_{20}(a^2 + 1) + f_{11}a = a$, which is equivalent to $f_{20} = \frac{a}{a^2+1}(1 - f_{11})$. Because the coefficients are integers there exists m such that $1 - f_{11} = m(a^2 + 1)$. Thus, we have $f_{20} = ma$ and $f_{11} = 1 - m(a^2 + 1)$.

Theorem 2. *The square-and-multiply algorithm in Algorithm 2 for the modular exponentiation with $\theta(x) = ax^2$ and $\varepsilon(x) = x^2$ is semi-ladderizable with:*

$$\ell(x) = ax$$
$$f(x, y) = ma(x^2 + y^2) + (1 - m(a^2 + 1))xy$$

producing (after memoization of one squaring in z) Algorithm 13.

Note that if $m = 0$, then $f(x, y) = xy$, in which case this solution is the common Montgomery ladder. In this restricted case, the solution was generated semi-automatically in a previous paper [11] by solving the ladder equations.

Algorithm 13 Semi-Interleaved Ladder for the Modular Exponentiation

input public a, n; secret k
1: $x \leftarrow 1$
2: $y \leftarrow a \bmod n$
3: $m \leftarrow \text{random}([0, n-1])$
4: $c_1 \leftarrow ma \bmod n$
5: $c_2 \leftarrow 1 - (c_1 a + m) \bmod n$
6: **for** $i = d$ **to** 0 **do**
7: **if** $k[i] = 1$ **then**
8: $z \leftarrow y^2 \bmod n$
9: $x \leftarrow c_1(x^2 + z) + c_2 xy \bmod n$
10: $y \leftarrow z$
11: **else**
12: $z \leftarrow x^2 \bmod n$
13: $y \leftarrow c_1(y^2 + z) + c_2 yx \bmod n$
14: $x \leftarrow z$
15: **end if**
16: **end for**
17: **return** x
output $x = a^k \bmod n$

Note also that the coefficients c_1 and c_2 only depend on m, so they can be precomputed (Lines 4 and 5) before the loop, for the sake of performance.

Finally, because there is no constraint on m, it can be randomly generated (Line 3). Even if we have $x \leftarrow \theta(x) = ax^2$ at the end of each iteration (Section 3), such a randomly generated integer would affect the bits flips that can be observed during intermediate computations of $x \leftarrow c_1(x^2 + z) + c_2 xy \bmod n$ (Line 9). Thus, m can be used as a blinding integer (Subsection 2.3) to reduce vulnerability against advanced side-channel attacks like CPA [10].

5.3 Fully-Interleaved Ladder Solution (General Case)

In this subsection we solve the ladder equations for the fully-interleaved case. Remain two equations. Equation (2a) p.10 $g(\theta(x), \ell(x)) = \ell(\theta(x))$ for the

fully-interleaved cases is equivalent to:

$$\begin{cases} g_{20}a^2 = 0 \\ g_{11}\ell_1 a = 0 \\ g_{11}\ell_0 a + g_{02}\ell_1^2 + g_{10}a = \ell_1 a \\ 2g_{02}\ell_1\ell_0 + g_{01}\ell_1 = 0 \\ g_{02}\ell_0^2 + g_{01}\ell_0 + g_{00} = \ell_0 \end{cases}$$

So, because $a \neq 0$ and $\ell_1 \neq 0$, we assume $g_{20} = 0$ from the first equation, $g_{11} = 0$ from the second, and the remaining constraints are:

$$\begin{cases} g_{02}\ell_1^2 + g_{10}a = \ell_1 a \\ 2g_{02}\ell_1\ell_0 + g_{01}\ell_1 = 0 \\ g_{02}\ell_0^2 + g_{01}\ell_0 + g_{00} = \ell_0 \end{cases}$$

According to Equation (2c) investigated in Subsection 5.1, $f(\ell(x), x) = \ell(\varepsilon(x)) = \ell_1 x^2 + \ell_0$, so by using $g_{20} = g_{11} = 0$ we have:

$$\begin{aligned} &g(f(\ell(x), x), x) \\ &= (g_{02} + g_{10}\ell_1)x^2 + (g_{01})x + (g_{10}\ell_0 + g_{00}) \end{aligned}$$

Finally, the last equation is Equation (2d) p.10 $g(f(\ell(x), x), x) = \varepsilon(x)$ for the fully-interleaved cases is equivalent to:

$$\begin{cases} g_{02} + g_{10}\ell_1 = 1 \\ g_{01} = 0 \\ g_{10}\ell_0 + g_{00} = 0 \end{cases}$$

Because $g_{20} = g_{11} = g_{01} = 0$ and we require $g(x, y) = g_{02}y^2 + g_{10}x + g_{00}$ to depend on both x and y, we have to assume $g_{02} \neq 0$ and $g_{10} \neq 0$. By using $g_{01} = 0$ the remaining constraints from Equation (2a) are:

$$\begin{cases} g_{02}\ell_1^2 + g_{10}a = \ell_1 a \\ g_{02}\ell_1\ell_0 = 0 \\ g_{02}\ell_0^2 + g_{00} = \ell_0 \end{cases}$$

Because $g_{02} \neq 0$ and $\ell_1 \neq 0$, the second equality implies that $\ell_0 = 0$. Thus, the remaining constraints from Equation (2a) and Equation (2d) are:

$$\begin{cases} g_{02}\ell_1^2 + g_{10}a = \ell_1 a \\ g_{02} + g_{10}\ell_1 = 1 \\ g_{20} = g_{11} = g_{01} = g_{00} = 0 \end{cases}$$

By using the second equality $g_{02} = 1 - g_{10}\ell_1$, the first equality is equivalent to:

$$g_{10}(a - \ell_1^3) = \ell_1(a - \ell_1)$$

Because $g_{10} \neq 0$ and $\ell_1 \neq 0$, note that $a = \ell_1^3 \Leftrightarrow a = \ell_1$, in which case $\ell_1^3 = \ell_1$, thus $a = \ell_1 = \pm 1$. To remove the constraint on a, we assume that $a \neq \ell_1^3$, and thus g_{10} and g_{02} depend only on a and ℓ_1:

$$\begin{aligned} g_{10} &= \ell_1 \frac{a - \ell_1}{a - \ell_1^3} \\ g_{02} &= 1 - g_{10}\ell_1 = a\frac{1 - \ell_1^2}{a - \ell_1^3} \\ \text{so } g(x, y) &= g_{02}y^2 + g_{10}x \\ &= \frac{1}{a - \ell_1^3}\left(a(1 - \ell_1^2)y^2 + \ell_1(a - \ell_1)x\right) \end{aligned}$$

Because $\ell_1 \neq 0$ and $g_{10} \neq 0$ we have to assume that, as opposed to the semi-interleaved cases, $\ell_1 \neq a$. As for the semi-interleaved cases, because $\ell_0 = 0$, the constraints from the common Equations (2b) and (2c) can be simplified:

$$\begin{cases} f_{20} + f_{11}\ell_1 + f_{02}\ell_1^2 = a & (L_1) \\ f_{20}\ell_1^2 + f_{11}\ell_1 + f_{02} = \ell_1 & (L_2) \\ f_{10} + f_{01}\ell_1 = 0 & (L_3) \\ f_{10}\ell_1 + f_{01} = 0 & (L_4) \\ f_{00} = 0 \end{cases}$$

In order to have a non-trivial ladder $\ell(x) = \ell_1 x$, we assume that $\ell_1 \neq \pm 1$. With $(L_4) - (L_3)$ we obtain $(f_{10} - f_{01})(\ell_1 - 1) = 0$ thus $f_{10} = f_{01}$, and with (L_3) we have $f_{10}(\ell_1 + 1) = 0$, so $f_{10} = f_{01} = 0$. With $(L_1) - (L_2)$ we obtain $(f_{02} - f_{20})(\ell_1^2 - 1) = a - \ell_1$, so $f_{02} = f_{20} + \frac{a - \ell_1}{\ell_1^2 - 1}$. Therefore, remain the following constraints:

$$\begin{cases} f_{20}(\ell_1^2 + 1) + f_{11}\ell_1 = a - \ell_1^2\dfrac{a - \ell_1}{\ell_1^2 - 1} = \ell_1 - \dfrac{a - \ell_1}{\ell_1^2 - 1} \\ f_{02} = f_{20} + \dfrac{a - \ell_1}{\ell_1^2 - 1} \\ f_{10} = f_{01} = f_{00} = 0 \end{cases}$$

The first equality is obtained from (L_1) and the second from (L_2). Note that $a - \ell_1^2\frac{a - \ell_1}{\ell_1^2 - 1} = \ell_1 - \frac{a - \ell_1}{\ell_1^2 - 1}$ is already satisfied and thus is not a constraint. Moreover, because $\ell_1 \neq 0$, by using the second line,

we can rewrite f_{11} as:

$$f_{11} = \frac{1}{\ell_1}\left(\ell_1 - \frac{a-\ell_1}{\ell_1^2-1} - f_{20}(\ell_1^2+1)\right)$$
$$= \frac{1}{\ell_1}\left(\frac{\ell_1^3-a}{\ell_1^2-1} - f_{20}(\ell_1^2+1)\right)$$

Thus, we have:

$$f(x,y) = f_{20}x^2 + f_{11}xy + f_{02}y^2$$
$$= f_{20}x^2 + \frac{1}{\ell_1}\left(\frac{\ell_1^3-a}{\ell_1^2-1} - f_{20}(\ell_1^2+1)\right)xy + \left(f_{20} + \frac{a-\ell_1}{\ell_1^2-1}\right)y^2$$
$$= f_{20}\left(x^2 - \frac{\ell_1^2+1}{\ell_1}xy + y^2\right) + \frac{1}{\ell_1^2-1}\left(\frac{\ell_1^3-a}{\ell_1}xy + (a-\ell_1)y^2\right)$$

Finally, f_{20} has no constraint. It could have been used as a blinding integer for f, but unfortunately not for g. So, for the sake of simplicity and performance, we assume $f_{20} = 0$ and we obtain the following functions:

Theorem 3. *The square-and-multiply Algorithm 2 for the modular exponentiation with $\theta(x) = ax^2$ and $\varepsilon(x) = x^2$ is fully-ladderizable with:*

$$\ell(x) = \ell_1 x$$
$$f(x,y) = \frac{1}{\ell_1^2-1}\left(\frac{\ell_1^3-a}{\ell_1}xy - (\ell_1-a)y^2\right)$$
$$g(x,y) = \frac{1}{\ell_1^3-a}\left(a(\ell_1^2-1)y^2 + \ell_1(\ell_1-a)x\right)$$

where ① $\ell_1 \neq a \bmod n$, ② $\ell_1 \in (\mathbb{Z}/n\mathbb{Z})^\star$, ③ $\ell_1^2 - 1 \in (\mathbb{Z}/n\mathbb{Z})^\star$, and ④ $\ell_1^3 - a \in (\mathbb{Z}/n\mathbb{Z})^\star$, producing (after memoization of one squaring in z) Algorithm 14.

Definition 7 (Ladder Constant)**.** We call *ladder constant* and denote by ℓ an integer satisfying constraints ①, ②, ③, and ④ from Theorem 3.

The first constraint ① on the ladder constant can be satisfied by checking whether $\ell - a = 0 \bmod n$. The other constraints can be satisfied by using the Extended Euclidean Algorithm $(d, u, u') \leftarrow \mathrm{EEA}(v, n)$, where $uv + u'n = d$, and $d = \gcd(v, n)$. Indeed, we have $uv = d \bmod n$, such that if $d = 1$ then v is invertible modulo n and $v^{-1} = u \bmod n$. Because we ignore the last Bézout coefficient u', for sake of simplicity we denote this function call $(d, u) \leftarrow \mathrm{EEA}(v, n)$ in Algorithm 14.

Because of ①, we have $\ell \neq a \bmod n$. Moreover, $0 \notin (\mathbb{Z}/n\mathbb{Z})^\star$, so according to ②, we have $\ell \neq 0$. Finally, if $\ell = \pm 1$, then $\ell_1^2 - 1 \notin (\mathbb{Z}/n\mathbb{Z})^\star$, so according to ③, we have $\ell \neq \pm 1$. Hence, we have that $\ell \in \mathbb{Z}/n\mathbb{Z}$ has to be chosen in $\mathrm{L}_{a,n} = [2, n-2] \setminus \{a\}$ (Line 2).

Then, Algorithm 14 checks if ℓ satisfies the constraints of Theorem 3 (Line 9). After a suitable ladder constant ℓ is chosen, the coefficients of f and g only depend on a and ℓ, thus can be pre-computed (Lines 10–13).

These computations to find suitable constants may cause an overhead (Lines 1–15) but that does not depend on the secret thus, in theory, trading performance for security. Nevertheless, we prove in the RSA case (Subsection 5.5) and the DSA case (Subsection 5.6) that a ladder constant ℓ satisfying the constraints from Theorem 3 is almost always obtained after one iteration. So, in practice, such overhead is negligible compared to the operations performed for each bit of the secret key.

Finally, note that ℓ is randomly generated (Line 2) and affects the computation of intermediate values (Lines 19, 20, 23 and 24) depending on the secret key. Hence, as in our semi-interleaved solution (Subsection 5.2), ℓ is a blinding integer that reduces vulnerability against advanced side-channel attacks.

5.4 Complexity and Performance

We provide in Table 2 a comparison of complexities for the Montgomery, semi-, and fully-interleaved ladders. The complexities are given in terms of cost per key bit, where M stands for multiplication, S for squaring, A for addition and subtraction, all modulo n, and T for the thread creation cost. Because the costs are given per key bit, we did not include the cost of the pre-computations, which are negligible in the RSA and the DSA cases, as described at the end of Subsections 5.2 and 5.3.

The first line of Table 2 demonstrates a trade-off between security and performance. As opposed to the Montgomery ladder, our semi- (Theorem 2) and fully- (Theorem 3) interleaved solutions have a blinding integer reducing vulnerability to advanced

Table 2: Cost per key bit (after memoization) of the Montgomery (Algorithm 3), semi- (Algorithm 13) and fully-interleaved ladders (Algorithm 14) for the modular exponentiation, where M stands for multiplication, S for squaring, A for addition/subtraction, and T for thread creation cost.

	Montgomery	Semi-Interleaved	Fully-Interleaved
sequential (1 core)	$M+S$	$3M+2S+2A$	$5M+S+2A$
parallel (2 cores)	$M+S$	$M+2S+2A+T$	$3M+S+A+T$
parallel (3 cores)	$M+S$	$2M+2A+T$	$2M+S+A+T$

side-channel attacks, but at the price of a factor 3 for the former and 5 for the latter, in terms of multiplications. Nevertheless, as illustrated in Table 1 p.14, our fully-interleaved solution provides more protection against fault-injection attacks than the Montgomery ladder or our semi-interleaved solution.

As introduced in the background (Subsection 2.5), we also consider parallel computations to evaluate performance. The second and third line of Table 2 provide a comparison when the algorithms run on a multi-core system, respectively with two and three cores. This does not change the cost for the Montgomery ladder (Algorithm 3) because the squaring must be computed after the multiplication. But this changes the cost for our solutions.

We now consider the `then` branch of our semi-interleaved ladder (Algorithm 13), both branches having the same complexity. We first consider two cores. $c_1(x^2+y^2)$ can be computed on the first core using two squarings, one addition, then one multiplication, i.e., $M+2S+A$. c_2xy can be computed on the second core using two multiplications, i.e., $2M$. Since usually $S \approx 0.8M$, we have $M \leq 2S+A$ and the maximal complexity between both cores is $M+2S+A$. Finally, the results are added together to obtain the value for x, obtaining a total complexity $M+2S+2A$, and the computed value y^2 is assigned to y. We now consider three cores. c_1x^2 can be computed on the first core in $M+S$, c_1y^2 can be computed on the second core in $M+S$, and c_2xy can be computed on the third core in $2M$. Since $M \geq S$, the maximal complexity between the cores is $2M$. Then, the three results are summed together, for a total complexity of $2M+2A$. Note that, since usually $S \approx 0.8M$, the total complexity for two cores is larger than for three cores, as expected.

In the same way, we consider the `then` branch of the fully-interleaved ladder (Algorithm 14). We first consider two cores. $c_0xy+c_1y^2$ can be computed on the first core in $3M+S+A$ to obtain the value for x. $c_2y^2+c_3x$ can be computed on the second core in $2M+S+A$ to obtain the value for y. Hence, the total complexity is $3M+S+A$. We now consider three cores. c_0xy can be computed on the first core in $2M$, c_1y^2 can be computed on the second core in $M+S$, and $c_2y^2+c_3x$ can be computed on the third core in $2M+S+A$. The results of the first two cores are then added to obtain the value for x, for a total complexity of $2M+A$ for these cores. The third core for y requires $2M+S+A$, hence the total complexity is $2M+S+A$.

These parallel computations do no change qualitatively the trade-offs between performance and security properties, but the price for the semi-interleaved ladder is now a factor 2 for the number of multiplications (instead of 3), and a factor 3 (instead of 5) for the fully-interleaved ladder. Hence, in that case, our solutions demonstrate performances closer to the Montgomery ladder.

In practice, performances are even closer than in theory. As a proof of concept, we implemented the Montgomery and semi-interleaved ladders using the GNU Multiple Precision (GMP) library [19]. To lower the non-negligible cost of memory management induced by this library, we used the provided low-level functions. We measured the number of cycles necessary to execute these algorithms for various key sizes on a single core. We also used the OpenMP API to measure the number of cycles for respectively 2 and 3 cores [42].

We provide a repository [34] to replicate the results, which was tested on a Intel® Core™ i9-11900KF @ 3.50GHz, gcc 11.3.0, Ubuntu 22.04. Each measured function was first executed 700 times (warm-up). Then, to lower the effect of background tasks running on the system, each iteration was repeated 700 times with the same inputs for the median value to be recorded.

Algorithm 14 Fully-Interleaved Ladder for the Modular Exponentiation (General Case)

input public a, n; secret k

do
  $\ell \leftarrow \text{random}([2, n-2] \setminus \{a\})$
  $v_0 \leftarrow \ell - a \bmod n$
  $(d_1, u_1) \leftarrow \text{EEA}(\ell, n)$
  $v_2 \leftarrow \ell^2 - 1 \bmod n$
  $(d_2, u_2) \leftarrow \text{EEA}(v_2, n)$
  $v_3 \leftarrow \ell^3 - a \bmod n$
  $(d_3, u_3) \leftarrow \text{EEA}(v_3, n)$
while $v_0 \bmod n = 0 \vee d_1 \neq 1 \vee d_2 \neq 1 \vee d_3 \neq 1$
$c_0 \leftarrow u_1 u_2 v_3 \bmod n$
$c_1 \leftarrow -v_0 u_2 \bmod n$
$c_2 \leftarrow a v_2 u_3 \bmod n$
$c_3 \leftarrow \ell v_0 u_3 \bmod n$
$x \leftarrow 1$
$y \leftarrow \ell$
for $i = d$ **to** 0 **do**
  if $k[i] = 1$ **then**
    $z \leftarrow y^2 \bmod n$
    $x \leftarrow c_0 xy + c_1 z \bmod n$
    $y \leftarrow c_2 z + c_3 x \bmod n$
  else
    $z \leftarrow x^2 \bmod n$
    $y \leftarrow c_0 yx + c_1 z \bmod n$
    $x \leftarrow c_2 z + c_3 y \bmod n$
  end if
end for
return x

output $x = a^k \bmod n$

The results are reported in Table 3. As the Montgomery ladder cannot be parallelized, we use the number of cycles obtained for this algorithm on a single core as a reference. Thus, we report the number of cycles for the semi-interleaved ladder as well as the ratio, i.e., this number divided by the number of cycles for the Montgomery ladder, for a relative comparison.

Dealing with parallelism increases the ratio for small key sizes (1024 and 2048). This is because thread creation has a fixed cost, which has a major impact on overall performance. This is compensated for larger key sizes, where the performed computations take precedence over the thread creation cost. For several cores and large key sizes, we obtain a ratio between 1.4 and 1.9, which is below the ratio of 2 expected from the cost analysis. One of the reasons is the instructions per cycle (IPC) for the semi-interleaved ladder is larger than for the Montgomery ladder.

Therefore, the price for the more secure semi-interleaved ladder compared to the Montgomery ladder is lower in practice, up to 1.4 for several cores and large key sizes.

5.5 Fully-Interleaved Ladder Solution (RSA Case)

We prove in this subsection that in the RSA case a ladder constant ℓ satisfying the constraints from Theorem 3 can almost always be obtained in one iteration from Algorithm 14.

In the context of RSA [45], we assume that the modulus is a product $n = pq$, where p, q are distinct primes. The modular exponentiation aims at computing $a^k \bmod n$, so we assume that the input a is already given modulo n i.e., $0 \leq a \leq n-1$. Because 0^k and $(\pm 1)^k$ are fairly trivial to compute, we assume that $a \neq 0, \pm 1$. So $2 \leq a \leq n-2$, thus $\text{card}(\text{L}_{a,n}) = n - 4$ where $\text{L}_{a,n} = [2, n-2] \setminus \{a\}$, which simplifies the following reasoning.

We assume that ℓ is randomly chosen in $\text{L}_{a,n}$ and we compute the probability that ℓ satisfies the ladder constraints. To satisfy constraints ① and ② of Theorem 3, i.e., $\ell \neq a$ and $\ell \in (\mathbb{Z}/n\mathbb{Z})^\star$, ℓ has to be chosen in $\text{L}_{a,n}$ such that $\gcd(\ell, n) = 1$. The only integers in $\text{L}_{a,n}$ such that $\gcd(\ell, n) > 1$ are jp for $j \in [1, q-1]$ or jq for $j \in [1, p-1]$, so the probability to randomly select a ladder constant not verifying ② is:

$$Pr(\text{not } ②) \leq \frac{(p-1)+(q-1)}{n-4} \approx \frac{1}{p} + \frac{1}{q}$$

In the RSA context, this means that a random $\ell \in \text{L}_{a,n}$ is invertible modulo n with probability almost equal to 1. Indeed, according to the National Institute of Standards and Technology (NIST) recommendations [4], if $\text{nBits} = \lfloor \log_2 n \rfloor + 1$ denotes the number of bits of $n = pq$, then p and q satisfy:

$$2^{\frac{\text{nBits}-1}{2}} < p < 2^{\frac{\text{nBits}}{2}}$$
$$2^{\frac{\text{nBits}-1}{2}} < q < 2^{\frac{\text{nBits}}{2}}$$

so $\frac{1}{2^{\frac{\text{nBits}-2}{2}}} < \frac{1}{p} + \frac{1}{q} < \frac{1}{2^{\frac{\text{nBits}-3}{2}}}$, where nBits is assumed to be large.

Table 3: Performance comparison between the Montgomery ladder and the semi-interleaved ladder for the modular exponentiation

Key size (bits)	Number of cycles (ratio compared to Montgomery)			
	Montgomery	Semi-Interleaved		
		1 core	2 cores	3 cores
1 024	3 324 108	7 549 860 (2.27)	11 144 772 (3.35)	9 430 042 (2.84)
2 048	21 047 774	48 250 646 (2.29)	49 601 524 (2.36)	39 992 556 (1.90)
4 096	130 630 682	290 334 462 (2.22)	248 868 716 (1.90)	206 804 182 (1.58)
8 192	811 361 402	1 755 290 126 (2.16)	1 393 469 694 (1.71)	1 173 139 582 (1.44)
16 384	4 971 382 772	10 437 469 318 (2.10)	8 181 204 906 (1.65)	6 979 736 808 (1.40)

Moreover, note that if one can compute ℓ such that $\gcd(\ell, n) > 1$, then $\gcd(\ell, n) = p$ or q, thus one is able to factorize n. The same problem occurs in RSA cryptosystem: if Bob can generate a message m such that $\gcd(m, n) > 1$, where n is Alice's public key, then Bob can factorize n and find Alice's private key. Such an event could occur but with a probability nearly equal to 0.

To satisfy constraint ③ of Theorem 3 i.e., $\ell^2 - 1 \in (\mathbb{Z}/n\mathbb{Z})^\star$, note that, according to the Chinese remainder theorem, integers ℓ such that $\ell^2 - 1$ are not invertible modulo n verify $\ell^2 = 1 \bmod p$ and $\ell^2 = 1 \bmod q$. There are at most four such integers: 1, $n-1$, γ, $n-\gamma$, where $\gamma = 1 \bmod p$ and $\gamma = -1 \bmod q$. 1 and $n-1$ are excluded, so the probability to randomly select a ladder constant in $\mathrm{L}_{a,n} = [2, n-2]$ not verifying ③ is:

$$Pr(\text{not } ③) \leq \frac{2}{n-4}$$

Finally, to determine integers such that the last constraint ④ of Theorem 3, i.e., $\ell^3 - a \in (\mathbb{Z}/n\mathbb{Z})^\star$, is satisfied, we use the following lemma from Gauss:

Definition 8 (r^{th} Residues)**.** Let n and $r \geq 2$ be two integers. An integer $a \in (\mathbb{Z}/n\mathbb{Z})^\star$ is a r^{th} *residue* modulo n if there exists an integer $\ell \in \mathbb{Z}/n\mathbb{Z}$ such that $\ell^r = a \bmod n$, otherwise it is a *non-residue* modulo n.

Lemma 1 (Gauss [17])**.** *Let p be a prime and $r \geq 2$ be an integer. We denote $b = \gcd(p-1, r)$ and we have:*

1. *For every integer $a \in (\mathbb{Z}/p\mathbb{Z})^\star$, a is a r^{th} residue modulo p if and only if $a^{\frac{p-1}{b}} = 1 \bmod p$,*
2. *there exists exactly $\frac{p-1}{b}$ r^{th} residues in $(\mathbb{Z}/p\mathbb{Z})^\star$,*
3. *if $a \in (\mathbb{Z}/p\mathbb{Z})^\star$ is a r^{th} residue modulo p, then there exists exactly b integers $\ell \in \mathbb{Z}/p\mathbb{Z}$ such that $\ell^r = a \bmod p$.*

According to the Chinese remainder theorem, $\ell^3 - a$ is invertible modulo $n = pq$ if and only if $\ell^3 - a$ is invertible modulo p and $\ell^3 - a$ is invertible modulo q. Moreover, because p is prime, $\ell^3 - a$ is invertible mod p if and only if $\ell^3 - a \neq 0 \bmod p$, i.e., a is a 3^{rd} non-residue modulo p. Therefore, constraint ④ is not satisfied only if a is a 3^{rd} residue modulo p or q.

For an input a fixed, because $b_p = \gcd(p-1, 3) = 1$ or 3 and $b_q = \gcd(q-1, 3) = 1$ or 3, according to Lemma 1, there are at most 9 non invertible $\ell^3 - a$ integers modulo n, so the probability to randomly select a ladder constant in $\mathrm{L}_{a,n}$ not verifying ④ is:

$$Pr(\text{not } ④) \leq \frac{9}{n-4}$$

At worst, the unfavorable cases are distinct, thus the probability to randomly select a suitable ladder constant is:

$$\begin{aligned} Pr(①②③④) &\geq 1 - \left(\tfrac{p-1+q-1}{n-4} + \tfrac{2}{n-4} + \tfrac{9}{n-4}\right) \\ &= 1 - \tfrac{p+q+9}{n-4} \approx 1 - \tfrac{1}{p} - \tfrac{1}{q} - \tfrac{1}{n} \end{aligned}$$

which is almost equal to 1 in a cryptographic context. Thus, in the RSA case $n = pq$, finding a suitable ladder constant (Lines 1–9) in Algorithm 14 costs only one iteration with probability almost equal to 1. Therefore, the overhead to pre-compute a suitable ladder constant in Algorithm 14 is not costly in practice.

5.6 Fully-Interleaved Ladder Solution (DSA Case)

We prove in this subsection that in the DSA case a ladder constant ℓ satisfying the constraints from

Theorem 3 can almost always be obtained in one iteration from Algorithm 14. In a Digital Signature Algorithm (DSA) [40] context (or for Diffie-Hellman key-exchange mechanism [44]), the exponentiation is computed modulo $n = p$, where p is prime.

The argument is similar to the previous one in Subsection 5.5 for the RSA case, but even simpler. If $\ell \in \mathrm{L}_{a,n} = [2, n-2] \setminus \{a\}$ then ① $\ell \neq a$. Moreover $\ell \neq 0 \bmod n$ so ② ℓ is invertible modulo n. Finally, $\ell \neq \pm 1$ so ③ $\ell^2 - 1 \neq 0 \bmod n$ thus $\ell^2 - 1$ is invertible modulo n. So the first three constraints are always satisfied for $n = p$, where p is prime.

Remains the last constraint ④ $\ell^3 - a$ is invertible modulo n. In the following, we denote $\mathrm{R}_n^3(a)$ that a is a cubic residue modulo n, and let $b = \mathrm{pgcd}(n-1, 3)$. To compute the probability of satisfying the fourth constraint, we use the formula of total probability $Pr(④) = Pr\big(④ \mid \mathrm{R}_n^3(a)\big) \times Pr\big(\mathrm{R}_n^3(a)\big) + Pr\big(④ \mid \text{not } \mathrm{R}_n^3(a)\big) \times Pr\big(\text{not } \mathrm{R}_n^3(a)\big)$.

According to Lemma 1, there exists $\frac{n-1}{b}$ cubic residue in $(\mathbb{Z}/n\mathbb{Z})^\star$. The excluded values -1 and 1 are cubic residue modulo n, as opposed to 0. So, the probability that an integer $a \in [2, n-2]$ is a cubic residue is:

$$Pr\big(\mathrm{R}_n^3(a)\big) = \frac{\frac{n-1}{b} - 2}{n-3} = \frac{1}{b} - \frac{2\left(1 - \frac{1}{b}\right)}{n-3}$$

According to Lemma 1, if a is a cubic residue, then there exists b cubic roots ℓ of a in $(\mathbb{Z}/n\mathbb{Z})^\star$. If $\ell = 0$ or ± 1 then $a = \ell^3 = 0$ or ± 1, which is excluded. If $\ell = a$ then $a = a^3$ i.e., $a(a+1)(a-1) = 0 \bmod p$, thus $a = 0, \pm 1$, which is also excluded. So, cubic roots ℓ of a are automatically in $\mathrm{L}_{a,n} = [2, n-2] \setminus \{a\}$. Thus, the probability that an integer $\ell \in \mathrm{L}_{a,n}$ is a cubic root of a is $\frac{b}{n-4}$. Therefore, if a is a cubic residue then the probability that $\ell \in \mathrm{L}_{a,n}$ satisfies the last constraint is:

$$Pr\big(④ \mid \mathrm{R}_n^3(a)\big) = 1 - \frac{b}{n-4}$$

Moreover, if a is not a cubic residue, then there exists no ℓ such that $a = \ell^3 \bmod n$. So, every $\ell \in \mathrm{L}_{a,n}$ satisfies $a - \ell^3 \neq 0$ modulo $n = p$, where p is prime. Thus, in that case, every $\ell \in \mathrm{L}_{a,n}$ satisfies the last constraint ④, i.e., $\ell^3 - a$ is invertible modulo n. Therefore, $Pr\big(④ \mid \text{not } \mathrm{R}_n^3(a)\big) = 1$.

Finally, the probability to randomly select a suitable ladder constant is:

$$\begin{aligned}
&Pr(①②③④) \\
&\quad = Pr(④) \\
&\quad = Pr\big(④ \mid \mathrm{R}_n^3(a)\big) \times Pr\big(\mathrm{R}_n^3(a)\big) \\
&\qquad + Pr\big(④ \mid \text{not } \mathrm{R}_n^3(a)\big) \times Pr\big(\text{not } \mathrm{R}_n^3(a)\big) \\
&\quad = \left(1 - \tfrac{b}{n-4}\right)\left(\tfrac{1}{b} - \tfrac{2\left(1-\frac{1}{b}\right)}{n-3}\right) \\
&\qquad + 1\left(1 - \tfrac{1}{b} + \tfrac{2\left(1-\frac{1}{b}\right)}{n-3}\right) \\
&\quad = 1 - \tfrac{1}{n-4} + \tfrac{2(b-1)}{(n-3)(n-4)}
\end{aligned}$$

which is almost equal to 1 in a cryptographic context. Therefore, again, the overhead to precompute a suitable ladder constant in Algorithm 14 is not costly in practice.

5.7 Results for the Modular Exponentiation

In this section, we provided concrete answers to **RQ1** by solving the ladder equations for cryptosystems using the modular exponentiation. We obtained Algorithm 13 for the semi-interleaved case and Algorithm 14 for the fully-interleaved case.

Both Algorithms 13 and 14 use a blinding integer, providing more protection against side-channel attacks than the Montgomery ladder, for instance against advanced power attacks like CPA [10]. This protection naturally arose from a free parameter while solving the ladder equations, demonstrating the fruitfulness of the ladderization approach (**RQ2**). Moreover, as detailed in Subsection 4.3, Algorithm 14 is more secure against fault-injection attacks than the Montgomery ladder. So, the security properties of the Montgomery ladder have not only been generalized, but improved in both cases.

We compared in Subsection 5.4 the complexity of these novel algorithms with the complexity of the Montgomery ladder, demonstrating a trade-off between security and performance: Algorithm 13 is more secure than the Montgomery ladder but less performant, and Algorithm 14 is even more secure but even less performant, even if these novel algorithms have performance closer to the Montgomery ladder if the computations can be parallelized. For instance, we demonstrated for the semi-interleaved ladder that in practice, for several cores and large key sizes, the cost is only 40% larger.

Finally, Algorithm 14 requires a ladder constant (Definition 7), which raised a challenge regarding

performance (**RQ3**). We tackled this challenge by proving, in the case of the RSA [45] (Subsection 5.5) and DSA [40] (Subsection 5.6) cryptosystems, that this ladder constant can be obtained at a negligible cost with probability almost equal to 1.

6 Generalizing the Montgomery Ladder for the Scalar Multiplication in ECC

As detailed in Subsection 2.4, the main operation in elliptic-curve cryptography (ECC) is the scalar multiplication $kA = A + \cdots + A$, where A is a point on a curve and k is an integer. Following the double-and-add Algorithm 4 p.7, for a given point A we consider in this section the following known functions:

$$\theta(P) = 2P + A$$
$$\varepsilon(P) = 2P$$

Moreover, we consider ladder functions (Section 3) of the following form:

$$f(P,Q) = f_P P + f_Q Q + f_A A$$
$$g(P,Q) = g_P P + g_Q Q + g_A A$$
$$\ell(P) = \ell_P P + \ell_A A$$

with $f_P, f_Q, \ell_P \neq 0$ for both cases, and $g_P, g_Q \neq 0$ for the fully-interleaved ladders case. Using scalar multiplications to compute scalar multiplications may seem circular, but we distinguish between non-secure and secure scalar multiplications. So, in the following, we assume that non-secure scalar multiplications (on public scalars f_P, f_Q, f_A, g_P, g_Q, g_A, ℓ_P, and ℓ_A) can be used to compute secure scalar multiplications (on sensitive scalars k).

As in Section 5, we provide concrete solutions to the ladder equations. In this section, we generalize the algorithmic strength of the Montgomery ladder by solving (Subsections 6.1 to 6.3) the ladder equations for the scalar multiplication used in ECC. We discuss in Subsection 6.4 the feasibility and performance of the novel solutions in the context of ECC. Then, in Subsection 6.5, we compare the complexity of the novel algorithms with the complexity of the Montgomery ladder. Results are summarized in Subsection 6.6.

6.1 Both Semi- and Fully-Interleaved Ladders

We start by solving the ladder equations which are common to the semi- and fully-interleaved cases. Equation (1b) (resp. Equation (2b)) $f(x, \ell(x)) = \theta(x)$ and Equation (1c) (resp. Equation (2c)) $f(\ell(x), x) = \ell(\varepsilon(x))$ for the semi- (resp. fully-) interleaved cases imply that:

$$\begin{cases} f_P + f_Q \ell_P = 2 \\ f_Q = (2 - f_P)\ell_P \end{cases}$$

So $(2 - f_P)(\ell_P^2 - 1) = 0$. If $f_P = 2$, then $f_Q = 0$, so we assume $f_P \neq 2$. To make no assumption about the finite field, we assume for the following $\ell_P = \pm 1$. The equations also imply that:

$$\begin{cases} f_Q \ell_A + f_A = 1 \\ f_A = (2 - f_P)\ell_A - \ell_A \end{cases}$$

So $(2 - f_P)(\ell_P + 1)\ell_A = 1 + \ell_A$.

- If $\ell_P = -1$ then $\ell_A = -1$, $f_Q = f_P - 2$ and $f_A = (f_P - 2) + 1$, thus the candidate ladder functions are:

$$f(P,Q) = f_P(P + Q + A) - (2Q + A)$$
$$\ell(P) = -(P + A)$$

- If $\ell_P = 1$, then by assuming that $3 - 2f_P$ is invertible we obtain $\ell_A = \frac{1}{3-2f_P}$, $f_Q = 2 - f_P$ and $f_A = 1 - \frac{2-f_P}{3-2f_P}$, thus the candidate ladder functions are:

$$f(P,Q) = f_P(P - Q) + 2Q + \frac{1 - f_P}{3 - 2f_P} A$$
$$\ell(P) = P + \frac{1}{3 - 2f_P} A$$

6.2 Semi-Interleaved Ladder Solution

In this subsection we solve the ladder equations for the semi-interleaved case. The remaining Equation (1a) $\varepsilon(\ell(x)) = \ell(\theta(x))$ for the semi-interleaved ladders is already satisfied for the $\ell_P = -1$ case. For the $\ell_P = 1$ case, it implies that $(\frac{1}{3-2f_P} - 1)A = \mathcal{O}$, thus to avoid constraint on A we assume $f_P = 1$.

Theorem 4. *The double-and-add algorithm with $\theta(P) = 2P + A$ and $\varepsilon(P) = 2P$ is semi-ladderizable with two solutions:*

1.

$$f(P,Q) = f_P(P+Q+A) - (2Q+A)$$
$$\ell(P) = -(P+A)$$

where $f_P \neq 0, 2$, producing (after memoization of one doubling in R) Algorithm 15, and:

2.

$$f(P,Q) = P+Q$$
$$\ell(P) = P+A$$

producing Algorithm 5 p.7.

Note that the second solution is actually the common Montgomery ladder for the scalar multiplication (Subsection 2.4). It was automatically generated in [11] by solving the ladder equations, confirming that our solutions are generalizations of the Montgomery ladder.

Note also that in the first solution there is no constraint on f_P, which thus can be randomly generated (Line 3) in $\mathbb{F}_p$ while satisfying constraints from Theorem 4. As in Subsection 5.2, this provides a blinding integer which hinders advanced side-channel attacks like CPA [10]. Because R depends only on f_P and A, it can be precomputed before the loop (Line 4). We discuss in Subsection 6.4 the feasibility and performance of the scalar multiplications by f_P involved in the algorithm.

Finally, note that the case $f_P = 1$ corresponds to the ladder function $f(P,Q) = P - Q$, similar to the Montgomery ladder except that the point Q has an opposite sign.

6.3 Fully-Interleaved Ladder Solution

In this subsection we solve the ladder equations in the fully-interleaved case. Remain two equations. In the case $\ell_P = -1$, Equation (2a) $g(\theta(x), \ell(x)) = \ell(\theta(x))$ for the fully-interleaved ladders is equivalent to:

$$\begin{cases} 2g_P - g_Q = -2 \\ g_P - g_Q + g_A = -2 \end{cases}$$

So $g_Q = 2(1+g_P)$ and $g_A = g_P$. Thus, Equation (2d) $g(f(\ell(x),x),x) = \varepsilon(x)$ for the fully-interleaved ladders implies that $f_P g_P = 0$. Because we assumed $f_P, g_P \neq 0$ and we look for a solution independent from the chosen field for the coefficients, we have no solution for this case.

Algorithm 15 Semi-Interleaved Ladder for the Scalar Multiplication

input public p, A; secret k
1: $P \leftarrow \mathcal{O}$
2: $Q \leftarrow -A$
3: $f_P \leftarrow \text{random}(\mathbb{F}_p \setminus \{0,2\})$
4: $R \leftarrow (f_P - 1)A$
5: **for** $i = d$ **to** 0 **do**
6: **if** $k[i] = 1$ **then**
7: $S \leftarrow 2Q$
8: $P \leftarrow f_P(P+Q) + R - S$
9: $Q \leftarrow S$
10: **else**
11: $S \leftarrow 2P$
12: $Q \leftarrow f_P(Q+P) + R - S$
13: $P \leftarrow S$
14: **end if**
15: **end for**
16: **return** P
output $P = kA$

In the case $\ell_P = 1$, Equation (2a) $g(\theta(x), \ell(x)) = \ell(\theta(x))$ for the fully-interleaved ladders is equivalent to:

$$\begin{cases} 2g_P + g_Q = 2 \\ g_P + \dfrac{g_Q}{3-2f_P} + g_A = 1 + \dfrac{1}{3-2f_P} \end{cases}$$

So $g_Q = 2(1-g_P)$ and $g_A = 1 + \frac{(2f_P-1)g_P - 1}{3-2f_P}$. Thus Equation (2d) $g(f(\ell(x),x),x) = \varepsilon(x)$ for the fully-interleaved ladders implies that $\frac{2(f_P(g_P-1)+1)}{3-2f_P} A = \mathcal{O}$. To avoid constraint on A, we assume $f_P(g_P - 1) + 1 = 0$; i.e., $g_P = 1 - \frac{1}{f_P}$.

Theorem 5. *The double-and-add algorithm with $\theta(P) = 2P + A$ and $\varepsilon(P) = 2P$ is fully-ladderizable with:*

$$f(P,Q) = f_P\left(P - Q - \tfrac{1}{3-2f_P}A\right) + 2Q + \tfrac{1}{3-2f_P}A$$
$$g(P,Q) = P + \tfrac{1}{f_P}\left(-P + 2Q + \tfrac{1}{3-2f_P}A\right) - \tfrac{1}{3-2f_P}A$$
$$\ell(P) = P + \tfrac{1}{3-2f_P}A$$

where $f_P \neq 0, 1, 2$, and where f_P and $3 - 2f_P$ are invertible, producing (after memoization in R and S) Algorithm 16.

Note that, because the coefficients are chosen in $\mathbb{F}_p$ with p prime, if $f_P \neq 0$ and $f_P \neq 3 \times 2^{-1}$, then f_P and $3 - 2f_P$ are invertible. Hence, f_P can be

Algorithm 16 Fully-Interleaved Ladder for the Scalar Multiplication

input public p, A; secret k

1: $P \leftarrow \mathcal{O}$
2: $f_P \leftarrow \text{random}(\mathbb{F}_p \setminus \{0, 1, 2, 3 \times 2^{-1}\})$
3: $C \leftarrow \frac{1}{3-2f_P} A$
4: $c_0 \leftarrow \frac{1}{f_P}$
5: $Q \leftarrow C$
6: **for** $i = d$ **to** 0 **do**
7: **if** $k[i] = 1$ **then**
8: $R \leftarrow P - C$
9: $S \leftarrow 2Q + C$
10: $P \leftarrow f_P (R - Q) + S$
11: $Q \leftarrow c_0 (S - P) + R$
12: **else**
13: $R \leftarrow Q - C$
14: $S \leftarrow 2P + C$
15: $Q \leftarrow f_P (R - P) + S$
16: $P \leftarrow c_0 (S - Q) + R$
17: **end if**
18: **end for**
19: **return** P

output $P = kA$

randomly generated (Line 2) while satisfying constraints from Theorem 5. As in Subsection 5.2, this provides a blinding integer reducing vulnerability to advanced side-channel attacks like CPA [10]. Because C and c_0 depends only on f_P and A, they can be precomputed before the loop (Lines 3 and 4). We discuss in Subsection 6.4 the feasibility and performance of the scalar multiplications by f_P and c_0 involved in the algorithm.

6.4 Application to ECC

In this subsection, we discuss the feasability of the novel algorithms obtained for the scalar multiplication in ECC. These algorithms are the first semi-interleaved ladder in Theorem 4 requiring a scalar multiplication by f_P, and the fully-interleaved ladder in Theorem 5 requiring a scalar multiplication by $\frac{1}{3-2f_P}$ during initialization, then f_P and $\frac{1}{f_P}$ during each iteration. These scalar multiplications can be costly, so they are a challenge regarding performance (**RQ3**).

Scalar multiplications cA for various scalar coefficients c can be precomputed if A is a fixed point, as in the DSA signing step or the first step of Diffie-Hellman protocol. This is the case for the scalar multiplication by $\frac{1}{3-2f_P}$, but not for the other cases, which involve scalar multiplications for non-fixed points R, linear combination of A, P, and Q.

A scalar multiplication cR for a specific coefficient c can be efficiently computed, i.e., with very few field operations, by using an elliptic curve endowed with an efficient endomorphism. On such curves, there exists a constant λ, depending on the prime modulus p and the parameters of the curve (Definition 1 p.6), such that λR can be computed with very few multiplications (only one in most cases) on $\mathbb{F}_p$. This is detailed in [16], where the authors describe the GLV (Gallant, Lambert, and Vanstone) method used in various standards, like the Bitcoin protocol specification or TLS (Transport Layer Security) [41].

Another way to optimize the cost of cR is to use a short addition chain [29], so that the number of point operations required to compute cR is equal to the length of the chain. The problem of computing the shortest addition chain for a given integer is hard [3], but for "small" integers there exists tables [8] providing the corresponding chain.

So, for the first semi-interleaved ladder in Theorem 4, one can use an elliptic curve endowed with an efficient endomorphism λ and assign $f_P = \lambda$. Or, for a non-specific curve, one can choose f_P amongst the integers corresponding to "very short" addition chains.

But these methods should not be used together. Indeed, when an efficient endomorphism is available for the chosen elliptic curve, multiplications on $\mathbb{F}_p$ are largely less costly than the additions and doublings required in addition chains, even very short ones. Hence, an attacker would be able to infer the method used to choose f_P, by leveraging even simple side-channel attacks [32, 31].

Moreover, there usually exists (if any) at most one or two (known) efficient endomorphism for a given elliptic curve [16]. Hence, because of the small range of values, efficient endomorphisms cannot be used as blinding integers.

Thus, to preserve f_P as a blinding integer for the semi-interleaved ladder (Algorithm 15), it must be randomly chosen amongst values corresponding to very short addition chains. This largely reduces the range for the random blinding integer, but this

makes this algorithm practical in the context of ECC.

For the fully-interleaved ladder (Theorem 5), neither an efficient endomorphism nor an addition chain can be used. Indeed, that $f_P R$ can be efficiently computed does not imply that $\frac{1}{f_P} R$ can also be efficiently computed, because both coefficients are dependent. Therefore, Algorithm 16 is, to our knowledge, not feasible in practice in the context of ECC.

6.5 Complexity

We remind that we consider solutions involving non-secure scalar multiplications with public scalars in order to compute secure scalar multiplications with sensitive scalars. We provide in Table 4 a comparison of complexities for the Montgomery, semi- and fully-interleaved ladders. As in Subsection 5.4, we consider both sequential and parallel cases, and the complexities are given in terms of cost per key bit, where M stands for non-secure scalar multiplication, D for doubling, and A for addition/subtraction of points of the elliptic curve.

The line "without optimization" of Table 4 demonstrates a trade-off between security and performance. As opposed to the Montgomery ladder, the semi-interleaved solution from Theorem 4 has a blinding integer reducing the vulnerability of intermediate values against advanced side-channel attacks, but at the price of a non-secure scalar multiplication for each key bit. Moreover, as illustrated in Table 1 p.14, the fully-interleaved solution from Theorem 5 is better protected against fault-injection attacks, but this time at the price of two non-secure scalar multiplications for each key bit.

The cost of the semi-interleaved ladder solution is detailed at the third column of Table 4. Since the cost of a few multiplications on $\mathbb{F}_p$ is negligible compared to point doubling or addition/subtraction, if an efficient endomorphism is available, this solution is almost as performant as the Montgomery ladder. But, as detailed in Subsection 6.4, such an efficient endomorphism cannot be used for the blinding integer. Hence, in this case, the Montgomery ladder dominates our semi-interleaved ladder solution. With short addition chains, the cost depends on the constant f_P and is denoted $cost_{f_P}(D, A)$, consisting on (a small number of) doubling and addition/subtraction operations. This time, as detailed in Subsection 6.4, a blinding integer can be used but at the price of several operations, demonstrating a trade-off between security and performance.

The cost of the fully-interleaved ladder solution is detailed at the last column of Table 4. Because of the dependency between f_P and $\frac{1}{f_P}$, determining an efficient endomorphism or a short addition chain for both coefficients is an open problem.

Finally, as in Subsection 5.4, we also consider parallel computations (Subsection 2.5) to evaluate performance. The second part of Table 4 provides a comparison when the algorithms run on a multi-core system. We consider the `then` branch of our semi-interleaved ladder (Algorithm 15), both branches having the same complexity. First, the doubling $S \leftarrow 2Q$ is computed. Then, $f_P(P + Q)$ and $R - S$ can be independently computed in one scalar multiplication and one addition. Finally the results are added together and Q receives the value from S.

In the same way, we consider the `then` branch of our fully-interleaved ladder (Algorithm 16). R and S can be independently computed in one doubling and one addition, gaining one addition, then P and Q are sequentially computed. In both cases, only one addition operation is gained, which does not change our conclusions from the sequential case.

6.6 Results for the Scalar Multiplication

In this section we provided concrete answers to **RQ1** by solving the ladder equations for the scalar multiplication in ECC. We obtained Algorithm 15 for the semi-interleaved case and Algorithm 16 for the fully-interleaved case.

Algorithm 15 uses a blinding integer, providing more protection against power side-channel attacks than the Montgomery ladder, for instance against advanced power attacks like CPA [10]. As in Subsection 5.7, this protection naturally arose from a free parameter while solving the ladder equations, thus demonstrating again the fruitfulness of the ladderization approach (**RQ2**). Moreover, as detailed in Subsection 4.3, Algorithm 16 is more secure against fault-injection attacks than the Montgomery ladder. So, the security properties of the Montgomery ladder have not only been generalized but improved in both cases.

Table 4: Cost per bit (after memoization) of the Montgomery (Algorithm 5 p.7), semi- (Algorithm 15) and fully-interleaved ladders (Algorithm 16) for the secure scalar multiplication, where M stands for non-secure scalar multiplication, A for addition or subtraction, and D for doubling

	Montgomery	Semi-Interleaved	Fully-Interleaved
sequential			
without optimization	$A + D$	$M + 3A + D$	$2M + 6A + D$
efficient endomorphism	$A + D$	$3A + D$	open problem
addition chain	$A + D$	$cost_{f_P}(D, A) + 3A + D$	open problem
parallel			
without optimization	$A + D$	$M + 2A + D$	$2M + 5A + D$
efficient endomorphism	$A + D$	$2A + D$	open problem
addition chain	$A + D$	$cost_{f_P}(D, A) + 2A + D$	open problem

We also compared the complexity of these novel algorithms in Subsection 6.5 with the complexity of the Montgomery ladder in ECC (Algorithm 5 p.7). The trade-off between security and performance is less favorable than in Subsection 5.4, because we use scalar multiplications on public values to compute scalar multiplications on the secret key.

These scalar multiplications raise a challenge regarding performance (**RQ3**), that we tackled in two ways. First, by leveraging (if any) an efficient endomorphism to compute these scalar multiplications. In that case, a blinding integer cannot be used and performance is no better than the common Montgomery ladder, which thus dominates our solution. Second, by restricting the blinding integer in Algorithm 15 to integers computed using short addition chains. This makes Algorithm 15 feasible, but at the price of limiting the range of values for the blinding integer and thus the protection against advanced power attacks. Finally, efficient endomorphisms or addition chains were not applicable to Algorithm 16 which is thus, to our knowledge, not feasible in practice in the context of ECC.

7 Conclusion

A ladderization is a process refactoring a sensitive iterative conditional branching into a semi- or fully-interleaved ladder, in order to protect sensitive information from various side-channel or fault-injection attacks. These interleaved ladders are obtained through solving ladder equations for the system of interest. We evaluated the ladderization approach with three research questions:

- **RQ1**: To which extent can the algorithmic strength of the Montgomery ladder be generalized to other algorithms?
- **RQ2**: How secure are interleaved ladders compared to the Montgomery ladder?
- **RQ3**: How feasible are interleaved ladders in practice and, if so, how performant are they compared to the Montgomery ladder?

7.1 Answers to the Research Questions

Theorem 1 provided a theoretical answer to **RQ1**, by characterizing the ladderizable algorithms. To provide a more concrete answer, we solved the ladder equations in Section 5 for the modular exponentiation and in Section 6 for the scalar multiplication. Security properties from the Montgomery ladder are preserved. First, the obtained code symmetry allows a protection against simple side-channel attacks like timing attacks [32] or SPA [31]. Second, a violation of the invariant $y = \ell(x)$ allows the algorithm to detect a fault-injection attack, potentially triggering a self-secure countermeasure.

RQ2 is answered in two ways. Exhaustively investigating fault-propagation patterns lead to the discovery of three novel fault-injection attacks described in Section 4. Moreover, interleaved ladders can be even more secure than the Montgomery ladder. First, as described in Table 1, fully-interleaved ladders are more secure than the Montgomery ladder against fault-injection attacks. Second, ladder equations in Subsections 5.2, 5.3 and 6.2 allows for all solutions except the fully-interleaved ladder

for scalar multiplication the use of blinding integers, which provide protection against more complex side-channel attacks like CPA [10].

But this increased protection came at the price of performance, which raised several challenges regarding feasibility (**RQ3**).

In the case of modular exponentiation, our semi-interleaved solution is 3 (resp. 2) times more costly using sequential (resp. parallel) operations, and our fully-interleaved solution is 5 (resp. 3) more costly than the Montgomery ladder. But we also demonstrated that, in practice, our semi-interleaved solution is only 1.4 times more costly than the Montgomery ladder, indicating that for several cores and large key sizes their performances are similar. Moreover, our fully-interleaved solution requires a ladder constant, which could be costly to obtain in general. Fortunately, we proved in the case of the RSA [45] (Subsection 5.5) and DSA [40] (Subsection 5.6) cryptosystems that this ladder constant can be obtained at a small cost with probability almost equal to 1.

In the case of scalar multiplication, one or two non-secure scalar multiplication(s) per secret bit are required to compute one secure scalar multiplication, which is tremendously costly. To tackle this challenge in the context of ECC, we investigated efficient endomorphisms. Unfortunately, this solution was not as performant as the Montgomery ladder, while providing the same security properties, hence has to be rejected. We investigated another solution, that restricts the blinding integer in Algorithm 15 to integers computed using short addition chains. We proved that in this case the semi-interleaved ladder is feasible in practice, but at the price of limiting the range of values for the blinding integer and thus the protection against advanced power attacks. But such a solution was not applicable to the fully-interleaved ladder, which is thus, to our knowledge, not feasible in practice in the context of ECC.

7.2 Future Work

The feasibility of the fully-interleaved ladder in the context of ECC depends on the efficient computation of several dependent coefficients for the scalar multiplications, which is an open and interesting problem.

Moreover, our approach has been already automated [11] for the semi-interleaved ladders in the `if-then` case, which remains to be done in the general case for both semi- and fully-interleaved ladders.

Finally, we only investigated the univariate case for conditional branching, but the multivariate case may be of interest. For instance, a multi-variable polynomial $\sum_{0\leq n\leq d}\sum_{n_1+\cdots+n_k=n} c_{n_1,\ldots,n_k}\prod_{1\leq i\leq k} x_i^{n_i}$ can be represented by a multidimensional array of coefficients, thus the manipulation of ladder equations could be handled using matrix operations.

Acknowledgments

This work was supported by the EU Horizon 2020 project *TeamPlay* (`https://www.teamplay-h2020.eu`), grant number 779882.

References

[1] Samuel Antão, Jean-Claude Bajard, and Leonel Sousa. Rns-based elliptic curve point multiplication for massive parallel architectures. *The Computer Journal*, 55(5):629–647, 2012.

[2] Jean Arlat. *Validation de la sûreté de fonctionnement par injection de fautes : méthode, mise en oeuvre, application.* PhD thesis, Institut national polytechnique (Toulouse, France), 1990.

[3] Hatem M. Bahig. Improved generation of minimal addition chains. *Computing*, 78(2):161–172, 2006.

[4] Elaine B. Barker, Lidong Chen, Allen L. Roginsky, Richard Davis, and Scott Simon. *Recommendation for Pair-Wise Key Establishment Using Integer Factorization Cryptography.* Number 800-56B Rev. 2 in Special Publication. NIST SP, , March 2019.

[5] Elaine B. Barker, LiLy Chen, Allen L. Roginsky, Apostol Vassilev, and Richard Davis.

Recommendation for Pair-Wise Key Establishment Using Dicrete Logarithm Cryptography. Number 800-56A Rev. 3 in Special Publication. NIST SP, , April 2018.

[6] Saikat Basu. A new parallel window-based implementation of the elliptic curve point multiplication in multi-core architectures. *Group*, 16(4a3):27b2, 2012.

[7] Daniel J Bernstein, Hsueh-Chung Chen, Ming-Shing Chen, Chen-Mou Cheng, Chun-Hung Hsiao, Tanja Lange, Zong-Cing Lin, and Bo-Yin Yang. The billion-mulmod-per-second pc. In *Workshop record of SHARCS*, volume 9, pages 131–144, 2009.

[8] Olivier Billet. Addition chains (accessed may 9, 2023).

[9] Arnaud Boscher, Robert Naciri, and Emmanuel Prouff. CRT RSA Algorithm Protected Against Fault Attacks. In Damien Sauveron, Konstantinos Markantonakis, Angelos Bilas, and Jean-Jacques Quisquater, editors, *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, pages 229–243, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[10] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*, pages 16–29. Springer, 2004.

[11] Christopher Brown, Adam D. Barwell, Yoann Marquer, Olivier Zendra, Tania Richmond, and Chen Gu. Semi-automatic ladderisation: Improving code security through rewriting and dependent types. In *Proceedings of the 2022 ACM SIGPLAN International Workshop on Partial Evaluation and Program Manipulation*, PEPM 2022, page 14–27, New York, NY, USA, 2022. Association for Computing Machinery.

[12] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Square Always Exponentiation. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *Progress in Cryptology – INDOCRYPT 2011*, pages 40–57, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[13] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman and Hall/CRC, , 2005.

[14] Jean-Sébastien Coron. Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems*, pages 292–302, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[15] Junfeng Fan, Kazuo Sakiyama, and Ingrid Verbauwhede. Elliptic curve cryptography on embedded multicore systems. *Design Automation for Embedded Systems*, 12:231–242, 2008.

[16] Robert P. Gallant, Robert J. Lambert, and Scott Alexander Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In *Advances in Cryptology — CRYPTO*, volume 2139 of *LNCS*, pages 190–200, , 2001. Springer.

[17] Carl Friedrich Gauss and Arthur A. Clarke. *Disquisitiones Arithmeticae*. Yale University Press, , 1965.

[18] C. Giraud. An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. *IEEE Transactions on Computers*, 55(9):1116–1120, Sep. 2006.

[19] GMP. Gnu multiple precision arithmetic library (gmp) (accessed august 6, 2023).

[20] JaeCheol Ha, YongJe Choi, Dooho Choi, and Hoonjae Lee. Power Analysis Attacks on the Right-to-Left Square-Always Exponentiation Algorithm. *J. Internet Serv. Inf. Secur.*, 4:38–51, 2014.

[21] Neil Hanley, HeeSeok Kim, and Michael Tunstall. Exploiting Collisions in Addition Chain-Based Exponentiation Algorithms Using a Single Trace. In Kaisa Nyberg, editor, *Topics in Cryptology — CT-RSA 2015*, pages 431–448, Cham, 2015. Springer International Publishing.

[22] Marc Joye. Highly Regular Right-to-Left Algorithms for Scalar Multiplication. In Pascal Paillier and Ingrid Verbauwhede, editors,

Cryptographic Hardware and Embedded Systems - CHES 2007, pages 135–147, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[23] Marc Joye. Highly Regular m-Ary Powering Ladders. In Michael J. Jacobson, Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, pages 350–363, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[24] Marc Joye and Sung-Ming Yen. The Montgomery Powering Ladder. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 291–302, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[25] Uma S. Kanniah and Azman Samsudin. Multithreading elliptic curve cryptosystems. In *2007 IEEE International Conference on Telecommunications and Malaysia International Conference on Communications*, pages 134–139, May 2007.

[26] Auguste Kerckhoffs. La cryptographie militaire (military cryptography). *Sciences Militaires (J. Military Science, in French)*, 9:5 – 38, January 1883.

[27] HeeSeok Kim, Tae Hyun Kim, Joong Chul Yoon, and Seokhie Hong. Practical Second-Order Correlation Power Analysis on the Message Blinding Method and Its Novel Countermeasure for RSA. *ETRI Journal*, 32(1):102–111, 2010.

[28] Ágnes Kiss, Juliane Krämer, Pablo Rauzy, and Jean-Pierre Seifert. Algorithmic Countermeasures Against Fault Attacks and Power Analysis for RSA-CRT. In François-Xavier Standaert and Elisabeth Oswald, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 111–129, Cham, 2016. Springer International Publishing.

[29] Donald Ervin Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 2. Addison Wesley, , 3rd edition, 07 1997.

[30] N. Koblitz. Elliptic Curve Cryptosystems. *Math. Comp*, 48:243–264, 01 1987.

[31] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[32] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[33] Peter L. Montgomery. Montgomery, P.L.: Speeding the Pollard and Elliptic Curve Methods of Factorization. Math. Comp. 48, 243-264. *Mathematics of Computation - Math. Comput.*, 48:243–243, 01 1987.

[34] Ladder benchmarks (accessed august 6, 2023).

[35] Duc-Phong Le, Chik How Tan, and Michael Tunstall. Randomizing the Montgomery Powering Ladder. In Raja Naeem Akram and Sushil Jajodia, editors, *Information Security Theory and Practice*, pages 169–184, Cham, 2015. Springer International Publishing.

[36] Yoann Marquer. Algorithmic Completeness of Imperative Programming Languages. *Fundamenta Informaticae*, 168(1):51–77, July 2019.

[37] Yoann Marquer and Tania Richmond. A hole in the ladder: Interleaved variables in iterative conditional branching. In *ARITH 2020 - 27th IEEE Symposium on Computer Arithmetic*, pages 56–63, Portland, Oregon, USA, United States, June 2020. IEEE.

[38] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power Analysis Attacks of Modular Exponentiation in Smartcards. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems*, pages 144–157, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[39] Victor S. Miller. Use of Elliptic Curves in Cryptography. In Hugh C. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, pages 417–426, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[40] National Institute of Standards and Technology. Digital signature standard (DSS). Technical Report Federal Information Processing Standards Publications (FIPS PUB) 186-4, U.S. Department of Commerce, Washington, D.C., July 2013.

[41] Yoav Nir, Simon Josefsson, and Manuel Pégourié-Gonnard. RFC8422: Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS) versions 1.2 and earlier, 2020.

[42] Openmp api (accessed august 6, 2023).

[43] M. Petrvalsky, T. Richmond, M. Drutarovsky, P. Cayrel, and V. Fischer. Differential power analysis attack on the secure bit permutation in the McEliece cryptosystem. In *2016 26th International Conference Radioelektronika (RADIOELEKTRONIKA)*, pages 132–137, April 2016.

[44] E. Rescorla. RFC2631: Diffie-Hellman key agreement method, 1999.

[45] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[46] EFD Web site. Explicit formulas database.

[47] Falko Strenzke, Erik Tews, H. Gregor Molter, Raphael Overbeck, and Abdulhadi Shoufan. Side Channels in the McEliece PKC. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography*, pages 216–229, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[48] Yen Sung-Ming, Seungjoo Kim, Seongan Lim, and Sangjae Moon. A Countermeasure against One Physical Cryptanalysis May Benefit Another Attack. In Kwangjo Kim, editor, *Information Security and Cryptology — ICISC 2001*, pages 414–427, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[49] C. D. Walter. The Montgomery and Joye Powering Ladders are Dual. *IACR ePrint Archive*, 1081:1–6, 2017.

[50] H. Ziade, R. Ayoubi, and R. Velazco. A survey on fault injection techniques. *International Arab Journal of Information Technology*, Vol. 1, No. 2, July:171–186, 2004.