A Hole in the Ladder:
Interleaved Variables in Iterative Conditional Branching

Yoann Marquer
TAMIS team, TeamPlay project
Inria, Univ Rennes, CNRS, IRISA
Rennes, France
yoann.marquer@inria.fr

Tania Richmond
TAMIS team, TeamPlay project
Inria, Univ Rennes, CNRS, IRISA
Rennes, France
tania.richmond@inria.fr

***Abstract*—The modular exponentiation is crucial to the RSA cryptographic protocol, and variants inspired by the Montgomery ladder have been studied to provide more secure algorithms. In this paper, we abstract away the iterative conditional branching used in the Montgomery ladder, and formalize systems of equations necessary to obtain what we call the semi-interleaved and fully-interleaved ladder properties. In particular, we design fault-injection attacks able to obtain bits of the secret against semi-interleaved ladders, including the Montgomery ladder, but not against fully-interleaved ladders that are more secure. We also apply these equations to extend the Montgomery ladder for both the semi- and fully-interleaved cases, thus proposing novel and more secure algorithms to compute the modular exponentiation.**

***Index Terms*—Cryptography, Countermeasures (computer), Fault detection, Iterative algorithms, Public-key cryptography, Security, Side-channel attacks**

I. Introduction

A. Contribution

We present common algorithms used to compute the modular exponentiation, based on an *iterative conditional branching*, where a conditional branching depending on the secret updates the value of a variable x on every iteration of one/several loop(s). Amongst these algorithms the Montgomery ladder, that uses a fresh variable y, satisfies desirable properties against side-channel or fault-injection attacks.

In this paper, we formalize these properties with equations corresponding to two families of cases: the *semi-interleaved ladders* where the value of x or y may (depending on the secret) depend only on its previous value, and the *fully-interleaved ladders* where the value of x and y both depend on both previous values.

We use our formalization to design novel algorithms for the modular exponentiation, including a semi-interleaved ladder with a mask updated at every iteration, and a fully-interleaved ladder. We also propose an attacker model using fault-injection able to obtain some bits of the secret in the semi-interleaved cases (including the Montgomery ladder) but none in the fully-interleaved cases, and a stronger variant of the attacker model able to obtain all bits of the secret in the semi-interleaved cases. The considered attacks cannot be used against the fully-interleaved cases, but other attacks might exist.

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

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 $x = a^k \bmod n$

TABLE I
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: $\quad$ **else**
7: $\quad\quad y \leftarrow ax \bmod n$
8: $\quad$ **end if**
9: **end for**
10: **return** x
output $x = a^k \bmod n$

TABLE II
Square and Multiply Always

B. Organization of the Paper

As introduction, we present in Section II some related works on the modular exponentiation and the Montgomery ladder. In Section III we formalize the iterative conditional branching, and we deduce the equations satisfied by the semi-interleaved and fully-interleaved ladders, and thus the requirements for ladderizable programs. In Section IV we introduce two attacker models using fault injection techniques, and compare the vulnerability of the non-, semi- and fully-interleaved ladders. Finally, in Section V we detail how to produce examples of the semi- and fully-interleaved ladders, including novel variants of the Montgomery ladder.

II. Related Works

In this section, we introduce known algorithms for the modular exponentiation, their relevance regarding security, and the desirable properties of the usual Montgomery ladder.

A. Modular Exponentiation

Let k be a secret key, and $k = \sum_{0 \leq i \leq d} k[i]\, 2^i$ be its binary expansion of size $d+1$, i.e. $k[i]$ is the bit i of k. The *square-and-multiply* algorithm described in Table I computes the (left-to-right) modular exponentiation $a^k \bmod n$, by using $a^{\sum_{0 \leq i \leq d} k[i] 2^i} = \prod_{0 \leq i \leq d} (a^{2^i})^{k[i]}$. This exponentiation is commonly used in crypto-systems like RSA [1].

For every iteration, the multiplication ax is computed only if $k[i] = 1$, which can be detected by observing execution time[1] [2] or power profiles[2] by means of e.g. SPA (Simple

[1]Even observing the duration of the whole execution can leak the Hamming weight of the secret key, and thus can narrow down the exploration space.

[2]A multiplication can be distinguished from a squaring, hence variants with squaring only have been proposed in [3] and improved in [4].

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

TABLE III
MONTGOMERY LADDER

```
1: x ← init
2: for i = 1 to n do
3:     ⋱
4:     if secret then
5:         x ← φ(x)
6:     else
7:         x ← ψ(x)
8:     end if
9:     ⋰
10: end for
```

TABLE IV
ITERATIVE CONDITIONAL BRANCHING

```
1: assert(secret ≤ bound)
2: for i = 0 to secret do
3:     ⋮
4: end for
```

```
1: for i = 0 to bound do
2:     if i ≤ secret then
3:         ⋮
4:     end if
5: end for
```

TABLE V
LOOP BOUNDED BY A SENSITIVE VARIABLE

```
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 ← ψ(y)
8:     else
9:         y ← f(y, x)
10:        x ← ψ(x)
11:    end if
12:    ...
13: end for
```

TABLE VI
SEMI-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
```

TABLE VII
FULLY-INTERLEAVED LADDERS

Power Analysis[3]) [8], and thus leads to information leakage from both time and power side-channel attacks.

To prevent SPA, regularity of the modular exponentiation algorithms is required, which means that both branches of the sensitive conditional branching perform the same operations, independently from the value of the exponent. Thus, an `else` branch is added with a dummy instruction [6] in the *square-and-multiply-always* algorithm described in Table II.

But countermeasures developed against a given attack may benefit another one [9]. Because the multiplication in the `else` branch of this algorithm is a dummy operation, a fault injected [10] in the register containing ax will eventually propagate through successive iterations and alter the final result only if $k[i] = 1$, thus leaking information. Therefore, an attacker (see the attacker model in Section IV) able to inject a fault in a given register at a given iteration can obtain the digits of the secret key by comparing the final output with or without fault, technique known as safe-error attack.

B. Montgomery Ladder

This is not the case in the algorithm proposed by Montgomery [11] and described in Table III, where a fault injected in a register will eventually propagate to the other one, and thus will alter the final result, preventing the attacker to obtain information. But, as described in Section IV, some information on the last digits may still leak, weakening the protection obtained from the ladder.

The Montgomery ladder is algorithmically equivalent [12] to the square-and-multiply(-always) algorithm(s), in the sense that x has the same value for every iteration. Actually, some variants [13] of the square-and-multiply-always algorithm[4] may be as resistant as the Montgomery ladder [17], both by checking invariants [18] violated if a fault is injected. In the case of the Montgomery ladder, the invariant $y = ax$ is satisfied for every iteration. These invariants are important for the self-secure exponentiation countermeasures [19].

Note that the `else` branch in Table III is identical to the `then` branch, except that x and y are swapped, which provides also (partial[5]) protection against timing and power leakage. Moreover, the variable dependency makes these variables interleaved, so this exponentiation is algorithmically (but partially) protected against safe-error attacks. Finally, as opposed to square-and-multiply-always in Table II, the code in the `else` branch is not dead, so will not be removed by compiler optimisations.

III. LADDER EQUATIONS

In this section, we formalize the iterative conditional branching occuring in algorithms like the Montgomery ladder, and deduce the requirements to optimize them with semi- or fully-interleaved ladders.

A. Iterative Conditional Branching

In this paper, we focus on programs as in Table IV called *iterative conditional branching*. It appears in programs like the modular exponentiation, its counterpart the double-and-add used for elliptic curve point multiplication [20], [21], or the secure bit permutation in the McEliece cryptosystem [22] attacked in [23]. It also appears naturally (Table V) when trying to turn a loop on the secret into a conditional branching depending on the secret, that can itself be balanced to remove or reduce the dependency on the secret.

But our approach does not depend on the number/depth of the considered loops, hence the dots in Table IV. We assume only that the conditional branching uses only one variable x, the multivariate case being future work (see Section VI).

Definition 1 (Iterative Conditional Branching)**.** A program as in Table IV is said with an (univariate) *iterative conditional branching* with two (unary) functions φ and ψ.

B. Semi-Interleaved Ladders

To prevent information leakage from side-channels or fault injections, we use another variable y in the algorithm described in Table VI. As in the Montgomery ladder in Table III, we need to find two functions ℓ and f such that for every iteration:

[3]The power profile depends also on the values in the considered registers, so computing a multiplication in every case is better against SPA but not against CPA (Correlation Power Analysis) [5], even if standard blinding techniques can prevent differential attacks [6], [7].

[4]See [14] for highly regular right-to-left variants, [15] for a generalization to any basis and left-to-right/right-to-left variants, and [16] for their duality.

[5]The variables x and y may have different access time, which hinders protection against timing leakage.

- $y = \ell(x)$, and
- x has the same value for every iteration as in Table IV.

$y = \ell(x)$ is satisfied at the initialization. By induction, let's assume $y = \ell(x)$ at the beginning of an iteration. In the `then` branch we have $x \leftarrow f(x, y)$ then $y \leftarrow \psi(y)$, thus in order to have $y = \ell(x)$ satisfied at the end of an iteration the following equation must hold:

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

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

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

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

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

and x is already updated to $\psi(x)$ during the iteration.

Definition 2 (Semi-Ladderizable)**.** Let P be a program with a univariate iterative conditional branching with two unary functions denoted φ and ψ. P is *semi-ladderizable* if there exists a unary function ℓ and a binary function f such that, for every considered value x:

$$\begin{cases} \psi(\ell(x)) = \ell(\varphi(x)) & (1) \\ f(x, \ell(x)) = \varphi(x) & (2) \\ f(\ell(x), x) = \ell(\psi(x)) & (3) \end{cases}$$

For the square-and-multiply algorithm we have $\varphi(x) = ax^2$ and $\psi(x) = x^2$, and we know that it can be semi-ladderized by using the Montgomery ladder with $\ell(x) = ax$ and $f(x, y) = xy$, but we show in Section V that there are other solutions. Note that to respect the form of the semi-interleaved ladder, we should have written $y \leftarrow yx$ and not $y \leftarrow xy$ in the `else` branch[6] of the Montgomery ladder.

C. Fully-Interleaved Ladders

Unfortunately, the semi-interleaved ladder is vulnerable to fault injection techniques, because in every branch at least one variable depends only on its previous value and not the previous value of both variables (see Section IV). Moreover, an attacker able to determine whether the output of one operation is used as the input to another one can [25] apply collision attacks[7] to deduce whether two following bits are the same. To solve these issues, we propose in Table VII a *fully-interleaved ladder* using three functions ℓ, f and g.

As for the semi-interleaved ladders, $y = \ell(x)$ is satisfied at the initialization. We assume again 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 an iteration the following equation must hold:

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

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

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

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

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

Definition 3 (Fully-Ladderizable)**.** Let P be a program with a univariate iterative conditional branching with two unary functions denoted φ and ψ. P is *fully-ladderizable* if there exists a unary function ℓ and two binary functions f and g such that, for every considered value x:

$$\begin{cases} g(\varphi(x), \ell(x)) = \ell(\varphi(x)) & (4) \\ f(x, \ell(x)) = \varphi(x) & (5) \\ f(\ell(x), x) = \ell(\psi(x)) & (6) \\ g(f(\ell(x), x), x) = \psi(x) & (7) \end{cases}$$

Note that Equation (2) is Equation (5) and Equation (3) is Equation (6). Without surprise, if g is chosen such that $g(x, y) = \psi(y)$ then Equation (1) is a special case of Equation (4), and Equation (7) is satisfied. Thus, semi-interleaved ladders are subcases of fully-interleaved ladders.

D. Ladderizable Programs

Theorem 4. *Let P be a program with an iterative conditional branching with two unary functions denoted φ and ψ. If P is semi-ladderizable with ℓ and f, or fully-ladderizable with ℓ, f and g, then for every iteration of the ladder variant:*

- $y = \ell(x)$
- *x is updated as in P:*

$$x \leftarrow \begin{cases} \varphi(x) & \textit{if secret} \\ \psi(x) & \textit{otherwise} \end{cases}$$

Thus, for every program with an iterative conditional branching, if there exists ℓ and f satisfying the equations in Definition 2 then the conditional branching can be rewritten as a semi-interleaved ladder, or even better if there exists ℓ, f and g satisfying the equations in Definition 3 then it can be rewritten as a fully-interleaved ladder.

Because in a semi- or fully-interleaved ladder the operations performed are the same for both the `then` and the `else` branch, this transformation is an algorithmic countermeasure against side-channel attacks [2], [8]. We detail in Section IV the impact of the semi- and fully-interleaved ladderization against fault injection techniques. Then, to demonstrate the concept, we construct examples of semi- and fully-interleaved ladders in Section V.

IV. Fault Injection

In this section, we introduce two attacker models using fault injection techniques, and compare the vulnerability of the non-, semi- and fully-interleaved ladders.

[6]The former is actually better regarding vulnerability to the M safe-error [17] or collision [24] attacks, showing that this method is good practice.

[7]A countermeasure proposed in [26] is to randomly blend variants of the ladder, or compute the exponentiation by taking a random (bounded) walk.

According to [10] 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. Hardware/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 [27] 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 system.

A. Attacker Model

For the iterative conditional branching (Table IV) and the semi- and fully-interleaved ladders (Tables VI and VII), we assume as in the Montgomery ladder (Table III) that "secret" is the condition $k[i] = 1$, where $k[i]$ is the i-th bit of the secret key k.

Definition 5 (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 register 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[8] $1, \ldots, n$, where:
  - $k[i]$ is the i-th bit of k,
  - x_i and y_i denote the values in the register x and y between the iterations $i-1$ and i.
- The attacker can $\lightning x_i$ (resp. $\lightning y_i$):
  - inject a random fault[9] (the affected variable is set to a random value) in the register of x (resp. y),
  - between the iterations $i-1$ and i.

The attacker can run a program for the square-and-multiply(-always) (Tables I and II) algorithm(s) for a given input x_{init}, obtain 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 the iterations $i-1$ and i, and obtain a value x_{fault}. If $x_{\text{fault}} = x_{\text{final}}$ then $k[i] = 0$, otherwise $k[i] = 1$. This can be done for every iteration (in any order), thus the attacker can obtain that way all the bits of the secret key.

In this algorithm, because the current value in x determines the next value in 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 that a fault in $y = ax$ propagates to x, denoted $\lightning y_i \Rightarrow \lightning x_{i+1}$, only if $k[i] = 1$. As opposed to this non-ladderized variant, in Tables VI and VII the values of x and y are interleaved, and the fault propagation patterns are the following:

1) For the semi-interleaved ladder:
$$\begin{aligned} &\lightning x_i \Rightarrow \lightning x_{i+1} \\ &\lightning y_i \Rightarrow \lightning y_{i+1} \\ &\lightning x_i \Rightarrow \lightning y_{i+1} \text{ only if } k[i] = 0 \\ &\lightning y_i \Rightarrow \lightning x_{i+1} \text{ only if } k[i] = 1 \end{aligned}$$
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.
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 prevents an attacker from obtaining the secret key, thus increases security. Thus, we reduced 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. Indeed, the attacker can $\lightning y_n$ as in Figure 1 and compare the x output. If $x_{\text{fault}} = x_{\text{final}}$ then $k[n] = 0$, otherwise $k[n] = 1$. Thus, the attacker can always obtain $k[n]$, the last bit of the secret key.

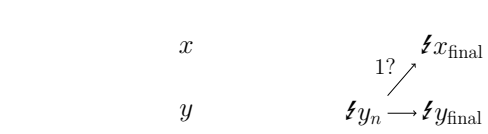

Fig. 1. Attack the Last Bit

If the obtained bit was 0 as in the left part of Figure 2, 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 the right part of Figure 2, 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.

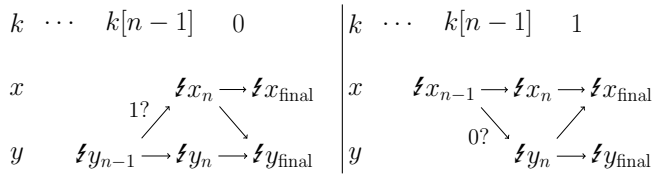

Fig. 2. Attack the Penultimate Bit

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 3 and 4. So, the attacker could obtain that way all the 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.

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

[9]In the Montgomery ladder (Table III) the invariant $y = ax$ is satisfied for every iteration. So, if a random fault is injected in x or y, the violation of the invariant allows the program to detect the fault [19] and thus to enhance the appropriate fault policy, as stopping the program or switching to a random key for the rest of the computation. This argument holds also for our semi- and fully-interleaved ladders with $y = \ell(x)$.

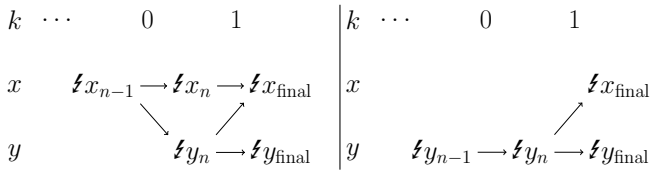

Fig. 3. $\lightning x$ or $\lightning y$ before 01

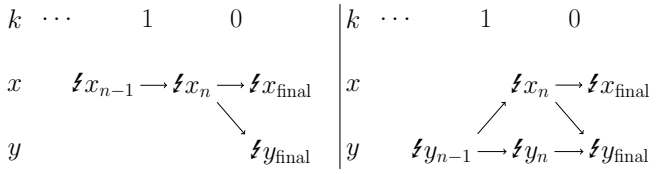

Fig. 4. $\lightning x$ or $\lightning y$ before 10

Therefore, the vulnerability to fault injection can be summarized in Table VIII. Against the attacker model described in Definition 5, fully-interleaved ladders are more secure than semi-interleaved ladders, which are more secure than without interleaving at all. But we show in the next subsection that a stronger attacker is able to obtain all the bits of the key from the semi-interleaved ladders.

B. Stronger Attacker Model

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

- The stronger attacker has the same goal and means that the attacker in Definition 5.
- The stronger attacker can also $\lightning k_{>i} = 0$ (resp. $\lightning k_{>i} = 1$):
  - *stuck-at* [10] 0 (resp. 1) all the bits, or at least those $> i$, of the register k
  - between iterations i and $i+1$.

The stronger attacker is able to break the semi-interleaved ladders by using the attack protocol described in Table IX in order to obtain all the bits of the secret key[10], where EXE($\lightning k_{>i} = 1$) at Line 4 means that the studied program is executed with inputs $x_{\text{init}}, y_{\text{init}}$ and a stuck-at $\lightning k_{>i} = 1$, and EXE($\lightning k_{>i} = 1, \lightning x_i$) at Line 5 is the same but with a fault $\lightning x_i$. In particular, if both x_{final} and y_{final} can be read, then the Montgomery Ladder (Table III) can be broken (by iterating over 0 to d). This attack does not work against fully-interleaved ladders, but other attacks might exist, and a fully-interleaved ladder is more difficult to obtain (when possible), as discussed in Subsection V-C.

V. Extend the Montgomery Ladder

The purpose of this section is to provide concrete examples of the ladder equations for cryptography, and to generalize the idea behind the Montgomery ladder to improve its protection against side-channel and fault injection attacks. To do that, we assume that $\varphi(x)$, $\psi(x)$, $f(x,y)$, $g(x,y)$ and $\ell(x)$ are quadratic polynomials with the following coefficients:

$$\begin{aligned}
\varphi(x) &= \varphi_2 x^2 \\
\psi(x) &= x^2 \\
f(x,y) &= f_{20}x^2 + f_{11}xy + f_{02}y^2 + f_{10}x + f_{01}y + f_{00} \\
g(x,y) &= g_{20}x^2 + g_{11}xy + g_{02}y^2 + g_{10}x + g_{01}y + g_{00} \\
\ell(x) &= \ell_1 x + \ell_0
\end{aligned}$$

[10]Note that the iterations are reversed in the attack protocol.

non-interleaved	All bits can be obtained.
semi-interleaved	Some bits can be obtained: • $\ldots\overline{10\ldots0}$ if x_{final} can be read • $\ldots\overline{01\ldots1}$ if y_{final} can be read
fully-interleaved	No bit can be obtained.

TABLE VIII
Vulnerability against the Attack Described for the Attacker Model from Definition 5

```
1: FIx ← 0
2: for i = n to 1 do
3:     if FIx = 1 then
4:         (x_final, y_final) ← EXE(⚡k_{>i} = 1)
5:         (x_fault, y_fault) ← EXE(⚡k_{>i} = 1, ⚡x_i)
6:         if y_fault = y_final then
7:             k[i] ← 1
8:         else
9:             k[i] ← 0
10:            FIx ← 0
11:        end if
12:    else
13:        (x_final, y_final) ← EXE(⚡k_{>i} = 0)
14:        (x_fault, y_fault) ← EXE(⚡k_{>i} = 0, ⚡y_i)
15:        if x_fault = x_final then
16:            k[i] ← 0
17:        else
18:            k[i] ← 1
19:            FIx ← 1
20:        end if
21:    end if
22: end for
23: return k
```

TABLE IX
Protocol to Attack the Semi-Interleaved Ladders

More general quadratic polynomials, e.g. $\ell_2 \neq 0$ or more complex $\varphi(x)$ and $\psi(x)$, can be investigated but the general systems of equations tend to be complicated, and in this paper we want to focus on the exponentiation algorithms. To ensure that $\varphi(x) = \varphi_2 x^2$ and $\ell(x) = \ell_1 x + \ell_0$ depends on x, we will assume that $\varphi_2 \neq 0$ and $\ell_1 \neq 0$.

A. Both Semi- and Fully-Interleaved Ladders

Equation (2) (resp. Equation (5)) $f(x, \ell(x)) = \varphi(x)$ for the semi- (resp. fully-) interleaved cases is equivalent to:

$$\begin{cases} f_{20} + f_{11}\ell_1 + f_{02}\ell_1^2 = \varphi_2 \\ f_{11}\ell_0 + 2f_{02}\ell_1\ell_0 + f_{10} + f_{01}\ell_1 = 0 \\ f_{02}\ell_0^2 + f_{01}\ell_0 + f_{00} = 0 \end{cases}$$

Equation (3) (resp. Equation (6)) $f(\ell(x), x) = \ell(\psi(x))$ for the semi- (resp. fully-) interleaved cases is equivalent to:

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

B. Semi-Interleaved Ladders

The remaining Equation (1) $\psi(\ell(x)) = \ell(\varphi(x))$ for the semi-interleaved cases is equivalent to:

$$\begin{cases} \ell_1^2 = \ell_1\varphi_2 \\ 2\ell_1\ell_0 = 0 \\ \ell_0^2 = \ell_0 \end{cases}$$

So, because $\ell_1 \neq 0$, we have $\ell_1 = \varphi_2$ and $\ell_0 = 0$. Therefore Equation (2) is equivalent to:

$$\begin{cases} f_{20} + f_{11}\varphi_2 + f_{02}\varphi_2^2 = \varphi_2 \\ f_{10} + f_{01}\varphi_2 = 0 \\ f_{00} = 0 \end{cases}$$

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

TABLE X
SEMI-INTERLEAVED LADDER FOR THE EXPONENTIATION

and Equation (3) is equivalent to:

$$\begin{cases} f_{20}\varphi_2^2 + f_{11}\varphi_2 + f_{02} = \varphi_2 \\ f_{10}\varphi_2 + f_{01} = 0 \\ f_{00} = 0 \end{cases}$$

From $f_{10} + f_{01}\varphi_2 = f_{10}\varphi_2 + f_{01}$ we deduce $f_{10}(\varphi_2 - 1) = f_{01}(\varphi_2 - 1)$, which is always true without constraint on φ_2 if $f_{10} = f_{01}$. Moreover, from $f_{10} + f_{01}\varphi_2 = 0$ we obtain $f_{10} = -f_{01}\varphi_2$, thus from $f_{10}\varphi_2 + f_{01} = 0$ we obtain $f_{01}(\varphi_2^2 - 1) = 0$, which is always true without constraint on φ_2 if $f_{01} = 0$. Therefore we assume $f_{10} = 0 = f_{01}$.

From $f_{20}+f_{11}\varphi_2+f_{02}\varphi_2^2 = f_{20}\varphi_2^2+f_{11}\varphi_2+f_{02}$ we deduce $f_{20}(\varphi_2^2 - 1) = f_{02}(\varphi_2^2 - 1)$, which is always true without constraint on φ_2 if $f_{20} = f_{02}$. The remaining constraint is $f_{20}(\varphi_2^2 + 1) + f_{11}\varphi_2 = \varphi_2$, which is equivalent to $f_{20} = \frac{\varphi_2}{\varphi_2^2+1}(1 - f_{11})$. By assuming as in RSA that the coefficients are integers there exists m such that $1 - f_{11} = m(\varphi_2^2 + 1)$. Thus, we have $f_{20} = m\varphi_2$ and $f_{11} = 1 - m(\varphi_2^2 + 1)$.

Therefore, from $\varphi(x) = \varphi_2 x^2$ and $\psi(x) = x^2$ the equations require (for integer coefficients):

$$f(x, y) = m\varphi_2(x^2 + y^2) + (1 - m(\varphi_2^2 + 1))xy$$
$$\ell(x) = \varphi_2 x$$

Note that if $m = 0$ then the solution in Table X is the common Montgomery ladder. But m can be chosen randomly for every iteration, providing a mask for the intermediate values and thus reducing the opportunity of information leakage.

C. Fully-Interleaved Ladders

Equation (4) $g(\varphi(x), \ell(x)) = \ell(\varphi(x))$ for the fully-interleaved cases is equivalent to:

$$\begin{cases} g_{20}\varphi_2^2 = 0 \\ g_{11}\ell_1\varphi_2 = 0 \\ g_{11}\ell_0\varphi_2 + g_{02}\ell_1^2 + g_{10}\varphi_2 = \ell_1\varphi_2 \\ 2g_{02}\ell_1\ell_0 + g_{01}\ell_1 = 0 \\ g_{02}\ell_0^2 + g_{01}\ell_0 + g_{00} = \ell_0 \end{cases}$$

So, because $\varphi_2 \neq 0$ and $\ell_1 \neq 0$, we have $g_{20} = 0$ from the first equation, $g_{11} = 0$ from the second, and the remaining constraints are:

$$\begin{cases} g_{02}\ell_1^2 + g_{10}\varphi_2 = \ell_1\varphi_2 \\ 2g_{02}\ell_1\ell_0 + g_{01}\ell_1 = 0 \\ g_{02}\ell_0^2 + g_{01}\ell_0 + g_{00} = \ell_0 \end{cases}$$

According to Equation (6) $f(\ell(x), x) = \ell(\psi(x)) = \ell_1 x^2 + \ell_0$, so by using $g_{20} = g_{11} = 0$ we have:

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

Thus, Equation (7) $g(f(\ell(x), x), x) = \psi(x)$ 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 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 (4) are:

$$\begin{cases} g_{02}\ell_1^2 + g_{10}\varphi_2 = \ell_1\varphi_2 \\ g_{02}\ell_1\ell_0 = 0 \\ g_{02}\ell_0^2 + g_{00} = \ell_0 \end{cases}$$

Because $g_{02} \neq 0$ and $\ell_1 \neq 0$, the second equation implies that $\ell_0 = 0$. Thus, the remaining constraints from Equation (4) and Equation (7) are:

$$\begin{cases} g_{02}\ell_1^2 + g_{10}\varphi_2 = \ell_1\varphi_2 \\ g_{02} + g_{10}\ell_1 = 1 \\ g_{20} = g_{11} = g_{01} = g_{00} = 0 \end{cases}$$

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

$$g_{10}(\varphi_2 - \ell_1^3) = \ell_1(\varphi_2 - \ell_1)$$

Because $g_{10} \neq 0$ and $\ell_1 \neq 0$, note that $\varphi_2 = \ell_1^3 \Leftrightarrow \varphi_2 = \ell_1$, in which case $\ell_1^3 = \ell_1$, thus $\varphi_2 = \ell_1 = \pm 1$. To remove the constraint on φ_2, we assume that $\varphi_2 \neq \ell_1^3$, and thus g_{10} and g_{02} depends only on φ_2 and ℓ_1:

$$g_{10} = \ell_1 \frac{\varphi_2 - \ell_1}{\varphi_2 - \ell_1^3}$$
$$g_{02} = 1 - g_{10}\ell_1 = \varphi_2 \frac{1 - \ell_1^2}{\varphi_2 - \ell_1^3}$$
$$g(x, y) = g_{02}y^2 + g_{10}x$$
$$= \frac{1}{\varphi_2 - \ell_1^3}\left(\varphi_2(1 - \ell_1^2)y^2 + \ell_1(\varphi_2 - \ell_1)x\right)$$

Note that because $\ell_1 \neq 0$ and $g_{10} \neq 0$ we have to assume that, as opposed to the semi-interleaved cases, $\ell_1 \neq \varphi_2$. Moreover, because $\ell_0 = 0$, the constraints from the common Equations (5) and (6) can be simplified:

$$\begin{cases} f_{20} + f_{11}\ell_1 + f_{02}\ell_1^2 = \varphi_2 & (L_1) \\ f_{20}\ell_1^2 + f_{11}\ell_1 + f_{02} = \ell_1 & (L_2) \\ f_{10} + f_{01}\ell_1 = 0 & (L_3) \\ f_{10}\ell_1 + f_{01} = 0 & (L_4) \\ f_{00} = 0 \end{cases}$$

In order to have a non-trivial ladder $\ell(x) = \ell_1 x$, we will 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 $(f_{02}-f_{20})(\ell_1^2-1)=\varphi_2-\ell_1$, so $f_{02}=f_{20}+\frac{\varphi_2-\ell_1}{\ell_1^2-1}$. Therefore, remain the following constraints:

$$\begin{cases} f_{20}(\ell_1^2+1)+f_{11}\ell_1=\varphi_2-\ell_1^2\dfrac{\varphi_2-\ell_1}{\ell_1^2-1} \\ \qquad\qquad\qquad\qquad =\ell_1-\dfrac{\varphi_2-\ell_1}{\ell_1^2-1} \\ \qquad\qquad\qquad f_{02}=f_{20}+\dfrac{\varphi_2-\ell_1}{\ell_1^2-1} \\ \quad f_{10}=f_{01}=f_{00}=0 \end{cases}$$

The first line being obtained from (L_1) and the second from (L_2). Note that $\varphi_2-\ell_1^2\frac{\varphi_2-\ell_1}{\ell_1^2-1}=\ell_1-\frac{\varphi_2-\ell_1}{\ell_1^2-1}$ is satisfied and thus is not a constraint. Moreover, because $\ell_1\neq 0$, by using the second line, f_{11} can be written as:

$$\begin{aligned} f_{11}&=\frac{1}{\ell_1}\left(\ell_1-\frac{\varphi_2-\ell_1}{\ell_1^2-1}-f_{20}(\ell_1^2+1)\right)\\ &=\frac{1}{\ell_1}\left(\frac{\ell_1^3-\varphi_2}{\ell_1^2-1}-f_{20}(\ell_1^2+1)\right)\end{aligned}$$

Thus, we have:

$$\begin{aligned} f(x,y)&=f_{20}x^2+f_{11}xy+f_{02}y^2\\ &=f_{20}x^2+\frac{1}{\ell_1}\left(\frac{\ell_1^3-\varphi_2}{\ell_1^2-1}-f_{20}(\ell_1^2+1)\right)xy\\ &\quad+\left(f_{20}+\frac{\varphi_2-\ell_1}{\ell_1^2-1}\right)y^2\\ &=f_{20}\left(x^2-\frac{\ell_1^2+1}{\ell_1}xy+y^2\right)\\ &\quad+\frac{1}{\ell_1^2-1}\left(\frac{\ell_1^3-\varphi_2}{\ell_1}xy+(\varphi_2-\ell_1)y^2\right)\end{aligned}$$

Finally, f_{20} has no constraint so, for sake of simplicity, we assume $f_{20}=0$ and obtain the following functions:

$$\begin{aligned} f(x,y)&=\frac{1}{\ell_1^2-1}\left(\frac{\ell_1^3-\varphi_2}{\ell_1}xy-(\ell_1-\varphi_2)y^2\right)\\ g(x,y)&=\frac{1}{\ell_1^3-\varphi_2}\left(\varphi_2(\ell_1^2-1)y^2+\ell_1(\ell_1-\varphi_2)x\right)\\ \ell(x)&=\ell_1 x\end{aligned}$$

where $\ell_1\neq\varphi_2$ and where ℓ_1, ℓ_1^2-1 and $\ell_1^3-\varphi_2$ should be invertible. The former can be satisfied by checking whether $(\ell_1-\varphi_2)\bmod n=0$. The latter can be satisfied by using the Extended Euclidean Algorithm $(d,u)\leftarrow\text{EEA}(v,n)$, where $uv=d\bmod n$, such that if $d=1$ then v is invertible modulo n and $v^{-1}=u\bmod n$. Note that when ℓ_1 is chosen, these coefficients are constant during the iterative conditional branching, thus they can be pre-computed at the beginning of the algorithm in Table XI. These computations cause an overhead that does not depend on the secret, thus trading execution time and energy consumption for more security. If no such ℓ_1 exists then the algorithm returns an error.

We provide in Table XII 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 and A for addition/subtraction, all modulo n. We did not include the cost of the pre-computations, which is far from negligible for the fully-interleaved ladder in Table XI.

input public a,n; secret k
1: $\ell\leftarrow 1$
2: **do**
3: **if** $\ell=n-1$ **then**
4: **return** error
5: **else**
6: $\ell\leftarrow\ell+1$
7: **end if**
8: $v_0\leftarrow\ell-a\bmod n$
9: $(d_1,u_1)\leftarrow\text{EEA}(\ell,n)$
10: $v_2\leftarrow\ell^2-1\bmod n$
11: $(d_2,u_2)\leftarrow\text{EEA}(v_2,n)$
12: $v_3\leftarrow\ell^3-a\bmod n$
13: $(d_3,u_3)\leftarrow\text{EEA}(v_3,n)$
14: **while** $v_0\bmod n=0\vee d_1\neq 1\vee d_2\neq 1\vee d_3\neq 1$
15: $c_0\leftarrow u_1u_2v_3\bmod n$
16: $c_1\leftarrow -v_0u_2\bmod n$
17: $c_2\leftarrow av_2u_3\bmod n$
18: $c_3\leftarrow\ell v_0u_3\bmod n$
19: $x\leftarrow 1$
20: $y\leftarrow\ell$
21: **for** $i=d$ **to** 0 **do**
22: **if** $k[i]=1$ **then**
23: $z\leftarrow y^2\bmod n$
24: $x\leftarrow c_0xy+c_1z\bmod n$
25: $y\leftarrow c_2z+c_3x\bmod n$
26: **else**
27: $z\leftarrow x^2\bmod n$
28: $y\leftarrow c_0yx+c_1z\bmod n$
29: $x\leftarrow c_2z+c_3y\bmod n$
30: **end if**
31: **end for**
32: **return** x
output $x=a^k\bmod n$ or error

TABLE XI
FULLY-INTERLEAVED LADDER FOR THE EXPONENTIATION

Montgomery	Semi-Interleaved	Fully-Interleaved
$M+S$	$5M+2S+3A$	$5M+S+2A$

TABLE XII
COST OF THE MONTGOMERY, SEMI- AND FULLY-INTERLEAVED LADDERS

While investigating the fully-interleaved cases, we found another solution shown but not detailed in Table XIII as additional material for this paper. It is a modular exponentiation for bases of the form $a=b^2(b^2+b-1)$ where b is invertible, for instance, $b=n-1$. Actually, if b is not invertible then the initialization could be $x\leftarrow b$ and $1\leftarrow 1\bmod n$ instead, in which case the result would be $x=b^{d+1}(b^2(b^2+b-1))^k\bmod n$ where b^{d+1} is public. Note also that this time the condition is $k[i]=0$.

VI. CONCLUSION AND FUTURE WORK

In this paper, we abstract away the algorithmic strength of the Montgomery ladder against side-channel and fault-injection attacks, by defining semi- and fully-ladderizable programs. We designed also fault-injection attacks able to obtain some/all bits of the secret key from the semi-interleaved ladders, like the Montgomery ladder, but none from the fully-interleaved ladders. As examples, we also provided for the modular exponentiation a better semi-interleaved ladder using a random mask updated at every iteration, a fully-interleaved ladder depending on an appropriate ladder constant, and a simpler fully-interleaved ladder for some bases.

The algorithm in Table XI is a general fully-interleaved ladder for the exponentiation, and depends on the existence of a ladder constant ℓ. Its existence is a mathematical problem

input public b, n; secret k
$x \leftarrow 1$
$y \leftarrow b^{-1} \bmod n$
for $i = d$ **to** 0 **do**
  if $k[i] = 0$ **then**
    $x \leftarrow b^2(b+1)y^2 - b^2xy \bmod n$
    $y \leftarrow (b+1)x - b(b^2+b-1)y^2 \bmod n$
  else
    $y \leftarrow b^2(b+1)x^2 - b^2yx \bmod n$
    $x \leftarrow (b+1)y - b(b^2+b-1)x^2 \bmod n$
  end if
end for
return x
output $x = (b^2(b^2+b-1))^k \bmod n$

TABLE XIII
SIMPLER FULLY-INTERLEAVED LADDER FOR SOME BASES

that should be investigated to ensure that the proposed solution is relevant. Moreover, obtaining a formula for ℓ depending only on a and n would reduce the overhead of the algorithm. Because in RSA we have $n = pq$ with p, q primes, verifying whether a solution exists or not might be easier, but a formula for ℓ should not depend on p, q because they are secret.

We have 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, and thus the manipulation of equations in Section V could be handled by using matrix operations.

Finally, the double-and-add and the secure bit permutation algorithms mentioned in Subsection III-A could be investigated as well, providing more examples to demonstrate the generality of the method described in this paper.

ACKNOWLEDGEMENTS

The authors want to thank the reviewers for their relevant and very useful comments.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[2] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Advances in Cryptology — CRYPTO '96*, N. Koblitz, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 104–113.

[3] C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil, "Square Always Exponentiation," in *Progress in Cryptology – INDOCRYPT 2011*, D. J. Bernstein and S. Chatterjee, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 40–57.

[4] J. Ha, Y. Choi, D. Choi, and H. Lee, "Power Analysis Attacks on the Right-to-Left Square-Always Exponentiation Algorithm," *J. Internet Serv. Inf. Secur.*, vol. 4, pp. 38–51, 2014.

[5] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2004, pp. 16–29.

[6] J.-S. Coron, "Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems," in *Cryptographic Hardware and Embedded Systems*, Ç. K. Koç and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 292–302.

[7] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Power Analysis Attacks of Modular Exponentiation in Smartcards," in *Cryptographic Hardware and Embedded Systems*, Ç. K. Koç and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 144–157.

[8] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[9] Y. Sung-Ming, S. Kim, S. Lim, and S. Moon, "A Countermeasure against One Physical Cryptanalysis May Benefit Another Attack," in *Information Security and Cryptology — ICISC 2001*, K. Kim, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 414–427.

[10] H. Ziade, R. Ayoubi, and R. Velazco, "A survey on fault injection techniques," *International Arab Journal of Information Technology*, vol. Vol. 1, No. 2, July, pp. 171–186, 2004. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00105562

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

[12] Y. Marquer, "Algorithmic Completeness of Imperative Programming Languages," *Fundamenta Informaticae*, vol. 168, no. 1, pp. 51–77, July 2019.

[13] A. Boscher, R. Naciri, and E. Prouff, "CRT RSA Algorithm Protected Against Fault Attacks," in *Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems*, D. Sauveron, K. Markantonakis, A. Bilas, and J.-J. Quisquater, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 229–243.

[14] M. Joye, "Highly Regular Right-to-Left Algorithms for Scalar Multiplication," in *Cryptographic Hardware and Embedded Systems - CHES 2007*, P. Paillier and I. Verbauwhede, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 135–147.

[15] ——, "Highly Regular m-Ary Powering Ladders," in *Selected Areas in Cryptography*, M. J. Jacobson, V. Rijmen, and R. Safavi-Naini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 350–363.

[16] C. D. Walter, "The Montgomery and Joye Powering Ladders are Dual," *IACR ePrint Archive*, vol. 1081, pp. 1–6, 2017. [Online]. Available: https://eprint.iacr.org/2017/1081.pdf

[17] M. Joye and S.-M. Yen, "The Montgomery Powering Ladder," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, B. S. Kaliski, ç. K. Koç, and C. Paar, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 291–302.

[18] Á. Kiss, J. Krämer, P. Rauzy, and J.-P. Seifert, "Algorithmic Countermeasures Against Fault Attacks and Power Analysis for RSA-CRT," in *Constructive Side-Channel Analysis and Secure Design*, F.-X. Standaert and E. Oswald, Eds. Cham: Springer International Publishing, 2016, pp. 111–129.

[19] C. Giraud, "An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis," *IEEE Transactions on Computers*, vol. 55, no. 9, pp. 1116–1120, Sep. 2006.

[20] V. S. Miller, "Use of Elliptic Curves in Cryptography," in *Advances in Cryptology — CRYPTO '85 Proceedings*, H. C. Williams, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 417–426.

[21] N. Koblitz, "Elliptic Curve Cryptosystems," *Math. Comp*, vol. 48, pp. 243–264, 01 1987.

[22] F. Strenzke, E. Tews, H. G. Molter, R. Overbeck, and A. Shoufan, "Side Channels in the McEliece PKC," in *Post-Quantum Cryptography*, J. Buchmann and J. Ding, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 216–229.

[23] 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)*, April 2016, pp. 132–137.

[24] H. Kim, T. H. Kim, J. C. Yoon, and S. Hong, "Practical Second-Order Correlation Power Analysis on the Message Blinding Method and Its Novel Countermeasure for RSA," *ETRI Journal*, vol. 32, no. 1, pp. 102–111, 2010. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.4218/etrij.10.0109.0249

[25] N. Hanley, H. Kim, and M. Tunstall, "Exploiting Collisions in Addition Chain-Based Exponentiation Algorithms Using a Single Trace," in *Topics in Cryptology — CT-RSA 2015*, K. Nyberg, Ed. Cham: Springer International Publishing, 2015, pp. 431–448.

[26] D.-P. Le, C. H. Tan, and M. Tunstall, "Randomizing the Montgomery Powering Ladder," in *Information Security Theory and Practice*, R. N. Akram and S. Jajodia, Eds. Cham: Springer International Publishing, 2015, pp. 169–184.

[27] J. Arlat, "Validation de la sûreté de fonctionnement par injection de fautes : méthode, mise en oeuvre, application," Ph.D. dissertation, Institut national polytechnique (Toulouse, France), 1990.