Randomizing the Montgomery Powering Ladder

Duc-Phong Le[1], Chik How Tan[1], and Michael Tunstall[2]

[1] Temasek Laboratories, National University of Singapore,
5A Engineering Drive 1, #09-02, Singapore 117411
{tslld,tsltch}@nus.edu.sg
[2] Rambus Cryptography Research Division,
425 Market Street, 11th Floor, San Francisco,
CA 94105, United States
michael.tunstall@cryptography.com

Abstract. In this paper, we present novel randomized techniques to enhance Montgomery powering ladder. The proposed techniques increase the resistance against side-channel attacks and especially recently published correlation collision attacks in the horizontal setting. The first of these operates by randomly changing state such that the difference between registers varies, unpredictably, between two states. The second algorithm takes a random walk, albeit tightly bounded, along the possible addition chains required to compute an exponentiation. We also generalize the Montgomery powering ladder and present randomized (both left-to-right and right-to-left) m-ary exponentiation algorithms.

Keywords: Montgomery Powering Ladder, Side-Channel Analysis, Countermeasures.

1 Introduction

Side-channel analysis is one of the most serious threats to the security of a given implementation of a cryptographic algorithm. In the traditional model, a given cryptographic algorithm is typically proven secure against various attacks under assumptions regarding the computational complexity of an attack. However, in a more practical scenario, Kocher noted that the time required to compute a cryptographic algorithm could reveal information on the keys used [1]. This was then extended to analyze differences in the power consumption of a microprocessor [2] and the variations in the surrounding electromagnetic field [3, 4]. The simplest such attack is based on the inspection of an acquired power consumption (resp. electromagnetic emanation) trace to derive information. This is referred to as Simple Power Analysis (SPA) (resp. Simple ElectroMagnetic Analysis (SEMA)). The exploitation of statistical differences in the instantaneous power consumption proposed by Kocher et al. [2] is termed Differential Power Analysis (DPA) (resp. Differential ElectroMagnetic Analysis (DEMA)), and alternatives have been proposed using, for example, a model and Pearson's correlation coefficient [5] or mutual information [6].

In choosing an exponentiation algorithm for a secure implementation, one needs to consider the possible attacks that could be applied. One can often discount attacks where input values need to be known, such as the doubling attack [7], template attacks [8] or DPA [2]. Such attacks can typically be prevented by blinding input values [9] or by using a suitable padding scheme. That is, these attacks are not typically prevented by choosing a specific exponentiation algorithm. However, one also needs to consider attacks based on inspecting a limited number of traces, such as SPA [2], or power attacks in the horizontal setting [10, 11]. The later was first introduced by Walter [10]. His attack, the so-called Big Mac attack, applied to m-ary exponentiation, although only simulated attacks are described. Clavier et al. [12] then exploited the collision correlation between selected points from two subtraces to derive information. Recent work by Bauer et al. [13] has also detailed how one could apply such collision attacks to implementations of scalar multiplications over elliptic

curves. Witteman et al. [14] demonstrated that this attack works on an ASIC implementation. Kim et al. [15] also determined how one could apply such an attack to the Montgomery ladder [16, 17]. The attack model was extended by Hanley et al. [18] to include an attacker that computes the correlation between carefully chosen points in a trace to detect where the output of one operation is used as the input to another operation.

In this paper, we present randomized variants of the Montgomery powering ladder that are resistant to SPA and power collision correlation attacks in the horizontal setting. The first algorithm is based on an amalgamation of two simple variants of the Montgomery powering ladder, where the content of the registers becomes *unpredicable*. The second algorithm is based on blinding addition chains, i.e. it takes tightly bounded random walks to compute an exponentiation. We also generalize the Montgomery powering ladder and present randomized m-ary exponentiation algorithms. This algorithm is not entirely resistant, but presents a significant increase in attack complexity compared to a naïve implementation of the Montgomery powering ladder, and therefore an adversary requires a significantly lower error rate to succeed. Furthermore, this variant will prevent lattice-based attacks [19] when used, for example, in implementations of ECDSA, as an adversary cannot derive information on individual bits.

The organization of the paper is as follows. Section 2 presents a strong attack model such that we can present a pessimistic security analysis of our algorithms. We also recall the Montgomery powering ladder and define the properties of variants of this algorithm in this section. In Section 3, we first describe a few simple variants, and then show how these can be combined to produce a secure algorithm. We introduce a randomized Montgomery powering ladder, that randomly changes state such that the difference between registers varies, unpredictably, between two states. In Section 4, we show how the Montgomery powering ladder can be modified to conduct a random walk through the addition chains that can be used to compute an exponentiation. Section 5 generalize the Montgomery powering ladder and present randomized m-ary exponentiation algorithms. We conclude in Section 6.

2 Preliminaries

2.1 Attack Model

In this paper we shall predominantly be considering an adversary that is able to take power consumption traces (or something equivalent) while a microprocessor is computing a group exponentiation algorithm. The adversary is then able to make deductions on what the microprocessor is computing. We shall consider three different attacks that require a limited number of traces when discussing the effects of our modifications to the Montgomery powering ladder.

1. The first attack is Simple Power Analysis (SPA) where one observes differences in the power consumption caused by different operations taking place. This was first demonstrated by Kocher *et al.* [2] who showed that one could, given a naïve implementation, observe the difference between operations during the computation of an exponentiation in $(\mathbb{Z}/N\,\mathbb{Z})^*$. Regular algorithms [9, 17, 20] can be used to thwart SPA attacks.
2. Another attack is the use of Pearson's correlation coefficient to detect collisions in variables to deduce key values [11]. For example, during the computation of an exponentiation in $(\mathbb{Z}/N\,\mathbb{Z})^*$ using Coron's *square-and-multiply-always* exponentiation algorithm, one could seek to determine the locations of multiplications with the same input. That is, operations that both take the same input should show a significant cross-correlation.
3. An extension to the collision correlation attack given above is where an adversary is able to detect collisions between the output of one operation and the input of another operation. This provides an attack

where a complete defense is not possible. However, limiting the information available to an adversary can make the attack impractical since it has been shown that the error rate for this attack is significantly higher than a straightforward comparison of operations [18].

We do not consider attacks that require chosen inputs, such as the doubling attack [7] or statistical differences in the power consumption over time [2, 5]. This is because these attacks are typically prevented by padding or blinding the input by using a redundant representation where the details depend on the group being used [9, 21]. For example, in many instances the input will also be padded such that an attacker cannot control, or predict, the input to a group exponentiation algorithm.

Typically, a blinded exponent is used, which is equivalent to the actual exponent, meaning that each trace must be attacked independently. The discussion of the security of the proposed algorithms will be largely informal, except where we wish to make specific claims about the amount of information available to an adversary.

2.2 The Montgomery Powering Ladder

The Montgomery powering ladder was originally proposed as a means of speeding up scalar multiplication over elliptic curves [16], and was later shown to be applicable to all multiplicatively written abelian groups [17]. The Montgomery powering ladder is an instance of an addition chain-based exponentiation algorithm that is often cited as an alternative to the binary exponentiation algorithm for resource constrained device such as smart cards. We recall the definition of an addition chain and define some notation.

Definition 1. *An addition chain of length ℓ in group $\mathbb{G}$ for a group element $x \in \mathbb{G}$ is a sequence of group elements $(a_0, a_1, a_2, \ldots, a_\ell)$ such that $a_0 = 1_\mathbb{G}$, $a_1 = x$ and $a_k = a_i \cdot a_j$, $0 \leq i \leq j < k \leq \ell$ and $\cdot$ is the group operation. The values of i and j are chosen such that a_k is a chosen power of a_1.*

For ease of expression we will denote the n-th power of x as x^n. Similarly, much of the discussion will implicitly assume that the computation is taking place in $\mathbb{Z}$ without loss of generality.

We recall the description of the Montgomery powering ladder given by Joye and Yen [17]. We consider the problem of computing $y = x^k$ in $\mathbb{G}$ for inputs x and k. Let $\sum_{i=0}^{n-1} k_i\, 2^i$ be the binary expansion of κ with bit length n. For ease of expression we shall also denote this as $(k_{n-1}, \ldots, k_0)_2$. The Montgomery powering ladder relies on the following observation. Defining $L_j = \sum_{i=j}^{n-1} k_i\, 2^{i-j}$ and $H_j = L_j + 1$, we have

$$L_j = 2\, L_{j+1} + k_j = L_{j+1} + H_{j+1} + k_j - 1 = 2\, H_{j+1} + k_j - 2 \tag{1}$$

and so we obtain

$$(L_j, H_j) = \begin{cases} (2\, L_{j+1},\, L_{j+1} + H_{j+1}) & \text{if } k_j = 0\,, \\ (L_{j+1} + H_{j+1},\, 2\, H_{j+1}) & \text{if } k_j = 1\,. \end{cases} \tag{2}$$

If we consider one register containing x^{L_j} and another containing x^{H_j} then (2) implies that

$$(x^{L_j}, x^{H_j}) = \begin{cases} \left(\left(x^{L_{j+1}}\right)^2,\, x^{L_{j+1}} \cdot x^{H_{j+1}} \right) & \text{if } k_j = 0\,, \\ \left(x^{L_{j+1}} \cdot x^{H_{j+1}},\, \left(x^{H_{j+1}}\right)^2 \right) & \text{if } k_j = 1\,. \end{cases}$$

Given that $L_0 = \kappa$ one can build an exponentiation algorithm that requires two group operations per bit of the exponent. Joye and Yen give several different versions of such an algorithm [17]. Algorithm 1 describes the most resistant to side-channel analysis version in their paper (as noted by Kim et al. [15] who describe implementations of cross correlation attacks on other versions).

Algorithm 1: Montgomery Powering Ladder

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Output: x^{κ}

1 $R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow x$;
2 **for** $i = n-1$ **down to** 0 **do**
3 $\quad R_{\neg k_i} \leftarrow R_{k_i} \cdot R_{\neg k_i}$;
4 $\quad R_{k_i} \leftarrow (R_{k_i})^2$;
5 **end**
6 **return** R_0

The Montgomery powering ladder, as described in Algorithm 1, has several properties that make it useful when defining a side-channel resistant implementation of an exponentiation. However, the Montgomery Powering ladder also has been shown to be vulnerable to recent collision correlation attacks in the horizontal setting [12, 18]. The attack in [18] functions by detecting where the outputs of lines 3 and 4 are used in the next iteration of the main loop. In the reminder of this paper we propose alternative versions of the Montgomery powering ladder to enhance its security against horizontal collision correlation attacks.

Definition 2. *We define a variant of the Montgomery powering ladder as an exponentiation algorithm that has the following properties.*

1. *The algorithm uses two registers in the main loop containing group elements, both of which are updated in each iteration.*
2. *Each iteration of the main loop treats one bit of the exponent and contains no more than two group operations.*
3. *The operands in the first group operation will only involve one or both of the registers used in the main loop.*
4. *The operands in the second group operation can involve one or both of the registers used in the main loop and/or some precomputed value.*

The 3rd and 4th properties allow for variants of the Montgomery ladder to be defined. That is, the Montgomery ladder is in the set of possible algorithms that satisfy these criteria.

For brevity in defining algorithms, we shall concentrate on the main loop of the algorithm. The computation before and after the main loop may contain **if**-statements. We shall not give fully secure versions where solutions are widely known, e.g. dummy operations or redundant representations [9, 21, 22].

3 Randomizing the Montgomery Powering Ladder

We note that when computing x^n using the Montgomery powering ladder, as defined in Algorithm 1, then at the end of each iteration we will have the condition where $R_1/R_0 = x$, or equivalently $R_0 \cdot x = R_1$. Thus, using the notation given above, and we allow some precomputed values to be used in the algorithm, then (2) can be rewritten as

$$(L_j, H_j) = \begin{cases} (2\,L_{j+1}, L_j + 1) & \text{if } k_j = 0\,, \\ (L_{j+1} + H_{j+1}, L_j + 1) & \text{if } k_j = 1\,. \end{cases} \tag{3}$$

Implying that

$$(x^{L_j}, x^{H_j}) = \begin{cases} \left(\left(x^{L_{j+1}}\right)^2, x^{L_j} \cdot x\right) & \text{if } k_j = 0\,, \\ \left(x^{L_{j+1}} \cdot x^{H_{j+1}}, x^{L_j} \cdot x\right) & \text{if } k_j = 1\,. \end{cases}$$

From which we can define Algorithm 2.

Algorithm 2: A Straightforward Variant I

Input: $x \in \mathbb{G}$, an n-bit integer
$\kappa = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Output: x^κ

1 $R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow x$;
2 **for** $i = n-1$ **down to** 0 **do**
3 $\quad R_0 \leftarrow R_0 \cdot R_{k_i}$;
4 $\quad R_1 \leftarrow R_0 \cdot x$;
5 **end**
6 **return** R_0

Algorithm 3: A Straightforward Variant II

Input: $x \in \mathbb{G}$, an n-bit integer
$\kappa = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$, and $k_{n-1} = 1$
Output: x^κ

1 $R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow 1_{\mathbb{G}}$;
2 **for** $i = n-1$ **down to** 0 **do**
3 $\quad R_1 \leftarrow R_0 \cdot R_{\neg k_i}$;
4 $\quad R_0 \leftarrow R_1 \cdot x$;
5 **end**
6 **return** R_0

Following the reasoning used to define (1), we can instead define $L_j = \sum_{i=j}^{n-1} k_i \, 2^{i-j}$ and $H_j = L_j - 1$, giving

$$L_j = 2\,L_{j+1} + k_j = L_{j+1} + H_{j+1} + k_j + 1 = 2\,H_{j+1} + k_j + 2 \tag{4}$$

and so, as with (3), we obtain

$$(L_j, H_j) = \begin{cases} (H_j + 1, L_{j+1} + H_{j+1}) & \text{if } k_j = 0\,, \\ (H_j + 1, 2\,L_{j+1}) & \text{if } k_j = 1\,. \end{cases} \tag{5}$$

If we consider one register containing x^{L_j} and another containing x^{H_j} then (5) implies that

$$(x^{L_j}, x^{H_j}) = \begin{cases} \left(x^{H_j} \cdot x, x^{L_{j+1}} \cdot x^{H_{j+1}}\right) & \text{if } k_j = 0\,, \\ \left(x^{H_j} \cdot x, \left(x^{L_{j+1}}\right)^2\right) & \text{if } k_j = 1\,. \end{cases}$$

From which we can define Algorithm 3. We note that at the end of each iteration we will have the condition where $R_0/R_1 = x$, or equivalently $R_1 \cdot x = R_0$.

Suppose that an adversary can distinguish multiplications from squaring operations, then the two above variants of Montgomery powering ladder are not immune to SPA. In line 3 of both algorithms, a squaring operation will occur for certain bit values. That is, a squaring operation will be computed if the bit value $k_i = 0$ in Algorithm 2 and in Algorithm 3 a squaring operation occurs when $k_i = 1$. The following randomized algorithm will deal with this problem.

We observe that Algorithms 2–3 can be blended together. That is, in loop ℓ, for some $\ell \in \{1, \ldots, n-1\}$, of Algorithm 2 if $k_\ell = 1$ one could compute $R_1 \leftarrow (R_0)^2$ followed by $R_0 \leftarrow R_1 \cdot x$. Before this step $R_1/R_0 = x$, and afterwards one bit of the exponent is treated and $R_0/R_1 = x$. Hence, one could continue to compute an exponentiation using Algorithm 3.

Likewise, in loop ℓ, for some $\ell \in \{1, \ldots, n-1\}$, of Algorithm 3, if $k_\ell = 0$, one could compute $R_0 \leftarrow (R_0)^2$ followed by $R_1 \leftarrow R_0 \cdot x$. Afterwards one bit of the exponent is treated and $R_1/R_0 = x$. Hence, one could continue to compute an exponentiation using Algorithm 2. We define Algorithm 4 as an example of how Algorithms 2 and 3 could be randomly blended together. We use a random generator producing a random bit b to determine when to change from one algorithm to the other, and to determine which algorithm is used to start the exponentiation algorithm. If $b = 0$, the exponentiation is computed by using Algorithm 2, that is $R_1/R_0 = x$. Otherwise, the exponentiation is computed by using Algorithm 3, that is, $R_0/R_1 = x$.

Algorithm 4: Randomized Montgomery Powering Ladder

Input: $x \in \mathbb{G}$, an n-bit integer
$\kappa = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$, and $k_{n-1} = 1$
Output: x^κ

$b \xleftarrow{R} \{0,1\}$; $R_0 \leftarrow 1_\mathbb{G}$;
if $b = 0$ **then**
  $R_1 \leftarrow x$;
else
  $R_1 \leftarrow 1_\mathbb{G}$;
end
for $i = n-1$ **down to** 0 **do**
  if $b \oplus k_i = 1$ **then**
    $b \xleftarrow{R} \{0,1\}$;
  end
  $R_b \leftarrow R_0 \cdot R_{b \oplus k_i}$;
  $R_{\neg b} \leftarrow R_b \cdot x$;
end
return R_0

Algorithm 5: A Straightforward Variant III

Input: $x \in \mathbb{G}$, an n-bit integer
$\kappa = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Output: x^κ

$R_0 \leftarrow 1_\mathbb{G}$; $R_1 \leftarrow 1_\mathbb{G}$;
for $i = n-1$ **down to** 0 **do**
  $R_{k_i} \leftarrow (R_0)^2$;
  $R_{\neg k_i} \leftarrow R_{k_i} \cdot x$;
end
return R_0

When $b \oplus k_i = 1$ (*i.e.*, $b \neq k_i$) in Algorithm 4, there will be two multiplications involved the bit k_i if one continues to compute an exponentiation using the current algorithm (*i.e.*, Algorithm 2 or Algorithm 3). Algorithm 4 will generate a random bit b to decide whether two Algorithms 2–3 are blended together or not. In the case of blending, two multiplications will be replaced by one squaring, and then one multiplication.

Suppose that an adversary is able to distinguish a multiplication from a squaring operation. Then, she would be able to determine individual bits of the exponent if she could determine if $R_0/R_1 = x$ or $R_1/R_0 = x$. However, the following lemma shows that this information is not available.

Lemma 1. *An adversary analyzing an instance of Algorithm 4 is able to reduce the hypotheses for the exponent from κ to $\kappa^{\frac{11}{12}}$ by distinguishing a multiplication from a squaring operation.*

Proof. If an adversary observes a squaring operation followed by a multiplication in the loop using index i, then the adversary knows one of the following operations has occurred:

1. Where $R_1/R_0 = x$ and $k_i = 0$.
2. Where $R_0/R_1 = x$ and $k_i = 1$.
3. Where $R_1/R_0 = x$ changes to $R_0/R_1 = x$ and $k_i = 1$.
4. Where $R_0/R_1 = x$ changes to $R_1/R_0 = x$ and $k_i = 0$.

At an arbitrary point, each of these occur with probability $\frac{1}{4}$. Likewise, if an adversary observes two multiplications in the loop using index i, hence the adversary knows one of the following operations has occurred:

1. Where $R_1/R_0 = x$ and $k_i = 1$.
2. Where $R_0/R_1 = x$ and $k_i = 0$.

Each of these occur with probability $\frac{1}{2}$. Hence, there is no information available to an adversary since for any observed sequence of operations $\Pr(k_i = 0) = \Pr(k_i = 1) = \frac{1}{2}$.

However, if an adversary observes y consecutive pairs of multiplications then the adversary will know that y consecutive bits have the same value. If an attacker observes a pair of multiplications then the distribution of the number of subsequent pairs W of multiplications is geometric. That is, $W \sim$ Geometric$(\frac{3}{4})$,

since the following bit has to be the same and the randomly generated bit has to be a specific value. The, by definition, the expected length of a run of multiplication is $\frac{4}{3}$ operations, where each observation would therefore provide an expected $\frac{4}{3} - 1 = \frac{1}{3}$ of a bit of the exponent. A pair of multiplications will occur with probability $\frac{1}{4}$, giving an expected $n/4$ bits for an n-bit exponent. For an n-bit exponent an expected $\frac{1}{3} \times \frac{n}{4} = \frac{n}{12}$ bits can expected to be derived. Hence, κ hypotheses can be expected to be reduced to $\kappa^{\frac{11}{12}}$ hypotheses. □

As with the Montgomery powering ladder shown in Algorithm 1, an attack using collisions based on the reuse of variables is not possible, but a collision attack based in the use of the output of operations is possible. One can attempt to observe whether the second operand in line 11 of Algorithm 4 is created from line 11 or line 12 in the previous iteration of the exponentiation loop. However, an analysis based on this will return the wrong key hypothesis when the algorithm changes from using (3) to (5).

Lemma 2. *An adversary analyzing an instance of Algorithm 4 using a collision attack an adversary would be able to reduce the hypotheses for the exponent from κ to $\kappa^{\frac{3}{4}}$.*

Proof. If we, arbitrarily, consider the ℓ-th loop of the exponentiation loop, we have $\Pr(b \oplus k_i = 1) = \frac{1}{2}$ and the probability that b changes, and hence the algorithm being used, is also $\frac{1}{2}$. An adversary making a deductions using a collision attack will have to guess the value of b in two consecutive loops of the algorithm. Given a correct guess for b it will remain the same with probability $\frac{3}{4}$. On the assumption that an attack will validate b in the first loop the value of b in the second loop will remain the same with probability $\frac{3}{4}$. If it changes an incorrect result will be given. Hence, the probability that a collision would detect an incorrect key bit is $\frac{1}{4}$, and an adversary would be able to determine a given bit with a probability $\frac{3}{4}$ leading to a reduction in the number of hypotheses from κ to $\kappa^{\frac{3}{4}}$. □

Given that an adversary could potentially obtain bits of an exponent if they analyze Algorithm 4, and are able to distinguish a multiplication from a squaring operation, it would be tempting to ensure that no pairs of multiplications occur. That is, make the algorithm change rather than compute two multiplications resulting in Algorithm 5. This is equivalent to the *square-and-multiply-always* exponentiation algorithm proposed by Coron [9]. This is a widely studied algorithm with numerous analyses in the literature, which will not be repeated here.

4 Random Walk Method

In this section, we generalize the difference between the two registers used in the Montgomery powering ladder to be some arbitrary power of the input. This leads to an algorithm that computes a group exponentiation taking a random, albeit tightly bounded, walk through the possible addition chains.

If one is working in a group where computing the inverses of an element can be readily computed, then other options for a variant of the Montgomery powering ladder are possible. We note that (2) can be rewritten as

$$(L_j, H_j) = \begin{cases} (H_j - 1, L_{j+1} + H_{j+1}) & \text{if } k_j = 0\,, \\ (L_{j+1} + H_{j+1}, L_j + 1) & \text{if } k_j = 1\,. \end{cases} \tag{6}$$

Implying that

$$(x^{L_j}, x^{H_j}) = \begin{cases} \left(x^{H_j} \cdot x^{-1}, x^{L_{j+1}} \cdot x^{H_{j+1}}\right) & \text{if } k_j = 0\,, \\ \left(x^{L_{j+1}} \cdot x^{H_{j+1}}, x^{L_j} \cdot x\right) & \text{if } k_j = 1\,. \end{cases}$$

(6) can be rewritten as follows:

$$(L_j, H_j) = \begin{cases} (L_{j+1} + H_{j+1}, L_j - 1) & \text{if } k_j = 0\,, \\ (L_{j+1} + H_{j+1}, L_j + 1) & \text{if } k_j = 1\,. \end{cases} \tag{7}$$

Implying that

$$(x^{L_j}, x^{H_j}) = \begin{cases} \left(x^{L_{j+1}} \cdot x^{H_{j+1}}, x^{L_j} \cdot x^{-1}\right) & \text{if } k_j = 0\,, \\ \left(x^{L_{j+1}} \cdot x^{H_{j+1}}, x^{L_j} \cdot x\right) & \text{if } k_j = 1\,. \end{cases}$$

From which we can define Algorithm 6. In previous examples of Montgomery powering ladders presented in this paper R_0 has acted as an accumulator and returned the result. In Algorithm 6 the accumulator, i.e. the register containing the correct power of x at the end of each loop, shifts depending on the value of the bit of the exponent being treated. Hence, $R_{\neg k_0}$ is returned at the end of the algorithm.

Algorithm 6: Variant with Inverses

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Output: x^κ

1 $R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow x$;
2 $U_0 \leftarrow x^{-1}$; $U_1 \leftarrow x$;
3 **for** $i = n - 1$ **down to** 0 **do**
4 $\quad R_0 \leftarrow R_0 \cdot R_1$;
5 $\quad R_1 \leftarrow R_0 \cdot U_{k_i}$;
6 **end**
7 **return** $R_{\neg k_0}$

In analyzing an instance of Algorithm 6, an adversary would not be able to determine any information on bits of the exponent based on distinguishing a multiplication from a squaring operation, since no squaring operations take place. However, a collision attack is possible by observing where the multiplication with U_i, for $i \in \{0, 1\}$, collides with the multiplications used to generate these values in line 2. If an adversary is able to determine whether $U_{k_i} = x^{-1}$ or $U_{k_i} = x$ was used, for some $i \in \{0, \ldots, n-1\}$, then individual bits of the exponent can be determined.

We can modify (7) by choosing $L'_{j+1} = L_{j+1} + \alpha$ and $H'_{j+1} = H_{j+1} + \beta$, giving

$$(L'_j, H'_j) = \begin{cases} (L'_{j+1} + H'_{j+1}, L'_j - 1) & \text{if } k_j = 0\,, \\ (L'_{j+1} + H'_{j+1}, L'_j + 1) & \text{if } k_j = 1\,. \end{cases} \tag{8}$$

where $L'_j = L_j + \gamma$, $H'_j = H_j + \mu$, and $\gamma = \alpha + \beta$. If we choose γ as a random element from $\{-h, \ldots h\}$ for some small integer h, then α and β can be chosen such that $\gamma = \alpha + \beta$. If we assume that α is fixed then $\beta = \gamma - \alpha$, i.e., $H'_{j+1} = H_{j+1} + \gamma - \alpha$. Given that H'_{j+1} is computed from L'_{j+1} this can be done by

$$H'_{j+1} = \begin{cases} L_{j+1} + (\gamma - \alpha - 1) = L'_{j+1} + (\gamma - 2\,\alpha - 1) & \text{if } k_j = 0\,, \\ L_{j+1} + (\gamma - \alpha + 1) = L'_{j+1} + (\gamma - 2\,\alpha + 1) & \text{if } k_j = 1\,. \end{cases}$$

This also removes the need to have the accumulating register change as described for Algorithm 6. If we define a value $L'_j = L_j + \gamma_j$ then (8) can be rewritten as $(L_j + \gamma_j, H_j + (\gamma_{j-1} - \gamma_j) =$

$$\begin{cases} \left(L_{j+1} + \gamma_{j+1} + H_{j+1} + (\gamma_j - \gamma_{j+1}), L'_j + (\gamma_{j-1} - 2\,\gamma_j - 1)\right) & \text{if } k_j = 0\,, \\ \left(L_{j+1} + \gamma_{j+1} + H_{j+1} + (\gamma_j - \gamma_{j+1}), L'_j + (\gamma_{j-1} - 2\,\gamma_j + 1)\right) & \text{if } k_j = 1\,. \end{cases} \tag{9}$$

If we consider one register containing $x^{L'_j}$ and another containing $x^{H'_j}$ then (9) implies that

$$(x^{L'_j}, x^{H'_j}) = \left(x^{L'_{j+1}} \cdot x^{H'_{j+1}}, x^{L'_j} \cdot x^{\Delta}\right)$$

where

$$\Delta = \begin{cases} \gamma_{j-1} - 2\,\gamma_j - 1 & \text{if } k_j = 0\,, \\ \gamma_{j-1} - 2\,\gamma_j + 1 & \text{if } k_j = 1\,. \end{cases}$$

Given that $L_0 = \kappa$ one can build an exponentiation algorithm as shown in Algorithm 7 where we set $\gamma_0 = 0$ to produce the correct result. Assume that γ can be arbitrarily chosen from the set $\{-h, \ldots, h\}$ in each iteration of computation, then $\Delta \in \{-3\,h - 1, \ldots, 3\,h + 1\}$. Hence, our algorithm makes use of an array of $6\,h + 3$ elements that stores the required values of x^{Δ}.

Algorithm 7: Blinded Montgomery Powering Ladder

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = (k_{n-1}, \ldots, k_0)_2$, small integer $h \in \mathbb{Z}$
Output: x^{κ}
Uses: U a $6\,h + 3$ element array.

1 $U_{3\,h+1} \leftarrow 1_{\mathbb{G}}$;
2 **for** $i = 1$ **to** $3\,h + 1$ **do**
3 $\quad U_{3h+1+i} \leftarrow U_{3h+i} \cdot x$;
4 $\quad U_{3h+1-i} \leftarrow U_{3h+2-i} \cdot x^{-1}$;
5 **end**
6 $R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow 1_{\mathbb{G}}$; $\alpha = 0$;
7 **for** $i = n - 1$ **down to** 1 **do**
8 $\quad R_0 \leftarrow R_0 \cdot R_1$;
9 $\quad \gamma \xleftarrow{R} \{-h, \ldots, h\}$;
10 $\quad \ell \leftarrow \gamma - 2\,\alpha + k_i - \neg k_i$;
11 $\quad R_1 \leftarrow R_0 \cdot U_{3\,h+1+\ell}$;
12 $\quad \alpha \leftarrow \gamma$;
13 **end**
14 $R_0 \leftarrow R_0 \cdot R_1$;
15 $R_0 \leftarrow R_0 \cdot U_{3\,h+1-\alpha-\neg k_0}$;
16 **return** R_0

An adversary who is able to distinguish a multiplication from a squaring operation would not be able to determine any information on bits of the exponent used since no squaring operations take place. However, a cross correlation attack is possible by observing where the multiplication with U_d, for $d \in \{0, \ldots, 6\,h + 2\}$, collides with the multiplications used to generate these values in line 11. From these values, an adversary can derive the addition chain that was used to compute the exponentiation. This will not give an adversary the exponent, since there is no means to map the digits used to bits of the exponent, but would give an adversary an equivalent addition chain.

In determining whether an attack is practical Hanley et al. [18] determine that when analyzing an implementation 192-bit exponentiation, the error rate need to be less than 22 bits. This is determined to be less than 2^{54} operations, based on the boundary set for block ciphers by Biryukov et al. [23]. That is, a DES secret key has been found but larger exhaustive searches have, to date, been unsuccessful. While public

key operations are typically more time consuming to compute, one can increase the speed of exhaustive searches by using the multiplicative group structure [24], hence justifying the use of the same threshold. The expected number of operations can be determined using Stinson's algorithm [25], where an t-bit error in a n-bit hypothesis leads to the exponent in time complexity $\mathcal{O}\left(n \sum_{i=0}^{\lceil t/2 \rceil} \binom{n/2}{i}\right)$ [25]. Once a hypothesis for the exponent used in an implementation of the Montgomery ladder is deduced one cannot directly apply Stinson's algorithm. This is because each time one wishes to flip a bit all the bits that are less significant also need to be flipped. However, adapting the algorithm is straightforward and the expected time complexity of an attack will be the same. If an attacker were to target Algorithm 7 then the error rate would need to be much lower to fall below time complexity 2^{54}. In applying the baby-step giant-step algorithm to correct errors in hypotheses for digits used in Algorithm 7 one would divide the digits into two sets and attempt to find two divisions where the number of incorrect digits is the same. Given Lemma 3 (see Appendix A), we have the following corollary.

Corollary 1. *We consider set of μ digits representing the multiplications in an addition chain where t digits are incorrect. There will exist a set of $\mu/2$ contiguous digits where $\lfloor t/2 \rfloor$ digits are incorrect.*

Then one can apply a version of Stinson's algorithm where one has to treat all the values that digits can take. Given a small integer h, as defined in Algorithm 7, a t-digit error in an μ-bit hypothesis leads to the exponent in time complexity $\mathcal{O}\left(\mu \binom{\mu/2}{t/2} (2\,h+1)^{\frac{\mu}{2}}\right)$. That is, there are μ possible divisions of the digits, each of which will have $\binom{\mu/2}{t/2}$ ways of selecting $t/2$ digits and each digit can take $(2\,h+1)$ values. In practice, t will not be known so the analysis will have time complexity $\mathcal{O}\left(\mu \sum_{i=0}^{\lceil t/2 \rceil} \binom{\mu/2}{i} (2\,h+1)^i\right)$ further reducing the required error rate for a successful attack.

Hanley et al. [18] define a practical attack to have time complexity less than 2^{54} operations, based on the boundary set for block ciphers by Biryukov et al. [23], and a trivial attack to have time complexity less than 2^{40}. For a 192-bit exponent, this means that for single bit errors we can define $t \leq 21$ to be a practical attack and $t \leq 13$ to be a trivial attack. In Table 1 we show how these thresholds change as h is increased in Algorithm 7. Far fewer errors can be tolerated by an adversary before an attack becomes unfeasible even for very small values of h, although there are diminishing returns as h increases.

Table 1. Time complexity of Stinson's algorithm for small h.

h	Max Error for Trivial Case (bits)	Max Error for Practical Case (bits)	h	Max Error for Trivial Case (bits)	Max Error for Practical Case (bits)
1	8	14	6	6	10
2	8	12	7	6	10
3	8	12	8	6	10
4	6	10	9	6	8
5	6	10	10	6	8

We note that an adversary is required to derive the entire exponent as information on part of the exponent is not useful. Moreover, an adversary will not know the exponent but an equivalent addition-chain. If we consider the group exponentiation algorithm used in ECDSA (Elliptic Curve Digital Signature Algorithm) [26] this provides a significant increase in security. Howgrave-Graham and Smart [19] noted that if a few bits of the ephemeral exponent are known for sufficiently many signatures, then the scheme can be broken, based on the so-called hidden number problem introduced by Boneh and Venkatesan [27]. Recently, De Mulder et

al. [28] demonstrated that, in practice, the lattice attack required as few as four bits of the ephemeral exponent assuming that some hundreds of such signatures could be obtained. However, this information would not be available to an adversary.

A more memory-efficient version of Algorithm 7 is described in Algorithm 8. This is a security-memory trade off where the value of γ_j is still chosen at random, but the value range depend on the previous value γ_{j+1}. Although the available choices for γ_j decreases, Algorithm 8 requires only $2\,h+2$ registers to store pre-computed values instead of $6\,h+3$ registers in Algorithm 7.

Algorithm 8: Blinded Montgomery Powering Ladder II

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$, small integer $h \in \mathbb{Z}$, U a look-up table of $2h+2$ precomputed values $x^{-h}, \ldots, x^{h+1}$.

Output: x^{κ}

$U_h \leftarrow 1_{\mathbb{G}}$;
for $i = 1$ **to** h **do**
  $U_{h+i} \leftarrow U_{h+i-1} \cdot x$;
  $U_{h-i} \leftarrow U_{h+1-i} \cdot x^{-1}$;
end
$U_{2\,h+1} \leftarrow U_{2\,h} \cdot x$; $R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow 1_{\mathbb{G}}$; $\alpha \leftarrow 0$;
for $i = n-1$ **down to** 1 **do**
  $R_0 \leftarrow R_0 \cdot R_1$;
  if $\alpha \geq 0$ **then**
    $\gamma \xleftarrow{R} \{2\,\alpha - h, \ldots, h+1-k_i\}$;
  end
  else
    $\gamma \xleftarrow{R} \{-h-k_i, \ldots, 2\,\alpha + h - 1\}$;
  end
  $R_1 \leftarrow R_0 \cdot U_{h+\gamma+k_i}$;
  $\alpha \leftarrow 2\,\alpha - \gamma$;
end
$R_0 \leftarrow R_0 \cdot R_1$;
$R_0 \leftarrow R_0 \cdot U_{h+\alpha+k_0}$;
return R_0

5 Generalizing the Montgomery Powering Ladder

The *square-and-multiply* (left-to-right and right-to-left)) exponentiation algorithms are the most efficient implementations for raising x to the power κ when the exponent is treated bit-by-bit. Furthermore, these exponentiation algorithms extend easily to any radix m for the purpose of speeding the computation. In this section, by using the random walk technique, we generalize the (both left-to-right and right-to-left) exponentiation algorithms and present new blinded m-ary exponentiation algorithms.

5.1 Left-to-right algorithms

We first consider the left-to-right exponentiation. Let $\kappa = (w_{\ell-1}, w_{\ell-2}, \ldots, w_0)_2$ be the m-ary representation of κ, where $0 \neq w_i < m$, $w_{\ell-1} \neq 0$, and ℓ is the length of κ in radix m. As in Section 2.2, by defining $L_j = \sum_{i=j}^{\ell-1} w_i\, m^{i-j}$, we have

$$L_j = m\,L_{j+1} + w_j \tag{10}$$

From then, the exponentiation algorithm will perform $x^{L_j} = (x^{L_{j+1}})^m \cdot x^{w_j}$. Noting that $x^{L_0} = x^\kappa$. Using the random walk technique as in Section 4, we modify $L'_j = L_j + \gamma_j$ where γ_j can be chosen from a pre-defined set $\{-h, \ldots, h\}$ for some small integer h. Then (10) can be rewritten as

$$L_j + \gamma_j = m\,L_{j+1} + \gamma_j + w_j = m\,L'_{j+1} - m\,\gamma_{j+1} + \gamma_j + w_j \tag{11}$$

This leads to a randomized algorithm (Algorithm 9) that computes a group exponentiation taking a random, albeit tightly bounded, walk through the possible addition chains. As with Algorithm 8, Algorithm 9 also requires a look-up table of $2(m+1)\,h + m - 1$ values to stores precomputated values. Algorithm 9 can be used in the cases where the computation of inversions is cheap, otherwise Algorithm 10 is more suitable.

Algorithm 9: Randomized Left-to-Right m-ary Exponentiation

Input: $x \in \mathbb{G}$, an integer $\kappa = (w_{\ell-1}, w_{\ell-2}, \ldots, w_0)_m$, where $0 \leq w_i < m$ and $w_{\ell-1} \neq 0$, small integer $h \in \mathbb{Z}$, T a look-up table of $2(m+1)\,h + m - 1$ precomputed values $x^{-(m+1)h-m+1}, \ldots, x^{(m+1)h+m-1}$.
Output: x^κ

$R \leftarrow 1_\mathbb{G}$, $\alpha \leftarrow 0$
for $i = \ell - 2$ **to** 1 **do**
  $R \leftarrow R^m$;
  $\gamma \xleftarrow{R} \{-h, \ldots, h\}$;
  $R \leftarrow R \cdot x^{m\,\alpha - \gamma + w_i}$;
  $\alpha \leftarrow \gamma$;
end
$R \leftarrow R^m$;
$R \leftarrow R \cdot x^{m\,\alpha + w_0}$;
return R

Algorithm 10: Randomized Left-to-Right m-ary Exponentiation without Inversions

Input: $x \in \mathbb{G}$, an integer $\kappa = (w_{n-1}, w_{n-2}, \ldots, w_0)_m$, where w_i is m-bit words, small integer $h \in \mathbb{Z}$, T a look-up table of $m\,(h+1) - 1$ precomputed values $x, \ldots, x^{m(h+1)-1}$.
Output: x^κ

$R \leftarrow 1_\mathbb{G}$, $\alpha \leftarrow 0$
for $i = \ell - 2$ **to** 1 **do**
  $R \leftarrow R^m$;
  $\gamma \xleftarrow{R} \{0, \ldots, \min(h, m\,\alpha + w_i)\}$;
  $R \leftarrow R \cdot x^{m\,\alpha - \gamma + w_i}$;
  $\alpha \leftarrow \gamma$;
end
$R \leftarrow R^m$;
$R \leftarrow R \cdot x^{m\,\alpha + w_0}$;
return R

We note that Algorithm 10 works in a similar way to the Overlapping Windows method [29] for a fixed base $m = 2^{k - h_{OWM}}$ and $m(h+1) = 2^k$[3]. The main difference is that our algorithm generates *on-the-fly* a randomized recoding of the binary representation of the secret exponent κ. This allows our algorithm to avoid side-channel attacks in the recoding phase.

5.2 Right-to-left algorithm

Likewise, we can devise a randomized *right-to-left* m-ary algorithm. Let $\kappa = (w_{\ell-1}, w_{\ell-2}, \ldots, w_0)_m$, where $w_{\ell-1} \neq 0$, the principle of the *right-to-left* m-ary exponentiation algorithm, as shown by Yao [30], makes use of the is relied on the following equality:

$$x^\kappa = \prod_{\substack{0 \leq i \leq \ell-1 \\ d_i = 1}} x^{m^i} \cdot \prod_{\substack{0 \leq i \leq \ell-1 \\ d_i = 2}} x^{2 \cdot m^i} \cdots \prod_{\substack{0 \leq i \leq \ell-1 \\ d_i = m-1}} x^{(m-1) \cdot m^i} = \prod_{j=1}^{m-1} \Big(\prod_{\substack{0 \leq i \leq \ell-1 \\ d_i = j}} x^{m^i} \Big)^j .$$

[3] We use the notation h_{OWM} instead of h as in [29] to distinguish it from the notation h in our algorithms.

The *right-to-left* m-ary exponentiation algorithm uses $(m-1)$ accumulators, $R[1], \ldots, R[m-1]$. At each iteration, it applies w successive squarings to compute $A = x^{m^j}$ from $x^{m^{(j-1)}}$, then multiplies the result to some accumulators $R[k]$. Let $R[k]^{(j)}$ (resp. $A^{(j)}$) denote the value of the accumulator $R[k]$ (resp. A) before entering step j. We have:

$$\begin{aligned} R[k]^{(j+1)} &= R[k]^{(j)} \cdot A^{(j)} \quad \text{for } k = w_j\,, \\ R[k]^{(j+1)} &= R[k]^{(j)} \quad\quad\quad\quad \text{for } k \neq w_j\,, \end{aligned}$$

and $A^{(j+1)} = (A^{(j)})^m$. At the end of the loop each accumulator $R[k]$ contains the product $\prod_{\substack{0 \le j \le \ell-1 \\ w_j = k}} x^{m^j}$. The different accumulators are finally aggregated as $\prod_{0 \le j \le \ell-1} R[k]^k = x^\kappa$. By defining $L_j = \sum_{i=0}^{j} w_i \cdot m^i$, we have

$$L_{j+1} = m^{j+1} \cdot w_{j+1} + Lj \tag{12}$$

Noting that $L_{\ell-1} = \kappa$. Similar, we use the random walk technique and modify $L'_j = L_j - m^{j+1} \cdot \gamma_j$, then equation (12) can be rewritten as:

$$\begin{aligned} L_{j+1} - m^{j+2} \cdot \gamma_{j+1} &= m^{j+1} \cdot w_{j+1} + L_j - m^{j+2} \cdot \gamma_{j+1} \\ &= m \cdot w_{j+1} + L'_j + m^{j+1} \cdot \gamma_j - m^{j+2} \cdot \gamma_{j+1} \\ &= (\gamma_j - m \cdot \gamma_{j+1} + w_{j+1}) m^{j+1} + L'_j \end{aligned}$$

This leads to a randomized *right-to-left* m-ary exponentiation algorithm (Algorithm 11). In this algorithm, we make use of $2(m+1)\,h+m-1$ accumulators $R[j]$, where $-(m+1)h - m + 1 \le j \le (m+1)h + m - 1$. Each $R[j]$ is initialized to $1_{\mathbb{G}}$, and then updated if $\alpha - m\gamma + w_i = j$. The different accumulators are finally aggregated as $\prod_{j=1}^{h} \left(R[j] \cdot (R[-j])^{-1}\right)^j = x^\kappa$.

Algorithm 11: Randomized Right-to-Left m-ary Exponentiation

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = (w_{\ell-1}, w_{\ell-2}, \ldots, w_0)_m$, small integer $h \in \mathbb{Z}$.
Output: x^κ

for $j = 0$ **to** $(m+1)h + m - 1$ **do**
  $R[j] \leftarrow 1_{\mathbb{G}}$; $R[-j] \leftarrow 1_{\mathbb{G}}$;
end
$A \leftarrow x$; $\alpha \leftarrow 0$;
for $i = 0$ **to** $\ell - 1$ **do**
  $\gamma \xleftarrow{R} \{-h, \ldots, h\}$;
  $R[\alpha - m\gamma + k_i] \leftarrow R[\alpha - m\gamma + k_i] \cdot A$;
  $\alpha \leftarrow \gamma$; $A \leftarrow A^2$;
end
$A \leftarrow \prod_{j=1}^{h} \left(R[j] \cdot (R[-j])^{-1}\right)^j$;
return A

6 Conclusion

In this paper we presented variants of the Montgomery powering ladder which have properties that increase the side-channel resistance of the exponentiation algorithm. The first of these operates by randomly changing states such that the difference between the two registers varies, unpredictably, between two states. The second variant of the Montgomery powering ladder that we presented takes a random walk, albeit tightly bounded, among the possible addition chains required to compute an exponentiation. While this variant is not resistant to all side-channel analysis, it will prevent lattice-based attacks when used, for example, in implementations of ECDSA. In other cases, significantly more computation is required to derive any exploitable information and, therefore, an adversary requires a lower error rate to succeed. By applying the random walk method, we also generalized the Montgomery powering ladder and present randomized (both left-to-right and right-to-left) m-ary exponentiation algorithms.

References

1. Kocher, P.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Koblitz, N., ed.: CRYPTO '96. Volume 1109 of LNCS., Springer (1996) 104–113
2. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In Wiener, M.J., ed.: CRYPTO '99. Volume 1666 of LNCS., Springer (1999) 388–397
3. Quisquater, J.J., Samyde, D.: Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In Attali, I., Jensen, T.P., eds.: E-smart 2001. Volume 2140 of LNCS., Springer (2001) 200–210
4. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In Koç, C.K., Naccache, D., Paar, C., eds.: CHES 2001. Volume 2162 of LNCS., Springer (2001) 251–261
5. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In Joye, M., Quisquater, J.J., eds.: CHES 2004. Volume 3156 of LNCS., Springer (2004) 16–29
6. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In Oswald, E., Rohatgi, P., eds.: CHES 2008. Volume 5154 of LNCS., Springer (2008) 426–442
7. Fouque, P.A., Valette, F.: The doubling attack — Why upwards is better than downwards. In Walter, C.D., Koç, Ç.K., Paar, C., eds.: CHES 2003. Volume 2779 of LNCS., Springer (2003) 269–280
8. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In Jr., B.S.K., Koç, C.K., Paar, C., eds.: CHES 2002. Volume 2523 of LNCS., Springer (2002) 13–28
9. Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In Koç, C.K., Paar, C., eds.: CHES 1999. Volume 1717 of LNCS., Springer (1999) 292–302
10. Walter, C.D.: Sliding windows succumbs to big mac attack. In Ç. K. Koç, Naccache, D., Paar, C., eds.: CHES 2001. Volume 2162 of LNCS., Springer (2001) 286–299
11. Clavier, C., Feix, B., Gagnerot, G., Giraud, C., Roussellet, M., Verneuil, V.: ROSETTA for single trace analysis. In Galbraith, S., Nandi, M., eds.: INDOCRYPT 2012. Volume 7668 of LNCS., Springer (2012) 140–155
12. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal correlation analysis on exponentiation. In: Proceedings of the 12th international conference on Information and communications security. ICICS'10, Berlin, Heidelberg, Springer-Verlag (2010) 46–61
13. Bauer, A., Jaulmes, E., Prouff, E., Wild, J.: Horizontal collision correlation attack on elliptic curves. In Lange, T., Lauter, K., Lisonek, P., eds.: SAC 2013. Volume 8282 of LNCS., Springer (2013) 553–570
14. Witteman, M.F., van Woudenberg, J.G.J., Menarini, F.: Defeating RSA multiply-always and message blinding countermeasures. In Kiayias, A., ed.: CT-RSA 2011. Volume 6558 of LNCS., Springer (2011) 77–88
15. Kim, H., Kim, T.H., Yoon, J.C., Hong, S.: Practical second-order correlation power analysis on the message blinding method and its novel countermeasure for RSA. ETRI Journal **32**(1) (2010) 102–111
16. Montgomery, P.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**(177) (1987) 243–264
17. Joye, M., Yen, S.M.: The Montgomery powering ladder. In Jr., B.S.K., Ç. K. Koç, Paar, C., eds.: CHES 2002. Volume 2523 of LNCS., Springer (2003) 291–302
18. Hanley, N., Kim, H., Tunstall, M.: Exploiting collisions in addition chain-based exponentiation algorithms using a single trace. In Nyberg, K., ed.: CT-RSA 2015. Volume 9048 of LNCS., Springer (2015) 431–448

19. Howgrave-Graham, N.A., Smart, N.P.: Lattice attacks on digital signature schemes. Design, Codes and Cryptography **23** (2001) 283–290
20. Joye, M.: Highly regular right-to-left algorithms for scalar multiplication. In Paillier, P., Verbauwhede, I., eds.: CHES 2007. Volume 4727 of LNCS., Springer (2007) 135–147
21. Win, E.D., Mister, S., Preneel, B., Wiener, M.: On the performance of signature schemes based on elliptic curves. In Buhler, J.P., ed.: ANTS 1998. Volume 1423 of LNCS., Springer (1998) 252–266
22. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks — Revealing the Secrets of Smart Cards. Springer (2007)
23. Biryukov, A., Dunkelman, O., Keller, N., Khovratovich, D., Shamir, A.: Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds. In Gilbert, H., ed.: EUROCRYPT 2010. Volume 6110 of LNCS., Springer (2010) 299–319
24. Bernstein, D.J., Lange, T.: Two grumpy giants and a baby. Cryptology ePrint Archive, Report 2012/294 (2012) `http://eprint.iacr.org/`.
25. Stinson, D.: Some baby-step giant-step algorithms for the low Hamming weight discrete logarithm problem. Mathematics of Computation **71**(237) (2002) 379–391
26. X9.62, A.: Public key cryptography for the financial services industry, the elliptic curve digital signature algorithm (ECDSA) (1999)
27. Boneh, D., Venkatesa, R.: Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In Koblitz, N., ed.: CRYPTO '96. Volume 1109 of LNCS., Springer (1996) 129–142
28. Mulder, E.D., Hutter, M., Marson, M.E., Pearson, P.: Using bleichenbacher"s solution to the hidden number problem to attack nonce leaks in 384-bit ECDSA. In Bertoni, G., Coron, J.S., eds.: CHES 2013. Volume 8086 of LNCS., Springer (2013) 435–452
29. Itoh, K., Yajima, J., Takenaka, M., Torii, N.: Dpa countermeasures by improving the window method. In: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems. CHES '02, London, UK, UK, Springer-Verlag (2003) 303–317
30. Yao, A.C.C.: On the evaluation of powers. SIAM J. Comput. **5**(1) (1976) 100–103

A The Discrete Logarithm Problem

In this section, we describe how the errors in the attacks given in this paper can be corrected so that an adversary could derive an unknown exponent. To start, we recall the discrete logarithm problem:

Definition 3. *Let $\alpha \in G$, for some Abelian group G, and suppose $\alpha \in \langle\beta\rangle$. The discrete logarithm $\log_\alpha \beta$ is the unique integer x such that $0 \leq x \leq ord(\alpha) - 1$ and $\alpha^x = \beta$. The Discrete Logarithm Problem (DLP) is to compute $\log_\alpha \beta$, given α and β.*

In a side-channel analysis of a given instance of an exponentiation algorithm, the results can only give the best guess of the exponent. Stinson describes a variant of the Baby-Step/Giant-Step algorithm, where it is assumed that the exponent has a small Hamming weight [25]. Stinson's algorithm requires the existence of a means of splitting a string of bits into two sets of equal Hamming weight.

Lemma 3. *We consider an integer of bit length n, as a string of bits of length $n \in 2\,\mathbb{Z}$ and Hamming weight $0 < t < n$. There will exist a set of contiguous bits with Hamming weight $\lfloor t/2 \rfloor$.*

We present a somewhat simplified version of Stinson's proof:

Proof. We begin with the case where t is even. Let X be an string of bits of length n with Hamming weight $t \in 2\,\mathbb{Z}$. Let each Y_i for $i \in \{1, \ldots, n/2\}$ represent one of the $n/2$ sets of contiguous bits starting from the i-th bit of the string. Let $\mathcal{H}$ be a function that returns the Hamming weight, then $\mathcal{H}(Y_1) = t - \mathcal{H}(Y_{n/2})$. Given that $\mathcal{H}(Y_i) - \mathcal{H}(Y_{i+1})$ will be in $\{-1, 0, 1\}$, there will exist be some set of contiguous bits with Hamming weight $n/2$. If t is odd, then the first bit of the bit string can be set to zero putting us the case described above. The bit can be returned to one once two sets of equal Hamming weight are found. Giving one set of Hamming weight $\lfloor n/2 \rfloor$ and the other of $\lceil n/2 \rceil$. □

This is sufficient for our requirements. We refer the reader to Stinson for versions of this proof where n is odd [25].

Given an estimate for the exponent x' where $x = x' \oplus e$, for some unknown e of Hamming weight t, we can attempt to determine x by guessing e. We let z_i denote the ith bit of z for an n-bit number z. Given an n-bit number z, we define the vector $\mathring{z}$ as follows

$$\mathring{z}_i = \begin{cases} 0 & \text{If } z_i = 0\,, \\ 1 & \text{If } z_i = 1 \text{ and } x'_i = 0\,, \\ -1 & \text{If } z_i = 1 \text{ and } x'_i = 1\,. \end{cases}$$

For a vector $\mathring{z}$, we define

$$g^{\mathring{z}} = \prod_{i=1}^{n} g^{\mathring{z}_i \cdot 2^{n-i}} \quad .$$

If we set $\beta' = \alpha^{x'}$, then given a proposed value of e, such that $x = x' \oplus e$, we can test whether it is correct by checking whether we have $\beta = \beta' \cdot \alpha^{\mathring{e}}$. The error e can be divided into two sets e_1 and e_2, where e_1 and e_2 have a Hamming weight of $t/2$ given by a splitting algorithm. We also define a and b as two integers such that $x' = a + b$ and the only bits that can be set to one for a and b are at the indexes defined by the splitting algorithm for e_1 and e_2, respectively. Then, $\alpha^x = (\alpha^a\, \alpha^{\mathring{e}_1})(\alpha^b\, \alpha^{\mathring{e}_2})$.

We produce a list of error vectors of Hamming weight $t/2$, where we define the i-th error from the set of possible errors e_1 as $e_{i,1}$. We define the Giant-Steps to be the table that consists of all pairs $\left(\frac{\beta}{\alpha^a\, \alpha^{\mathring{e}_{i,1}}}, a + \mathring{e}_{i,1}\right)$, for all $e_{i,1}$. We define the Baby-Steps as pairs $\left(\alpha^b\, \alpha^{\mathring{e}_{j,2}}, b + \mathring{e}_{j,2}\right)$, for all $e_{j,2}$. As in the Baby-Step/Giant-Step method, we can terminate the method when a collision is found between $\left(\frac{\beta}{\alpha^a\, \alpha^{\mathring{e}_{i,1}}}\right)$ and $\left(\alpha^b\, \alpha^{\mathring{e}_{j,2}}\right)$ for a given i, j. We can then derive the exponent as $x = (a + \mathring{e}_{i,1}) + (b + \mathring{e}_{j,2})$.

For an n-bit exponent, one would be required to compute $\binom{n}{t/2}$ Giant-Steps and $\binom{n}{t/2}$ Baby-Steps for an error of Hamming weight t. The above assumes that t is even. If t is odd, then the extra bit can be assigned, arbitrarily, to the computation of baby steps. The required computation then becomes $\binom{n}{\lfloor t/2 \rfloor}$ Giant-Steps and $\binom{n}{\lfloor t/2 \rfloor + 1}$ Baby-Steps for an error of Hamming weight t.

Other than the inclusion of an initial guess, this algorithm is the same as that defined by Stinson [25], and has time complexity of $\mathcal{O}\left(n \binom{n/2}{t/2}\right)$. However, this assumes that t is known. If t is not known, then an adversary has to start with $t = 1$ and increase the Hamming weight until t is found. One would expect the resulting time complexity to be $\mathcal{O}\left(n \sum_{i=0}^{t} \binom{n/2}{i/2}\right)$. However, by Lemma 3, we can ignore the cases where i is odd since the required baby and giant steps will be computed for the cases $i - 1$ and $i + 1$. The resulting time complexity is therefore $\mathcal{O}\left(n \sum_{i=0}^{\lceil t/2 \rceil} \binom{n/2}{i}\right)$ when t is unknown.