Exploiting Collisions in Addition Chain-based Exponentiation Algorithms Using a Single Trace

Neil Hanley[1], HeeSeok Kim[2,3], and Michael Tunstall[2]

[1] The Institute of Electronics, Communications and Information Technology
Queen's University Belfast, Queen's Island
Queens Road, Belfast BT3 9DT, United Kingdom
n.hanley@qub.ac.uk

[2] Department of Computer Science, University of Bristol
Merchant Venturers Building, Woodland Road
Bristol BS8 1UB, United Kingdom
mike.tunstall@bristol.ac.uk

[3] Center for Information Security Technologies, Korea University
Seoul, Republic of Korea
80khs@korea.ac.kr

Abstract. Public key cryptographic algorithms are typically based on group exponentiation algorithms where the exponent is private. A collision attack is typically where an adversary seeks to determine whether two operations in an exponentiation have the same input. In this paper we extend this to an adversary who seeks to determine whether the output of one operation is used as the input to another. We describe implementations of these attacks to a 192-bit scalar multiplication over an elliptic curve that only require a single power consumption trace to succeed with a certain probability. All the implementations of similar attacks described in the literature require an adversary to generate a mean trace from numerous traces to succeed. Moreover, our attacks do not require any knowledge of the input to the exponentiation algorithm. These attacks would therefore be applicable to EC-DSA, where an exponent is ephemeral, or to implementations where an exponent is blinded. Moreover, we prove that collision attacks are applicable to all addition chain-based exponentiation algorithms. This means that a side-channel resistant implementation of a group exponentiation algorithm will require countermeasures that introduce enough noise such that an attack is not practical.

1 Introduction

It has been shown in the literature that an adversary can potentially derive a private key used to generate a RSA [1] or EC-DSA [2] signature by observing the power consumption during the computation of a naïvely implemented group exponentiation algorithm. The attack, defined by Kocher et al. [3], targets implementations of group multiplication and squaring operations used in the binary exponentiation algorithm. This vulnerability was present because the instantaneous power consumption during the computation of a group squaring operation was different to that of a group multiplication, and could, therefore, be distinguished by simply inspecting a power consumption trace. Many group exponentiation algorithms have since been proposed in the literature, designed such that an adversary cannot implement side-channel attacks by inspecting a power consumption trace (see [4] for a survey).

In this paper we present attacks that are applicable to scalar multiplication algorithms that one would, for example, use in implementing EC-DSA [2] on a resource constrained device that is potentially vulnerable to side-channel analysis. We exploit collisions between variables used in group operations required to compute a scalar multiplication. Analyzing collisions in variables was first proposed by Schramm et al. [5] to attack block ciphers (see [6] for recent results and a survey of relevant publications).

Typically, one would assume that an input point is blinded such that the input to an exponentiation algorithm is not known to an adversary. Okeya and Sakura [7] observed that one could potentially derive the bits of an exponent by comparing two parts of the same power consumption trace corresponding to two operations in a group exponentiation. Amiel and Feix [8] and Kim et al. [9] describe some attacks and implementations of this strategy applied to the BRIP exponentiation algorithm [10]. A similar attack has been described by Witteman et al. [11] applicable to Coron's

double and add always exponentiation algorithm. Kim et al. [9] also describe how one could apply such an attack the the Montgomery ladder [12, 13].

One feature of all these attacks is that an adversary will attempt to determine whether the input to two group operations is the same [7–11]. In this paper we extend this to consider an adversary who is able to determine whether the output of one operation is used as the input to another operation. Thus, allowing an adversary to apply collision attacks to more algorithms and overcoming the countermeasures described by Kim et al. [9] to protect the Montgomery ladder.

The common requirement for the the attacks described in the literature is that the adversary needs numerous traces with the same input and the same exponent to derive an unknown exponent using collisions [7–11]. That is, an adversary will take numerous acquisitions and reduce noise in the acquired power consumption traces by computing a mean trace. In this paper we describe implementations of attacks described in the literature, and introduce an extended attack model which only requires a single trace to succeed.

Given that our practical work justifies our extended adversary model for collision attacks, we then prove that it is not possible to define a countermeasure that prevents collision attacks being applied to an addition chain-based exponentiation algorithm.

The rest of this paper is organised as follows. In Section 2 we give preliminary definitions. In Section 3 we discuss the previous work related to the attacks described in this paper. In Section 4 we define our attacks, and describe our experiments in Section 5. In Section 6 we prove that the results can be applied to any exponentiation algorithm. We conclude in Section 7.

2 Preliminaries

In this section we provide some background and notation that we will refer to in later sections.

2.1 Addition Chain-based Exponentiation

Addition chain-based exponentiation methods are used to find a sequence of elements, in a given multiplicative group $\mathbb{G}$, such that the first number is $1_{\mathbb{G}}$ and the last is n^{λ} for some arbitrary $n \in \mathbb{G}$ and $\lambda \in \mathbb{Z}_{>0}$. Each member of the sequence is the product of two previous members of the sequence. A more formal definition is given below.

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

Kocher et al. [3] demonstrated that under some circumstances it is possible to distinguish a multiplication from a squaring operation using some side-channel, thus revealing all or part of the exponent. To counter this many highly regular exponentiation algorithms have been proposed in the literature [13, 14]. We define highly regular as the following:

Definition 2. *An exponentiation algorithm is defined as highly regular if it consists of an operation g that is composed $g \circ \cdots \circ g \circ g$, where g occurs $\kappa \in \mathbb{Z}_{>0}$ times and κ is fixed for an exponent of a given bit length.*

We describe some examples of highly regular exponentiation algorithms alongside our attacks in Section 4.

2.2 Elliptic Curves

Let $\mathbb{F}_q$ be a finite field. An elliptic curve $\mathcal{E}$ over $\mathbb{F}_q$ consists of points (x, y), with x, y in $\mathbb{F}_q$, that satisfy the full Weierstraß equation

$$\mathcal{E} : y^2 + a_1\, x\, y + a_3\, y = x^3 + a_2\, x^2 + a_4\, x + a_6$$

with $a_i \in \mathbb{F}_q$ ($1 \le i \le 6$), and the point at infinity denoted $\mathcal{O}$. The set $\mathcal{E}(\mathbb{F}_q)$ is defined as

$$\mathcal{E}(\mathbb{F}_q) = \{(x, y) \in \mathcal{E} \mid x, y \in \mathbb{F}_q\} \cup \{\mathcal{O}\},$$

where $\mathcal{E}(\mathbb{F}_q)$ forms an Abelian group under the chord-and-tangent rule and $\mathcal{O}$ is the identity element.

The addition of two points $\boldsymbol{P} = (x_1, y_1)$ and $\boldsymbol{Q} = (x_2, y_2)$ with $\boldsymbol{P} \ne -\boldsymbol{Q}$ is given by $\boldsymbol{P} + \boldsymbol{Q} = (x_3, y_3)$ where

$$x_3 = \lambda^2 + a_1\,\lambda - a_2 - x_1 - x_2, \quad y_3 = (x_1 - x_3)\lambda - y_1 - a_1\,x_3 - a_3 \tag{1}$$

with $\lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{if } \boldsymbol{P} \ne \boldsymbol{Q}, \\[2mm] \dfrac{3\,x_1{}^2 + 2\,a_2\,x_1 + a_4 - a_1\,y_1}{2\,y_1 + a_1\,x_1 + a_3} & \text{if } \boldsymbol{P} = \boldsymbol{Q}. \end{cases}$

Provided that the characteristic of field $\mathbb{F}_q$ is not 2 or 3, we can take $a_1 = a_2 = a_3 = 0$. In the sequel we will also assume that $q = p$ is prime. We define the short Weierstraß form over prime field $\mathbb{F}_p$ by the equation

$$y^2 = x^3 + a\,x + b \ . \tag{2}$$

Note that the slope λ in the doubling then becomes $\lambda = (3\,x_1{}^2 + a)/(2\,y_1)$, which can be rewritten as $3(x_1 - 1)(x_1 + 1)/(2\,y_1)$ when $a = -3$.

The scalar multiplication of a given point is equivalent to an addition chain-based exponentiation and is a fundamental operation in cryptographic algorithms that use elliptic curve arithmetic, i.e. $k\,\boldsymbol{P}$ for some integer $k < |\mathcal{E}|$. In this paper we concentrate on the short Weierstraß form since it is used in standards such as the FIPS 186-3 [2], WTLS [15] and ANSI X9.62 [16].

3 Previous Work

One of the first examples of a collision attack applicable to public key cryptographic algorithms is referred to as the doubling attack, proposed by Fouque and Valette [17], which requires an adversary to acquire traces of a suitable side channel during the computation of two scalar multiplications where the input messages are chosen. Specifically, the scalar multiplication would be made to operate on the chosen points $\boldsymbol{P}$ and $2\,\boldsymbol{P}$. An adversary could then focus on adjacent operations executed during the computation of $k\,\boldsymbol{P}$ and $k(2\,\boldsymbol{P})$. If we consider, for example, Coron's double-and-add-always algorithm [18] (see Algorithm 1), an adversary would seek to compare the doubling operations in steps $i - 1$ and i of the computation of $k\,\boldsymbol{P}$ and $k(2\,\boldsymbol{P})$ respectively. When the side channel information acquired during the computation of these doubling operations are observed to be identical (taking into account a certain amount of noise), it can be assumed that the two doubling operations have the same input point. This would allow an adversary to deduce that the i-th bit of k is equal to zero, since otherwise the addition in the i-th round will mean that one doubling operation is operating on $x\,\boldsymbol{P}$ and the other on $(x + 1)\,\boldsymbol{P}$ for some integer x. The authors show that two of Coron's three countermeasures [18] do not provide any resistance against this type of attack. The extension of this attack to exponentiation in $\mathbb{Z}_n$ is straightforward, but it does not apply to right-to-left or non-redundant exponentiation algorithms.

Yen at al. [19] extended the attack proposed by Fouque and Valette [17] to apply it to the Montgomery ladder [12, 13]. The attack functioned using the same chosen input as described above, i.e. points $\boldsymbol{P}$ and $2\,\boldsymbol{P}$, but an adversary is only able to deduce whether two adjacent bits of the scalar are equal (if a collision occurred) or not (otherwise).

A more recent result on the doubling attack has been presented by Homma et al. [20]. The proposed attack explores a collision between two traces at arbitrary (not necessary adjacent) steps in modular exponentiation algorithms. The idea is to use two input messages Y and Z that satisfy the equation $Y^\alpha = Z^\beta \mod N$, where α and β are given constants. The value of α is chosen to provide information on a certain bit, and $\beta \le \alpha$ is chosen arbitrarily. This attack requires two power traces for each bit of the exponent, and the application to elliptic curve scalar multiplication algorithms is straightforward. This attack also applies to both left-and-right and right-to-left exponentiation algorithms

including m-ary and sliding window methods. In addition, Homma et al. demonstrated the effectiveness of their attacks on a PowerPC processor and a FPGA.

The above attacks require that an adversary is able to choose the input to a group exponentiation algorithm. Typically, one would assume that an input point is blinded such that the input to an exponentiation algorithm is not known to an adversary. Okeya and Sakura [7] observed that one could potentially derive the bits of an exponent by comparing two parts of the same power consumption trace corresponding to two operations in a group exponentiation. Amiel and Feix [8], and Kim et al. [9] describe some attacks implementing the strategy proposed by Okeya and Sakura. They describe a collision attacks on the BRIP exponentiation algorithm [10]. They observed that in the i-th round the first operation will operate on the same value as the last operation in round $i-1$ depending on the value of bits of the exponent. An adversary can then compute the correlation between two power consumption traces taken while computing these operations to determine if the same value was used. A similar attack has been described by Witteman et al. [11] applicable to Coron's double and add always exponentiation algorithm. Witteman et al. demonstrate that this attack works on an ASIC implementation. However, all the attacks described in the literature require numerous acquisitions to be taken to reduce noise to the point where an attack will function.

Kim et al. [9] also describe how one could apply such an attack to the Montgomery ladder [12, 13]. We describe this attack in more detail in Section 4.3. Kim et al. also describe how one could prevent these collision attacks by having a random structure that will vary form one execution to another. We show how this countermeasure can be overcome in Section 4.3.

The work described above either requires an adversary to choose the input to a group exponentiation algorithm or to acquire a set of traces where the same exponent is used. In the attacks described in this paper we shall assume that the input is unknown and that the exponent will be different for each execution of the group exponentiation algorithm. This provides a more realistic model than considered previously, since one would expect a secure implementation to use blinded inputs and a blinded (or ephemeral) exponent.

4 Attacking Addition Chain-based Exponentiation Algorithms

In this section we describe some attacks that could be applied to an elliptic curve group exponentiation algorithms using collisions. Two principle strategies appear in the literature that one could use to protect a group exponentiation algorithm from side channel analysis.

1. The first countermeasure to protect a group exponentiation algorithm was proposed by Coron [18], where dummy operations are included to provide a regular structure to a group exponentiation algorithm.
2. A set of highly-regular algorithms have been defined where a group exponentiation algorithm is broken up into indistinguishable atomic operations, as first described by Chevallier-Mames et al. [21].

In the remainder of this section we consider one example of each of these strategies to define an attack an adversary could use that would be applicable to many other algorithms defined in the literature. We also consider the Montgomery ladder [12, 13] since the version described resists all the collision-based attacks described in the literature, and we describe a novel attack against this version. We describe some implementations of these attacks using a single trace in Section 5 and generalise the method in Section 6.

4.1 Scalar Multiplication with Dummy Operations

The first countermeasure defined to prevent the identification of group operations was proposed by Coron [18]. In the additive group formed over an elliptic curve the resulting algorithm is shown in Algorithm 1. Witteman et al. [11] describe an attack based on the observation that in the i-th round the first operation will operate on the same value as the last operation in round $i - 1$ if the output of this operation is discarded under the assumption that the operation is the same.

Algorithm 1 Coron's Double-and-Add-Always Algorithm *[18]*

__Require:__ P a point on elliptic curve $\mathcal{E}$, an n-bit scalar $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
__Ensure:__ $Q = kP$
  1: $R_0 \leftarrow P$
  2: $R_1 \leftarrow P$
  3: __for__ $i \leftarrow n - 2$ __down to__ 0 __do__
  4: $R_0 \leftarrow 2R_0$
  5: $b \leftarrow 1 - k_i$
  6: $R_b \leftarrow R_0 + P$
  7: __end for__
  8: __return__ R_0

Lemma 1. *In one loop of Algorithm 1, if the bit treated in the loop is equal to zero the input to the dummy operation R_0 will be the same as the input to the subsequent doubling operation.*

Proof. Trivially, we assume that an adversary knows the ℓ most significant bits of the scalar. Then in the $(\ell + 1)$-th loop R_0 remains the same when $(\ell + 1)$-th most significant bit of the scalar is equal to 0 and changes when it is equal to 1.

Given Lemma 1, an adversary can make a comparison between two subtraces from a power consumption trace taken while computing these operations to determine if the same value was used. Witteman et al. required numerous acquisitions to be taken to reduce noise to the point where an attack will function. For this attack to function on an implementation of a group exponentiation over an elliptic curve, a method of comparing an addition and a doubling operation will be required since one typically uses different algorithms to compute these operations.

4.2 Highly-Regular Right-to-Left Scalar Multiplication

In the additive group formed over an elliptic curve, a group exponentiation has been defined by Joye that only uses additions [14] as shown in Algorithm 2. The attack described below is a straightforward variant of the attacks described by Kim et al. [9] and Witteman et al. [11] applied to a different algorithm.

Algorithm 2 Joye's Add-Only Scalar Multiplication *[14]*

Require: P *a point on elliptic curve* $\mathcal{E}$, *an* n-*bit scalar* $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Ensure: $Q = k\,P$
1: $R_0 \leftarrow \mathcal{O}$
2: $R_1 \leftarrow P$
3: $R_2 \leftarrow P$
4: **for** $i \leftarrow 0$ **to** $n - 1$ **do**
5:     $b \leftarrow 1 - k_i$
6:     $R_b \leftarrow R_b + R_2$
7:     $R_2 \leftarrow R_0 + R_1$
8: **end for**
9: **return** R_0

Lemma 2. *In one loop of Algorithm 2,* R_0 *remains the same when* $k_{\ell+1} = 0$ *and* R_1 *remains the same when* $k_{\ell+1} = 1$.

Proof. We assume that an adversary knows the ℓ least significant bits of the scalar, and define $k' = (k_{\ell-1} \ldots k_0)_2$ then we wish to show that the $(\ell + 1)$-th bit can be derived. The intermediate values in R_0, R_1 and R_2 at the end of the ℓ-th loop will be

$$R_0 = k'\,P, \quad R_1 = (2^\ell - k')\,P \quad \text{and} \quad R_2 = 2^\ell\,P\,.$$

In the $(\ell + 1)$-th loop we consider the change in the registers R_0 and R_1:

If $k_{\ell+1} = 0$ then, at the end of the ℓ-th round $R_0 = k'\,P$ and $R_1 = (2^\ell - k')\,P$ changes to

$$R_0 = k'\,P \quad \text{and} \quad R_1 = (2^{\ell+1} - k')\,P$$

in the $(\ell + 1)$-th round. If $k_{\ell+1} = 1$ then, at the end of the ℓ-th round $R_0 = k'\,P$ and $R_1 = (2^\ell - k')\,P$ changes to

$$R_0 = (2^{\ell+1} + k')\,P \quad \text{and} \quad R_1 = (2^\ell - k')\,P$$

in the $(\ell + 1)$-th round. We can see that R_0 remains the same when $k_{\ell+1} = 0$, while R_1 remains the same when $k_{\ell+1} = 1$.

4.3 Montgomery Ladder

Another variant of addition chain-based exponentiation is the Montgomery ladder [12, 13] as shown in Algorithm 3.

Algorithm 3 Montgomery Ladder *[12, 13]*

Require: $\boldsymbol{P}$ *a point on elliptic curve $\mathcal{E}$, an n-bit scalar $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$*
Ensure: $\boldsymbol{Q} = k\,\boldsymbol{P}$
1: $\boldsymbol{R}_0 \leftarrow \mathcal{O}$
2: $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$
3: **for** $i = n - 1$ **down to** 0 **do**
4: $\quad \boldsymbol{R}_{\neg k_i} \leftarrow \boldsymbol{R}_{k_i} + \boldsymbol{R}_{\neg k_i}$
5: $\quad \boldsymbol{R}_{k_i} \leftarrow 2\,\boldsymbol{R}_{k_i}$
6: **end for**
7: **return** $\boldsymbol{R}_0$

A method of attacking the Montgomery ladder is described by Kim et al. [9]. However, it requires that Line 4 in Algorithm 3 is

$$\boldsymbol{R}_{\neg k_i} = \boldsymbol{R}_0 + \boldsymbol{R}_1 \,.$$

This is because they used the collision between input operands of addition and doubling operation. In each loop their implementation computes $\boldsymbol{R}_0 + \boldsymbol{R}_1$ and $\boldsymbol{R}_0 + \boldsymbol{R}_0$ when a bit of the scalar is 0, and $\boldsymbol{R}_0 + \boldsymbol{R}_1$ and $\boldsymbol{R}_1 + \boldsymbol{R}_1$ when a bit of the scalar is 1. That is, the second operands are different when a given bit of a scalar is equal to 0 and equal otherwise.

This attack described by Kim et al. [9] cannot be applied to Algorithm 3 as described since Line 4 is

$$\boldsymbol{R}_{\neg k_i} \leftarrow \boldsymbol{R}_{k_i} + \boldsymbol{R}_{\neg k_i} \,.$$

However, one can still observe collisions between variables:

Lemma 3. *In Algorithm 3, if the bits treated in two consecutive loops are the same then the output of the operation in Line 5 in the first loop will be the input to the operation in Line 5 in the second loop.*

Proof. We assume that an adversary knows the ℓ most significant bits of the scalar, and define $k' = (k_{n-1} \ldots k_{n-\ell})_2$, and the intermediate results at the end of the ℓ-th loop will be

$$\boldsymbol{R}_0 = k'\,\boldsymbol{P}, \quad \text{and} \quad \boldsymbol{R}_1 = (k' + 1)\,\boldsymbol{P}\,.$$

If $k' = 2\,u$, for some $u \in \mathbb{Z}_{>0}$, then the bit $k_{n-\ell} = 0$ and $\boldsymbol{R}_0 = k'\,\boldsymbol{P}$ is computed by doubling $u\,\boldsymbol{P}$ and then, if the next bit is also equal to zero, the following loop will compute the doubling operation

$$\boldsymbol{R}_0 \leftarrow 2\,(k'\,\boldsymbol{P})\,, \quad \text{otherwise} \quad \boldsymbol{R}_0 \leftarrow k'\,\boldsymbol{P} + (k' + 1)\,\boldsymbol{P}\,.$$

Likewise, if $k' = 2\,u' + 1$, for some $u' \in \mathbb{Z}_{>0}$, then the bit $k_{n-\ell} = 1$ and $\boldsymbol{R}_1 = (k' + 1)\,\boldsymbol{P}$ is computed by doubling $((u+1)\,\boldsymbol{P})$ and, then if the next bit is also equal to one, the following loop will compute

$$\boldsymbol{R}_1 \leftarrow 2\,((k' + 1)\,\boldsymbol{P})\,, \quad \text{otherwise} \quad \boldsymbol{R}_1 \leftarrow k'\,\boldsymbol{P} + (k' + 1)\,\boldsymbol{P}\,.$$

That is, for a given value of the bit $k_{n-\ell}$ then the output of the doubling operation in the ℓ-th round will be used as the input to the doubling operation in the next round if the next bit of the scalar is equal to $k_{n-\ell}$.

Lemma 4. *In Algorithm 3, if the bits treated in two consecutive loops are different then the output of the operation in Line 4 in the first loop will be the input to the operation in Line 5 in the second loop.*

Proof. We assume that an adversary knows the ℓ most significant bits of the scalar, and define $k' = (k_{n-1} \ldots k_{n-\ell})_2$, and the intermediate results at the end of the ℓ-th loop will be

$$R_0 = k' P, \quad \text{and} \quad R_1 = (k'+1) P.$$

If $k' = 2u + 1$, for some $u \in \mathbb{Z}_{>0}$, then the bit $k_{n-\ell} = 1$ and $R_0 = k' P$ is computed by the addition $u P + (u+1) P$ and then, if the next bit is also equal to zero, the following loop will compute doubling operation

$$R_0 \leftarrow 2 (k' P) \quad \text{otherwise} \quad R_0 \leftarrow k' P + (k'+1) P.$$

Likewise, if $k' = 2u'$, for some $u' \in \mathbb{Z}_{>0}$, then the bit $k_{n-\ell} = 0$ and $R_1 = (k'+1) P$ is computed by the addition $u P + (u+1) P$ and, then if the next bit is also equal to one, the following loop will compute

$$R_1 \leftarrow 2 ((k'+1) P).$$

That is, for a given value of the bit $k_{n-\ell}$ then the output of the addition in the ℓ-th round will be used as the input to the doubling operation in the next round if the next bit of the scalar is not equal to $k_{n-\ell}$.

Given Lemma 3 and Lemma 4 an attack is not straightforward since one cannot compare field operations directly. This is because one wishes to compare the input of one operation with the output of another operation. We describe how one can achieve this in the following section.

5 Case Studies: Implementing the Attacks

In this section we describe how one can attack different group exponentiation algorithms using collisions between group operations. For each attack we shall assume that a adversary is obliged to attack individual traces independently rather than build up knowledge from numerous traces. This represents a significant advantage over the attacks described in the literature that typically require numerous traces to succeed [7–11].

In the following we describe instances of the attacks defined in Section 4 using a single power consumption trace. The aim in each case is to produce a hypothesis for the scalar that will enable the actual scalar to be derived in a practical amount of time. This is determined to be less than 2^{54} operations, based on the boundary set for block ciphers by Biryukov et al. [22]. This can only be considered to be an approximate guide since public key algorithms typically require significantly more time to compute than a block cipher. We also define an attack of time complexity less than 2^{40} to be trivial (while not strictly trivial this represents an arbitrary threshold for what could be achieved with modest resources). We note that these attacks typically cannot determine the first and last bit treated by the exponentiation algorithm, which also impacts the time complexity of the subsequent analysis.

The expected number of operations can be determined using Stinson's algorithm [23], where a t-bit error in a m-bit hypothesis leads to the expoentn in time complexity $\mathcal{O}\left(m\sum_{n=0}^{\lceil t/2\rceil}\binom{m/2}{n}\right)$ (see the Appendix for details). Hence, we define $t \leq 21$ to be a practical attack and $t \leq 13$ to be a trivial attack.

Once a guess 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.

5.1 Attack Platforms

We focus on implementations of a scalar multiplication over elliptic curves because of their use in standards, as described in Section 2.2. Specifically, we discuss attacks on a scalar multiplication where the scalar and coordinates are 192-bit values given in one of the example curves in the FIPS 186-3 standard [2]. In each implementation we use projective coordinates [24] where the base point is blinded by replacing the z-coordinate with a 192-bit random value and modifying the x and y-coordinates as required.

For each of three exponentiation algorithms suitable for use in resource-constrained devices we describe attack strategies for attacking implementations on two platforms. The purpose of these implementations is to demonstrate the accuracy of the attack on two different platforms. The implementation choices were different in each case since the intention is not to evaluate the platforms. The details of the two platforms are as follows:

ARM7 An embedded software implementation on an ARM7TDMI microprocessor, using homogenous projective coordinates and the point addition algorithm proposed by Izu and Takagi [25]. The implementation was based on a series of functions for manipulating large numbers written in the ARM7 assembly language. The multiplication instructions were used such that no change in the execution time would occur for different inputs by using algorithms defined by Großschädl et al. [26]. These functions were used as part of an implementation in C.

Of the implementations discussed below, the implementations of Joye's add-only [14] and Coron's double-and-add-always [18] algorithms used Montgomery multiplication [27] as the basic operation. The implementation of the Montgomery ladder [12, 13] discussed below used a standard multi-precision multiplication followed by a rapid reduction algorithm, made possible since the underlying prime field is a generalized Mersenne prime [28]. The results attained attacking this implementation were similar to those attained attacking an implementation using Montgomery multiplication. Only one set of results are given since this difference in implementation had no significant impact on the results.

The power consumption acquisitions were taken with the microprocessor clocked at 7.37 MHz. The entire scalar multiplication generation was recorded at 125 MS/s and then filtered using a low-pass filter with a corner frequency set to 7.37 MHz. Experimentation while conducting the attacks described below determined that this improved the success rate of any subsequent side-channel analysis.

SASEBO A VHDL implementation on a SASEBO-G board, using a Montgomery multiplier based on the CIOS algorithm [29] and containing a single 32-bit single-precision multiplier, and using homogenous projective coordinates with the point addition algorithm proposed by Cohen et al. [30], and doubling algorithm proposed by Bernstein and Lange [31]. The underlying architecture is similar to that as presented in [32], with a single ALU containing the modular Montgomery multiplication unit and an addition/subtraction unit. The power consumption acquisitions were taken with the FPGA clocked at 24 MHz. The entire scalar multiplication algorithm was recorded at 250 MS/s and filtered with both a high pass filter to remove a DC drift in the traces, and a low pass filter to remove high frequency noise.

5.2 Implementing the Attacks

In each case described below we assume that an adversary has divided up the acquired power consumption traces into smaller traces corresponding to individual operations. That is, if we consider a trace S we can describe the trace as a series of n subtraces

$$S = \{O_1, O_2, O_3, \ldots, O_{n-1}, O_n\},$$

where the exponentiation algorithm consists of n group operations. That is, O_i will be the power consumption during the i-th operation in trace S. An operation is defined as either an addition, a doubling or a field operation and it should be clear from the context to which we refer. A description of how one achieves this, and a description of an attack using these subtraces as a set of traces is given by Clavier et al. [33].

To give an example of how one can extract these subtraces , Figure 1 shows a portion of a power consumption trace where nine field multiplications are visible as a repeating pattern.
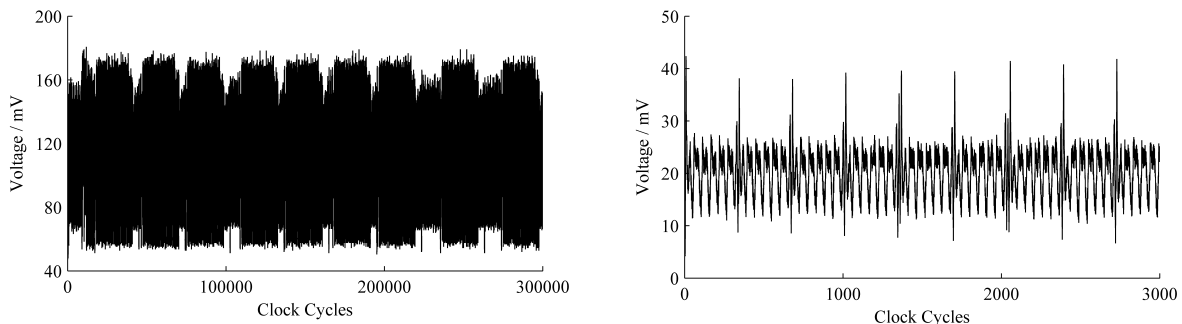

Fig. 1. Example power consumption trace showing the individual multiplication operations in a group operation for the ARM7 (upper) and SASEBO (lower) implementations.

The attacks described in the literature typically require that an adversary takes a certain number of traces and then generate a mean trace by computing the mean power consumption at each point in the set of traces [7–11]. This strategy gives a trace that is representative of the power consumption without high frequency noise, since noise will be removed as one computes the mean.

In our implementations we took a different strategy. We took all the operations $\{O_1, O_2, O_3, \ldots, O_{n-1}, O_n\}$ and computed a mean subtrace $\bar{O}$. This mean trace was then subtracte point-by-point from each element in $\{O_1, O_2, O_3, \ldots, O_{n-1}, O_n\}$ to give $\{O'_1, O'_2, O'_3, \ldots, O'_{n-1}, O'_n\}$. The resulting set of subtraces where the effect of the instructions being executed has been removed and one is left with the noise and variation caused by the manipulated data.

A description of an implementation of the attacks defined in Section 4 to our target implementations is given below.

Highly-Regular Right-to-Left Scalar Multiplication Given Lemma 2, we wish to compare the power consumption during the second addition of a given round with the first addition of the next round. To reduce the amount of computation required to conduct the attack, all of the acquisitions used were also compressed by only keeping the maximum point per clock cycle [34]. If we consider the m-point addition traces a_i that make up the power consumption trace A taken during the computation of a n-bit scalar multiplication as defined in Algorithm 2. Then

$$A = \{a_{1,1} \ldots a_{m,1}, a_{1,2} \ldots a_{m,2}, \ldots, a_{1,2\,(n-1)} \ldots a_{m,2\,(n-1)}\}$$

using the notation above $O'_i = a_{1,1} \ldots a_{m,1}$ for all $1 \leq i < n$, i.e. each O'_i is a trace consisting of m points.

Then one can compute a trace where we assume that a collision occurs at every possible location identified by Lemma 2. Giving

$$\begin{aligned}
C &= c_1 \ldots c_{m-1} \\
&= \rho((a_{1,2} \ldots a_{m,2}, a_{1,3} \ldots a_{m,3}), \ldots \\
&\qquad \ldots, (a_{1,2\,n-4} \ldots a_{m,2\,n-4}, a_{1,2\,n-3} \ldots a_{m,2\,n-3}))
\end{aligned}$$

where ρ computes Pearson's correlation coefficient. The correlation trace C consisting of m points will show a significant correlation where a collision could be detected. That is enough of (a_i, a_{i+1}) will collide, without knowing which, that it will be visible in a correlation trace. An example trace is shown in Figure 2.
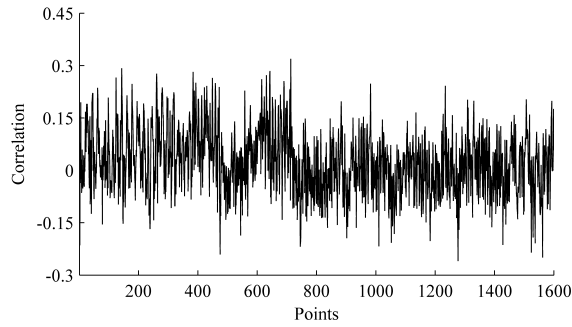

Fig. 2. An example correlation trace showing the correlation across one operation asuming that a collision occurs in every loop of the exponentiation. The example is generated from one power consumption trace taken while a SASEBO board was computing an exponentiation using Joye's add-only exponentiation algortihm.

A threshold can then be chosen where all points that have a magnitude above this threshold have their index noted, where the choice of threshold depends on the platform one is attacking. Extracting these points one can form a trace from q-point subtraces by concatenating the points at the selected indices from each a_i. For example, we can denote

$$A' = \{a'_{1,1} \ldots a'_{q,1}, a'_{1,2} \ldots a'_{q,2}, \ldots, a'_{1,2\,(n-1)} \ldots a'_{q,2\,(n-1)}\}$$

This trace is then used to determine whether a collision occurs between individual operations. For example, using Pearson's correlation coefficient to detect a collision one computes a trace

$$\begin{aligned}
D &= d_1 \ldots d_{n-1} \\
&= \{\rho(a'_{1,2} \ldots a'_{q,2}, a'_{1,3} \ldots a'_{q,3}), \ldots \\
&\qquad \ldots, \rho(a'_{1,2\,n-4} \ldots a'_{q,2\,n-4}, a'_{1,2\,n-3} \ldots a'_{q,2\,n-3})\}
\end{aligned}$$

The correlation coefficients in D can then be converted to hypothetical values for bits of the exponent by observing whether they fall above or below the mean correlation across all d_i for $i \in \{1, \ldots, n-1\}$. Here ρ can be replaced by

another distinguisher such as the Euclidean distance. Hence, the value of d_i will give a hypothesis for the i-th bit of the exponent for $i \in \{1, \ldots, n-1\}$.

An attack, as described above, was applied to a set of 1000 traces taken while an ARM7 microprocessor was computing an exponentiation and 8000 traces taken while an SASEBO board was computing an exponentiation using Joye's add-only exponentiation algorithm. The only difference in the way that the attack was applied was in the way that the threshold used to select points from C is chosen. In the application to the SASEBO board the threshold was set to the mean correlation of $c_1 \ldots c_m$. This approach could not be taken with the implementation of the attack applied to the ARM7 implementation because of the number of points in each trace. An arbitrarily chosen threshold of 0.55 was chosen to select the majority of peaks in the observed traces.

On both implementations an attack can be defined where an adversary can deduce the exponent with a high probability. Indeed, the majority of the values for the time complexity fall well below our defined threshold for a trivial attack. However, the correlation coefficient is less effective than the Euclidean distance. The results are summarized in Table 1.

Table 1. The probability of successfully attacking an implementation of the add-only exponentiation algorithm on an ARM7 microprocessor given 1000 obsevations and a SASEBO board given 8000 observations.

Platform	Algorithm	Matching Method	E (#Errors)	σ	Pr(trivial attack)	Pr(practical attack)
ARM7	Add-only	Euclidean distance	5.78	4.30	0.926	0.991
		Correlation Coefficient	5.52	4.96	0.935	0.993
	Dummy	Euclidean distance	6.10	7.10	0.894	0.968
		Correlation Coefficient	8.40	8.66	0.820	0.920
	Montgomery	Euclidean distance	14.7	4.35	0.306	0.926
		Correlation Coefficient	21.7	4.74	0.0110	0.409
SASEBO	Add-only	Euclidean distance	7.69	2.68	0.955	1
		Correlation Coefficient	24.8	4.93	0.00338	0.190
	Dummy	Euclidean distance	37.7	5.88	0.00188	0.00225
		Correlation Coefficient	24.4	4.88	0.00525	0.207

Scalar Multiplication with Dummy Operations The attack described in the previous section only needs to consider collisions between point additions. That is, the power consumption traces corresponding to the operations of interest can be superposed. However, if an exponentiation using Coron's dummy-and-add-always algorithm, see Section 4, one would typically use a different algorithms to compute an addition and a doubling operation (as described in Section 2.2).

We consider a trace T made up of subtraces corresponding to doubling operations D_i and additions A_i as follows:

$$T = \{D_1, A_1, D_2, A_2 \ldots, D_{n-1}, A_{n-1}\}$$

where a doubling operation D_i and addition A_i consist of g and h field multiplications respectively. We assume that the power consumption taken during one field multiplication consists of m points. One can then use all the available operations to compare all the field multiplications in a doubling operation with those in the following addition in a manner similar to that described in Section 4.2, i.e. assuming that a collision occurs in every round given Lemma 1. This gives produces a $g \times h$ matrix where each entry is a m-point trace. That is:

$$C = \begin{pmatrix} c_{1,1} & c_{1,2} & \ldots & c_{1,g} \\ c_{2,1} & c_{2,2} & \ldots & c_{2,g} \\ \vdots & \vdots & \ddots & \vdots \\ c_{h,1} & c_{h,2} & \ldots & c_{h,g} \end{pmatrix}$$

where each $c_{i,j}$, for $1 \leq i \leq h$ and $1 \leq j \leq g$, is a m-point trace. For convenience only the field multiplications are considered and the other field operations are discarded.

Two different approaches of using this matrix where explored. In applying the collision attack to traces taken while an ARM7 microprocessor was computing an exponentiation using Coron's dummy-and-add-always we noted that a significant number of operations that should have no linear relation produced significant correlation coefficients. We therefore computed a second matrix where no collisions where possible, i.e. by randomly selecting operations to compare. This matrix was then subtracted from the first matrix point-by-point to remove spurious peaks in the correlation matrix.

As for the attack described in Section 4.2, a threshold set to 0.55 to determine which points from which operations were selected. If there are q indices that are selected then we form one trace from the points indicated on one dimension by extracting the relevant points from each subtrace in T

$$A' = \{a'_{1,1} \ldots a'_{q,1}, a'_{1,2} \ldots a'_{q,2}, \ldots, a'_{1,2\,(n-1)} \ldots a'_{q,2\,(n-1)}\},$$

and another trace from points indicated on the other dimension

$$B' = \{b'_{1,1} \ldots b'_{q,1}, b'_{1,2} \ldots b'_{q,2}, \ldots, b'_{1,2\,(n-1)} \ldots b'_{q,2\,(n-1)}\}.$$

That is, computing $\rho(a'_{1,1} \ldots a'_{q,1}, b'_{1,1} \ldots b'_{q,1})$ will compute the correlation coefficient between all the combinations of points indicated by C.

As previously, one can use these traces points can be used to generate a trace of correlation coefficients,

$$
\begin{aligned}
D &= d_1 \ldots d_{n-1} \\
&= \{\rho(a'_{1,2} \ldots a'_{q,2}, b'_{1,3} \ldots b'_{q,3}), \ldots \\
&\qquad \ldots, \rho(a'_{1,2\,n-4} \ldots a'_{q,2\,n-4}, b'_{1,2\,n-3} \ldots b'_{q,2\,n-3})\}
\end{aligned}
$$

The mean of these coefficients was used to determine whether a given bit of the exponent is set to 1 or 0, and hence generate a hypothesis for the exponent. The attack was repeated on 1000 power consumption traces and the results are summarized in Table 1.

In applying the collision attack to traces taken while an SASEBO board was computing an exponentiation using Coron's dummy-and-add-always, we generated traces by selecting traces from the matrix C given our knowledge of the algorithms used to compute additions and doubling operations. That is, concatenating the subtraces of interest from C that should allow a collision to be determined. Given the algorithms used, six field multiplications could be compared to allow collisions to be detected and the mean value of the concatenated subtraces was used as a threshold to extract the points that would be used to generate A' and B'. These traces were then used to generate a series of correlation coefficients D to then form hypotheses on the bits of the exponent as above. The attack we repeated on 8000 power consumption traces and the results are summarized in Table 1.

The ARM7 implementation can be attacked readily as the median case is well below our defined threshold for a trivial attack. Contrary to the previous attack, the correlation coefficient provides a better distinguisher than the Euclidean distance when applied to our SASEBO implementation. A attack using the correlation coefficient will require approximately 280 traces, where each exponent may be distinct, to produce a result that can be analysed with a trivial complexity. That is, if one expended a maximum effort of 2^{40} operations per trace a valid attack will have an overall time complexity of 2^{48}.

Montgomery Ladder As previously, we consider a trace T made up of subtraces corresponding to doubling operations D_i and additions A_i as follows:

$$T = \{A_1, D_1, A_2, D_2, \ldots, A_{n-1}, D_{n-1}\}$$

We wish to compare the output of a doubling operations with the input of the subsequent addition and doubling operations. That is, we wish to detect collision shown to be present by Lemma 3 and Lemma 4 to determine if two consecutive bits of the exponent are the same.

Then to evaluate whether a collision occurs, as indicated by Lemma 4, we wish to compare the power consumption during an addition with the power consumption during a doubling operation. We shall assume that a power consumption corresponding to a doubling operation and an addition consist of r and s points respectively. We determine a matrix of correlation coefficients C

$$C = \begin{pmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,r} \\ c_{2,1} & c_{2,2} & \dots & c_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ c_{s,1} & c_{s,2} & \dots & c_{s,r} \end{pmatrix}$$

where $c_{i,j}$ is the correlation of the i-th point from the doubling operations with the j-th point from the addition operation across all the pairs of operations indicated by Lemma 4.

As with the attack described in Section 5.2, the correlation coefficients generated in the matrix contained many spurious, yet seemingly significant, correlation coefficients. A second matrix was then generated where no collisions were possible and subtracted from the matrix to remove the spurious correlation coefficients. From this differential matrix we chose all the indices where the correlation coefficient was superior to an arbitrarily chosen threshold of 0.15. As previously, if there are q indices that are selected then we form one trace from the points indicated on one dimension by extracting the relevant points from each subtrace in T

$$A' = \{a'_{1,1} \dots a'_{q,1}, a'_{1,2} \dots a'_{q,2}, \dots, a'_{1,2\,(n-1)} \dots a'_{q,2\,(n-1)}\},$$

and another trace from points indicated on the other dimension

$$B' = \{b'_{1,1} \dots b'_{q,1}, b'_{1,2} \dots b'_{q,2}, \dots, b'_{1,2\,(n-1)} \dots b'_{q,2\,(n-1)}\}.$$

That is, computing $\rho(a'_{1,1} \dots a'_{q,1}, b'_{1,1} \dots b'_{q,1})$ will compute the correlation coefficient between all the combinations of points indicated by C.

One can then use these reduced traces to generated a series of correlation coefficients

$$\begin{aligned} D &= d_1 \dots d_{n-1} \\ &= \{\rho(a'_{1,2} \dots a'_{q,2}, b'_{1,3} \dots b'_{q,3}), \dots \\ &\qquad \dots, \rho(a'_{1,2\,n-4} \dots a'_{q,2\,n-4}, b'_{1,2\,n-3} \dots b'_{q,2\,n-3})\} \end{aligned}$$

as described in Section 4.2 and the mean of these coefficients used to generate hypotheses for the exponent.

This can be repeated to compare a doubling operation with the following doubling operation to evaluate the collisions indicated by Lemma 3. This analysis produced two traces of correlation coefficients that can be used to produce hypotheses for each bit of the exponent. Where the hypotheses differ we selected the hypothesis with the largest difference from the mean correlation coefficient under the assumption that this will be the strongest distinguisher.

The ARM7 implementation can be defined readily as the median case is well below our defined threshold for a trivial attack when the correlation coefficient is used. The results of the attack applied to 1000 traces are summarized in Table 1.

No practical attack was derived for on our SASEBO implementation. We assume that the leakage required to conduct this attack is not present.

6 Further Comments

In the above we show that an adversary can determine if two variables are used at arbitrary points in an exponentiation algorithm. To resist such attacks an exponentiation algorithm would need to to be more than highly regular, as defined in Definition 2. If variables are reused then, to resist our attacks, the order that they are used should not reveal any information to an adversary. That is, an adversary cannot find meaningful collisions based on comparing the output of one operation with the input or output of another operation. Hence, we define the following property:

Definition 3. *An exponentiation is defined as to be totally regular if it consists of operations* $g_1, g_2, \ldots, g_\kappa$ *that are composed* $g_\kappa \circ \cdots \circ g_2 \circ g_1$, *where* $\kappa \in \mathbb{Z}$ *is fixed for an exponent of a given bit length and each* g_i *is of the form* $z_i = g_i(x_i, y_i)$ *for* $i \in \{1, \ldots, \kappa\}$. *The function* $h = g_i$ *for* $i \in \{1, \ldots, \kappa\}$ *and some function* h *and the address is fixed for each* x_i, y_i, z_i *for* $i \in \{1, \ldots, \kappa\}$.

However, such an exponentiation algorithm is not possible.

Lemma 5. *A totally regular addition chain-based exponentiation algorithm that can compute* n^λ *for any* $n, \lambda \in \mathbb{Z}_{>0}$, *assuming* λ *has a fixed bit length, does not exist.*

Proof. Let **A** be an addition chain that can be used to compute n^λ from n for some arbitrary $n, \lambda \in \mathbb{Z}_{>0}$. That is, there will exist some function

$$f : \mathbb{Z}_{>0} \longrightarrow \mathbb{Z}_{>0} : n \longmapsto n^\lambda$$

that uses addition chain **A** to compute $f(n)$ for some arbitrary λ of a given bit length. If f is totally regular this would imply that **A** would remain unchanged for all possible exponents that have the same bit length as λ. However, by Definition 1 the last element of **A** is n^λ, and we note that λ cannot be changed by modifying a_0, a_1. Hence

$$f \iff \mathbf{A}$$

and a totally regular addition chain-based exponentiation algorithm is not possible.

This means that one needs to ensure that such a side-channel produces an attack whose time complexity is too large to be practical. The reader is referred to Fan and Verbauwhede [35] for a thorough treatment of this topic.

7 Conclusion

We describe attacks based on collisions of variables manipulated in group operations to determine an unknown exponent and demonstrate that they can be applied to a single trace without requiring any profiling. This represents a significant advantage over attacks described in the literature that typically require numerous traces to be acquired [7–11]. Moreover, an adversary does not require any knowledge of the input to the exponentiation algorithm. However, an adversary may have to independently apply the attack to more than one trace to succeed or accept that an attack will only succeed with a certain probability.

We chose to apply attacks to Joye's add-only [14] and Coron's double-and-add-always [18] exponentiation algorithms to demonstrate their effectiveness against the two main methods of preventing an adversary from deriving an exponent by inspecting a power consumption trace. That of reducing an exponentiation algorithm to indistinguishable atomic operations or using dummy operations. We extended the attacks described in the literature by defining, and implementing, an attack on the Montgomery ladder [12, 13] that would allow an adversary to analyze versions of the Montgomery ladder that were previously thought to be secure. Again, only one trace is required to derive the exponent with a high probability.

In Section 5 we present some results of attacking an implementation of a scalar multiplication using the correlation coefficient and the Euclidean distance as a distinguisher. In the case of the Joye's add-only [14] exponentiation algorithm, using the Chebychev distance returned the exponent with no errors for all 8000 acquired traces the hardware

implementation on the SASEBO board. As the Chebychev distance is simply the maximum absolute distance between the two traces this is somewhat counterintuitive. Examining the bit transitions during the add-only algorithm using the bundled simulator with Xilinx ISE, it was observed that when the key-bit was one, there was a transition on a data bus from the prime (consisting mostly of bits set to one) to zero which caused a disproportionate effect on the resultant collision attack. This essentially reduces the attack to an SPA attack on the beginning of the point addition operation, where if the power consumption is relatively larger then the key bit is 1. Similar results could not be achieved for the other algorithms described in this paper. While implementation attacks are by their nature implementation specific, this does highlight that unexpected leakage can lead to devastating attacks. Moreover, in evaluating an implementations resistance to the attacks described in this paper one needs to investigate further than a straightforward implementation of our attacks.

We also prove that the attacks described in this paper are applicable to all exponentiation algorithms based on addition chains. This means that a side channel resistant implementation of a group exponentiation will require countermeasures that introduce enough noise that an attack is not practical.

Acknowledgments

References

1. Rivest, R., Shamir, A., Adleman, L.M.: Method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2) (1978) 120–126
2. National Institute of Standards and Technology (NIST): Recommended elliptic curves for federal government use. In the appendix of FIPS 186-3, available from `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf` (June 2009)
3. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In Wiener, M.J., ed.: CRYPTO '99. Volume 1666 of LNCS., Springer (1999) 388–397
4. Joye, M., Tunstall, M.: Exponent recoding and regular exponentiation algorithms. In Preneel, B., ed.: AFRICACRYPT 2009. Volume 5580 of LNCS., Springer (2009) 334–349
5. Schramm, K., Wollinger, T., Paar, C.: New class of collision attacks and its application to DES. In Johansson, T., ed.: FSE 2003. Volume 2887 of LNCS., Springer (2003) 206–222
6. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Improved collision-correlation power analysis on first order protected AES. In Preneel, B., Takagi, T., eds.: CHES 2011. Volume 6917 of LNCS., Springer (2011) 49–62
7. Okeya, K., Sakura, K.: A second-order DPA attack breaks a window-method based countermeasure against side channel attacks. In Chan, A.H., Gligor, V., eds.: ISC 202. Volume 2433 of LNCS., Springer (2002) 389–401
8. Amiel, F., Feix, B.: On the BRIP algorithms security for RSA. In Onieva, J.A., ed.: WISTP 2008. Volume 5019 of LNCS., Springer (2008) 136–149
9. 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
10. Mamiya, H., Miyaji, A., Morimoto, H.: Efficient countermeasures against RPA, DPA, and SPA. In Joye, M., Quisquater, J.J., eds.: CHES 2004. Volume 3156 of LNCS., Springer (2004) 343–356
11. 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
12. Montgomery, P.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**(177) (1987) 243–264
13. 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

14. 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
15. Wireless Application Protocol (WAP) Forum: Wireless transport layer security (WTLS) specification. Available from `http://www.wapforum.org`
16. X9.62, A.: Public key cryptography for the financial services industry, the elliptic curve digital signature algorithm (ECDSA) (1999)
17. 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
18. 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
19. Yen, S.M., Ko, L.C., Moon, S.J., Ha, J.: Relative doubling attack against Montgomery ladder. In Won, D., Kim, S., eds.: ICISC 2005. Volume 3935 of LNCS., Springer (2005) 117–128
20. Homma, N., Miyamoto, A., Aoki, T., Satoh, A., Shamir, A.: Collision-based power analysis of modular exponentiation using chosen-message pairs. In Oswald, E., Rohatgi, P., eds.: CHES 2008. Volume 5154 of LNCS., Springer (2008) 15–29
21. Chevallier-Mames, B., Ciet, M., Joye, M.: Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. IEEE Transactions on Computers **53**(6) (2004) 760–768
22. 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
23. Stinson, D.: Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem. Mathematics of Computation **71**(237) (2002) 379–391
24. 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
25. Izu, T., Takagi, T.: Fast elliptic curve multiplications resistant against side channel attacks. IEICE Transactions **88-A**(1) (2005) 161–171
26. Großschädl, J., Oswald, E., Page, D., Tunstall, M.: Side channel analysis of cryptographic software via early-terminating multiplications. In Lee, D., Hong, S., eds.: ICISC 2009. Number 5984 in LNCS, Springer (2009) 176–192
27. Montgomery, P.: Modular multiplication without trial division. Mathematics of Computation **44** (1985) 519–521
28. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press (1997)
29. Koc, C.K., Acar, T., Kaliski Jr, B.S.: Analyzing and comparing montgomery multiplication algorithms. Micro, IEEE **16**(3) (1996) 26–33
30. Cohen, H., Miyaji, A., Ono, T.: Efficient elliptic curve exponentiation using mixed coordinates. In Ohta, K., Pei, D., eds.: ASIACRYPT '98. Volume 1514 of LNCS., Springer (1998) 51–65
31. Bernstein, D.J., Lange, T.: Faster addition and doubling on elliptic curves. In Kurosawa, K., ed.: ASIACRYPT 2007. Volume 4833 of LNCS., Springer (2007) 29–50
32. Keller, M., Byrne, A., Marnane, W.P.: Elliptic curve cryptography on fpga for low-power applications. TRETS **2**(1) (2009)
33. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal correlation analysis on exponentiation. In Soriano, M., Qing, S., López, J., eds.: ICICS 2010. Volume 6476 of LNCS., Springer (2010) 46–61
34. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks — Revealing the Secrets of Smart Cards. Springer (2007)
35. Fan, J., Verbauwhede, I.: An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In Naccache, D., ed.: Cryptography and Security. Volume 6805 of LNCS., Springer (2012) 265–282

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 4. *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 [23]. Stinson's algorithm requires the existence of a means of splitting a string of bits into two sets of equal Hamming weight.

Lemma 6. *We consider an integer of bit length m, as a string of bits of length $m \in 2\mathbb{Z}$ and Hamming weight $0 < t < m$. 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 m with Hamming weight $t \in 2\mathbb{Z}$. Let each Y_i for $i \in \{1, \ldots, m/2\}$ represent one of the $m/2$ sets of contiguous bits starting from the i-th bit of the string. Let H be a function that returns the Hamming weight, then $H(Y_1) = t - H(Y_{m/2})$. Given that $H(Y_i) - H(Y_{i+1})$ will be in $\{-1, 0, 1\}$ there will be some set of contiguous bits with Hamming weight $m/2$. 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 $\lfloor m/2 \rfloor$ and the other of $\lceil m/2 \rceil$.

This is sufficient for our requirements. We refer the reader to Stinson for versions of this proof where m is odd [23].

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

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

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 which 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 m-bit exponent one would be required to compute $\binom{m}{t/2}$ Giant-Steps and $\binom{m}{t/2}$ 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 $\binom{m}{\lfloor t/2 \rfloor}$ Giant-Steps and $\binom{m}{\lfloor t/2 \rfloor + 1}$ 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 [23], and has time complexity of $\mathcal{O}\left(m \binom{m/2}{t/2} \right)$. 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 $\mathcal{O}\left(m \sum_{n=0}^{t} \binom{m/2}{n/2} \right)$. However, by Lemma 6 we can ignore the cases where n is odd. Since the required baby and giant steps will be computed for the cases $n-1$ and $n+1$. The resulting time complexity is therefore $\mathcal{O}\left(m \sum_{n=0}^{\lceil t/2 \rceil} \binom{m/2}{n} \right)$ when t is unknown.