# 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 high probability. Moreover, our attacks do not require any knowledge of the input to the exponentiation algorithm. These attacks would therefore be applicable to algorithms such as EC-DSA, where an exponent is ephemeral, or to implementations where an exponent is blinded. Moreover, we define attacks against exponentiation algorithms that are considered to be resistant to collision attacks and prove that collision attacks are applicable to all addition chain-based exponentiation algorithms. Hence, we demonstrate 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 an 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 use, for example, 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. Walter [7] observed that one could potentially derive the bits of an exponent by comparing two parts of a consumption trace corresponding to two operations in a group exponentiation and described some simulations. 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, and Kim et al. [9] also describe how one could apply such an attack to 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 [8–11, 14]. 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 implementations of the attacks described in the literature is that the adversary needs numerous traces with the same input and same exponent to derive an unknown exponent using collisions [8–11, 14]. That is, an adversary will take numerous acquisitions and reduce noise in the acquired power consumption traces by computing a mean trace.

### 1.1 Contribution

In this paper, we first show the practical analysis using a single trace based on previous work detailing attacks on the reuse of values in operations. We extend this attack model to some countemeasures previously thought to be secure against attacks described in the literature.

In existing attacks, an adversary typically attempts to determine whether the input to two group operations is the same [8–11, 14, 15]. However, 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, an adversary can apply our collision attacks to some more algorithms, secure against all existing collision attacks.

The common requirement for the the attacks described in the literature is that an adversary needs numerous traces with the same exponent to derive an unknown exponent using collisions [8–11, 14]. 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 introduce an extended attack model which only requires a single trace to succeed.

Some collision attacks using chosen-plaintext [16–19] cannot be exploited to attack ECDSA (Elliptic Curve Digital Signature Algorithm) [20] using typical DPA countermeasures. However, in the case of our attacks, an adversary does not need to know the plaintext. In addition, because our attack uses only one trace, the secret key of ECDSA and the countermeasures can be easily exposed. In the case of ECDSA, our attack method finds the ephemeral key $k$ from the power consumption trace while generating the signature ($r$, $s$). As is widely known, this means that an adversary can find the private key $d$ as $r^{-1}(k\,s - h(m))$ mod $q$ where $h(m)$ and $q$ denote the hash value of the message $m$ and the order of a generator of the elliptic curve, respectively (see [20] for a thorough description of ECDSA).

Given that our practical work justifies our extended adversarial model for collision attacks, we define attacks that could be applied to exponentiation algorithms that are considered to be resistant to collision attacks. Specifically, the exponentiation algorithm MIST proposed by Walter [21] and two exponentiation algorithms recently proposed by Belenky et al. [22]. 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.

### 1.2 Organization

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 define attacks against exponentiation algorithms that are considered to be resistant to collision attacks. In Section 7 we prove that the results can be applied to any exponentiation algorithm. We conclude in Section 8.

## 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 $s^{\chi}$ for some arbitrary $s \in \mathbb{G}$ and $\chi \in \mathbb{Z}_{>0}$. Each member of the sequence is the product of two previous members of the sequence. A more formal definition now given.

**Definition 1.** *An addition chain of length $\tau$ in group $\mathbb{G}$ for a group element $s \in \mathbb{G}$ is a sequence of group elements $(\omega_0, \omega_1, \omega_2, \ldots, \omega_\tau)$ such that $\omega_0 = 1_{\mathbb{G}}$, $\omega_1 = s$ and $\omega_k = \omega_i \otimes \omega_j$, $0 \leq i \leq j < k \leq \tau$ and $\otimes$ is the group operation. The values of $i$ and $j$ are chosen such that $\omega_k$ is a chosen power of $\omega_1$.*

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, 23]. We define highly regular as the following:

**Definition 2.** *A group $\mathbb{G}$ exponentiation algorithm is defined as highly regular if it consists of an operation on $g \in \mathbb{G}$ that is composed of $g \otimes \cdots \otimes g \otimes g$, where $\otimes$ is the group operation, $g$ occurs $\kappa \in \mathbb{Z}_{>0}$ times and $\kappa$ 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

$$\mathcal{E} : y^2 + \alpha_1\, x\, y + \alpha_3\, y = x^3 + \alpha_2\, x^2 + \alpha_4\, x + \alpha_6$$

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

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

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

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

$$x_3 = \lambda^2 + \alpha_1\, \lambda - \alpha_2 - x_1 - x_2,\; y_3 = (x_1 - x_3)\lambda - y_1 - \alpha_1\, x_3 - \alpha_3 \tag{1}$$

$$\text{with } \lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{if } \boldsymbol{P} \neq \boldsymbol{Q}\,, \\ \dfrac{3\, {x_1}^2 + 2\, \alpha_2\, x_1 + \alpha_4 - \alpha_1\, y_1}{2\, y_1 + \alpha_1\, x_1 + \alpha_3} & \text{if } \boldsymbol{P} = \boldsymbol{Q}\,. \end{cases}$$

Provided that the characteristic of field $\mathbb{F}_q$ is not 2 or 3, we can take $\alpha_1 = \alpha_2 = \alpha_3 = 0$. In the following we will also assume that $q = p$ is prime. We define the short Weierstraß form over prime field $\mathbb{F}_p$ by the equation

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

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

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 [24] and ANSI X9.62 [20].

## 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 [16], 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 [25] (see Algorithm 2), 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 [25] 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. [17] extended the attack proposed by Fouque and Valette [16] 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. [18]. 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 $M_1$ and $M_2$ that satisfy the equation $M_1^{\pi} = M_2^{\psi} \mod N$, where $\pi$ and $\psi$ are given constants. The value of $\pi$ is chosen to provide information on a certain bit, while $\psi \leq \pi$ 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. Some further extensions have recently been proposed by Yen et al. [19] who provide some simulated results of their atacks.

The above attacks require that an adversary is able to choose the input to a group exponentiation algorithm. However, one would typically assume that an input point is blinded such that the input to an exponentiation algorithm is not known to an adversary. Walter [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. However, Walter only presented results based on simulations. Amiel and Feix [8], and Kim et al. [9] describe some attacks implementing a collision attack strategy proposed by Okeya and Sakura [14]. They describe a collision attack 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] and how one could prevent these collision attacks by modifying the algorithm. 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. Moreover, all of the attacks in the literature require that a value used in one operation is reused in another operation. We extend this model in later sections leading to new attacks.

## 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 [25], 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. [26].

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 generalize the methods in Sections 6 and 7.

### 4.1 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 [23] as shown in Algorithm 1. 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 1 Joye's Add-Only Scalar Multiplication** *[23]*

***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: $\boldsymbol{R}_0 \leftarrow \boldsymbol{\mathcal{O}}$
2: $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$
3: $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$
4: ***for*** $i \leftarrow 0$ **to** $n-1$ ***do***
5: $\quad b \leftarrow 1 - k_i$
6: $\quad \boldsymbol{R}_b \leftarrow \boldsymbol{R}_b + \boldsymbol{R}_2$
7: $\quad \boldsymbol{R}_2 \leftarrow \boldsymbol{R}_0 + \boldsymbol{R}_1$
8: ***end for***
9: ***return*** $\boldsymbol{R}_0$

**Lemma 1.** *In one loop of Algorithm 1,* $\boldsymbol{R}_0$ *remains the same when* $k_{i+1} = 0$ *and* $\boldsymbol{R}_1$ *remains the same when* $k_{i+1} = 1$.

*Proof.* We assume that an adversary knows the $\ell$ 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 $\boldsymbol{R}_0$, $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ at the end of the $\ell$-th loop will be

$$\boldsymbol{R}_0 = k'\,\boldsymbol{P}\,, \quad \boldsymbol{R}_1 = (2^\ell - k')\,\boldsymbol{P} \quad \text{and} \quad \boldsymbol{R}_2 = 2^\ell\,\boldsymbol{P}\,.$$

In the $(\ell+1)$-th loop we consider the change in the registers $\boldsymbol{R}_0$ and $\boldsymbol{R}_1$:

If $k_{\ell+1} = 0$ then, from the end of the $\ell$-th round where $\boldsymbol{R}_0 = k'\,\boldsymbol{P}$ and $\boldsymbol{R}_1 = (2^\ell - k')\,\boldsymbol{P}$ , the updated values change to

$$\boldsymbol{R}_0 = k'\,\boldsymbol{P} \quad \text{and} \quad \boldsymbol{R}_1 = (2^{\ell+1} - k')\,\boldsymbol{P}$$

in the $(\ell+1)$-th round. Conversely where $k_{\ell+1} = 1$ then, the values at the end of the $\ell$-th round instead change to

$$\boldsymbol{R}_0 = (2^{\ell+1} + k')\,\boldsymbol{P} \quad \text{and} \quad \boldsymbol{R}_1 = (2^\ell - k')\,\boldsymbol{P}$$

in the $(\ell+1)$-th round. We can see that $\boldsymbol{R}_0$ remains the same when $k_{\ell+1} = 0$, while $\boldsymbol{R}_1$ remains the same when $k_{\ell+1} = 1$. □

### 4.2 Scalar Multiplication with Dummy Operations

The first countermeasure defined to prevent the identification of group operations was proposed by Coron [25]. In the additive group formed over an elliptic curve the resulting algorithm is shown in Algorithm 2. 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 2 Coron's Double-and-Add-Always Algorithm** *[25]*

***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: $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$
2: $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$
3: ***for*** $i \leftarrow n-2$ **down to** $0$ ***do***
4: $\quad \boldsymbol{R}_0 \leftarrow 2\,\boldsymbol{R}_0$
5: $\quad b \leftarrow 1 - k_i$
6: $\quad \boldsymbol{R}_b \leftarrow \boldsymbol{R}_0 + \boldsymbol{P}$
7: ***end for***
8: ***return*** $\boldsymbol{R}_0$

**Lemma 2.** *In one loop of Algorithm 2, if the bit treated in the loop is equal to zero the input to the dummy operation* $\boldsymbol{R}_0$ *will be the same as the input to the subsequent doubling operation.*

*Proof.* Trivially, we assume that an adversary knows the $\ell$ most significant bits of the scalar. Then in the $(\ell+1)$-th loop $\boldsymbol{R}_0$ 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 2, 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. We demonstrate how to do this in Section 5.2.

### 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: $\boldsymbol{R}_0 \leftarrow \boldsymbol{\mathcal{O}}$
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 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 $\ell$ most significant bits of the scalar, and define $k' = (k_{n-1} \ldots k_{n-\ell})_2$. The intermediate results at the end of the $\ell$-th loop will be

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

If $k'$ is even, then the bit $k_{n-\ell}$ is necessarily equal to zero hence $\boldsymbol{R}_0$ was calculated via the doubling operation. Then if the next bit, $k_{n-\ell-1}$, is also equal to zero, the following $(\ell+1)$-th 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 $k'$ is odd, then the bit $k_{n-\ell}$ is necessarily equal to one hence $\boldsymbol{R}_1$ was calculated via the doubling operation. Then if the next bit, $k_{n-\ell-1}$, is also equal to one, the following $(\ell+1)$-th loop will compute the doubling operation

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

That is, for a given value of the bit $k_{n-\ell}$ then the output of the doubling operation in the $\ell$-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 $\ell$ most significant bits of the scalar, and define $k' = (k_{n-1} \dots k_{n-\ell})_2$, and the intermediate results at the end of the $\ell$-th loop will be

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

If $k'$ is odd, then the bit $k_{n-\ell}$ is necessarily equal to one hence $\boldsymbol{R}_0$ was calculated via the addition operation. Then if the next bit is now equal to zero, the following $(\ell+1)$-th 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 $k'$ is even, then the bit $k_{n-\ell}$ is necessarily equal to zero hence $\boldsymbol{R}_1$ was calculated via the addition operation. Then if the next bit, $k_{n-\ell-1}$, is now equal to one, the following $(\ell+1)$-th loop will compute the doubling operation

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

That is, for a given value of the bit $k_{n-\ell}$ then the output of the addition in the $\ell$-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. The discussion is somewhat informal since the aim is to provide a guide to the steps required to conduct collision attacks. 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 implementations of collision attacks described in the literature that typically require numerous traces to succeed [8–11, 14].

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. [27]. It 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 [28], where an $t$-bit error in a $n$-bit hypothesis leads to the exponent in time complexity $\mathcal{O}\left(n \sum_{i=0}^{\lceil t/2 \rceil} \binom{n/2}{i}\right)$ (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 using the attack described in Section 4.3, 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 [29] 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 the three chosen exponentiation algorithms which are 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 feasibility of the attack on both software and hardware platforms. The implementation choices were different in each case since the intention is not to evaluate the platforms but the resistance of the chosen algorithms to collision attacks. 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 [30]. 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 Großschädel et al. [31]. These functions were used as part of an implementation in C.

Of the implementations discussed below, the implementations of Joye's add-only [23] and Coron's double-and-add-always [25] algorithms used Montgomery multiplication [32] as the basic operation. The implementation of the Montgomery ladder [12, 13] used a multi-precision multiplication followed by a rapid reduction algorithm, made possible since the underlying prime field is a generalized Mersenne prime [33]. 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.

**Virtex-II** A VHDL implementation on a SASEBO-G FPGA board [34], using a Montgomery multiplier based on the CIOS algorithm [35] containing a single 32-bit single-precision multiplier. Homogenous projective coordinates are used in conjunction with the point addition algorithm proposed by Cohen et al. [36], and the doubling algorithm proposed by Bernstein and Lange [37]. The underlying architecture is similar to that presented in [38], with a single ALU containing a modular Montgomery multiplication unit and an addition/subtraction unit, controlled via ROM programmed during synthesis. 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 $T$ we can describe the trace as a series of $m$ subtraces

$$T = \{O_1, O_2, O_3, \ldots, O_{m-1}, O_m\},$$

where the exponentiation algorithm consists of $m$ group operations. That is, $O_i$ for $i \in \{1, \ldots, m\}$ will be the power consumption during the $i$-th group operation in trace $T$. A group 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. [39].

To give an example of how one can extract these subtraces, Figure 1 shows a portion of two power consumption traces, each of which show nine field multiplications visible as a repeating pattern.

The attacks described in the literature typically require that an adversary takes a certain number of traces and generate a mean trace by computing the mean power consumption at each point in the set of traces [8–11, 14]. 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.

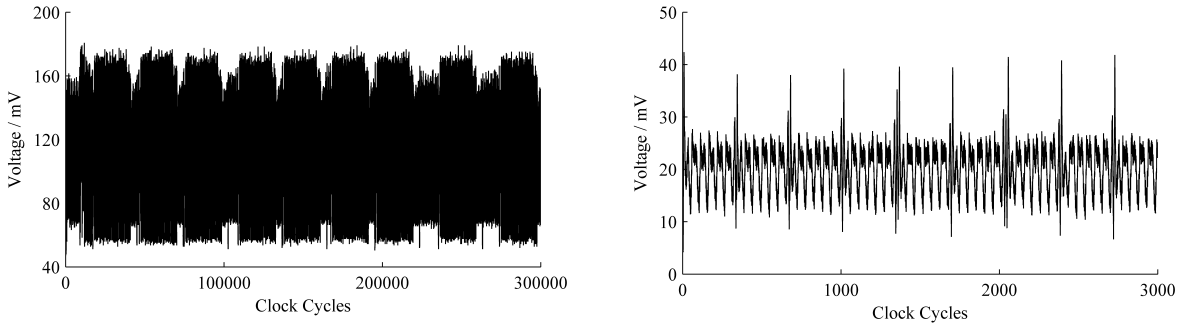


**Fig. 1.** Example power consumption trace showing the individual field multiplication operations in a group operation for the ARM7 (upper) and Virtex-II (lower) implementations.

In our implementations we took a different strategy. We took all the subtraces $\{O_1, O_2, O_3, \ldots, O_{m-1}, O_m\}$ and computed a mean subtrace $\hat{O}$. This mean trace was then subtracted point-by-point from each element in $\{O_1, O_2, O_3, \ldots, O_{m-1}, O_m\}$ to give $\{O'_1, O'_2, O'_3, \ldots, O'_{m-1}, O'_m\}$. The resulting set of subtraces will have the effect of the instructions being executed removed and one is left with 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 now given below.

**Highly-Regular Right-to-Left Scalar Multiplication** Given Lemma 1, 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 [40]. If we consider the $u$-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 1. Then

$$A = \{a_{1,1} \ldots a_{1,u}\, , a_{2,1} \ldots a_{2,u}\, , \ldots\, , a_{m,1} \ldots a_{m,u}\}$$

using the notation above $O'_i = a_{i,1} \ldots a_{i,u} = \bar{a}_i$ for all $1 \leq i < m$, i.e. each $\bar{a}_i$ is a trace consisting of $u$ points. For a $n$-bit scaler, $m$ is equal to $2\,n$ (see Algorithm 1) since there are always two point additions per round.

To locate the points in the group addition trace where collisions would occur, one can compute a trace where we assume that a collision always occurs at every possible location identified by Lemma 1, i.e. we assume that the input of the second addition in round $\ell$ is always equal to the first addition in round $\ell + 1$. This gives

$$C = \rho((\bar{a}_2, \bar{a}_4, \ldots, \bar{a}_{m-2})\, , (\bar{a}_3, \bar{a}_5, \ldots, \bar{a}_{m-1}))$$

where $\rho$ computes Pearson's correlation coefficient for each of the $u$ points in the group addition trace independently. The correlation trace $C$ consisting of $u$ points will show a significant correlation where a collision could be detected if present. That is, enough of $(\bar{a}_i, \bar{a}_{i+1})$ will collide, without knowing which, that it will be visible in a correlation trace to find the relevant points in time. An example trace is shown in Figure 2.

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 and must be decided on a case-by-case basis. Extracting these points, one can form a trace from $v$-point subtraces by concatenating the points at the selected indices from each $\bar{a}_i$. For example, we can denote

$$\begin{aligned} A' &= \{a'_{1,1} \ldots a'_{1,v}, a'_{2,1} \ldots a'_{2,v}, \ldots, a'_{m,1} \ldots a'_{m,v}\} \\ &= \{\bar{a}'_1, \bar{a}'_2, \ldots, \bar{a}'_m\} \end{aligned}$$

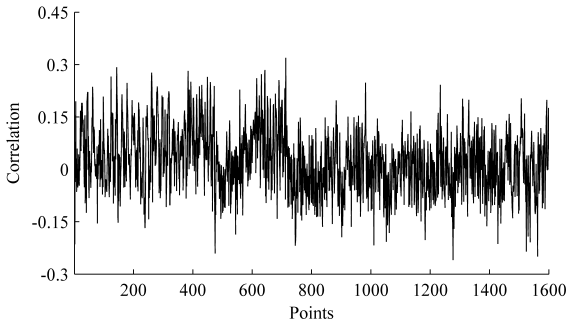


**Fig. 2.** An example correlation trace showing the correlation across one operation assuming that a collision occurs in every loop of the exponentiation. The example is generated from one power consumption trace taken while a Virtex-II FPGA was computing an exponentiation using Joye's add-only exponentiation algorithm.

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

$$\begin{aligned} D &= d_1 \dots d_{n-1} \\ &= \{\rho(\bar{a}'_2, \bar{a}'_3), \rho(\bar{a}'_4, \bar{a}'_5), \dots, \rho(\bar{a}'_{m-2}, \bar{a}'_{m-1})\} \\ &= \{\rho(\bar{a}'_2, \bar{a}'_3), \rho(\bar{a}'_4, \bar{a}'_5), \dots, \rho(\bar{a}'_{2n-2}, \bar{a}'_{2n-1})\} \end{aligned}$$

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 $i \in \{1, \dots, n-1\}$. Here $\rho$ 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 $i \in \{1, \dots, n-1\}$.

An attack, as described above, was applied to each trace individually in a set of $1000$, taken while an ARM7 microprocessor was computing an exponentiation, and $8000$ taken while a 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 $c_1 \dots c_u$. 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.

In both cases we studied, 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.

**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, for an exponentiation implemented 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) which prevents use of the method previously described.

We consider a trace $T$ made up of subtraces corresponding to doubling operations $\delta_i$ and additions $\alpha_i$ as follows:

$$T = \{\delta_1, \alpha_1, \delta_2, \alpha_2 \dots, \delta_{n-1}, \alpha_{n-1}\}$$

where a doubling operation $\delta_i$ and addition $\alpha_i$ for $i \in \{1, \dots, n-1\}$ consist of $f$ and $h$ field multiplications respectively. We assume that the power consumption taken during one field multiplication consists of $u$ points. One can

**Table 1.** The probability of successfully attacking an implementation of the add-only exponentiation algorithm on an ARM7 microprocessor given $1k$ observations and a Virtex-II FPGA given $8k$ observations.

| Platform | Algorithm | Matching Method | E (#Errors) | $\sigma$ | 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 |

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 4.1, i.e. assuming that a collision occurs in every round given Lemma 2. This gives the $f \times h$ matrix

$$C = \begin{pmatrix} \bar{c}_{1,1} & \bar{c}_{1,2} & \dots & \bar{c}_{1,h} \\ \bar{c}_{2,1} & \bar{c}_{2,2} & \dots & \bar{c}_{2,h} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{c}_{f,1} & \bar{c}_{f,2} & \dots & \bar{c}_{f,h} \end{pmatrix}$$

where each $\bar{c}_{i,j}$, for $1 \leq i \leq f$ and $1 \leq j \leq h$, is a $u$-point trace. For convenience only the field multiplications were considered and the other field operations are discarded.

Two different approaches of using this matrix were 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 were 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 4.1, an arbitrary threshold set to $0.55$ was used to determine which points from which operations were selected. If there are $v$ 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 $T$. Then, as in Section 5.2, we define

$$\begin{aligned} A' &= \{a'_{1,1} \dots a'_{1,v}, a'_{2,1} \dots a'_{2,v}, \dots, a'_{2\,(n-1),1} \dots a'_{2\,(n-1),v}\} \\ &= \{\bar{a}'_1, \bar{a}'_2, \dots, \bar{a}'_{2\,(n-1)}\}, \end{aligned}$$

from points indicate in one dimension and

$$\begin{aligned} B' &= \{b'_{1,1} \dots b'_{1,v}, b'_{2,1} \dots b'_{2,v}, \dots, b'_{2\,(n-1),1} \dots b'_{2\,(n-1),v}\} \\ &= \{\bar{b}'_1, \bar{b}'_2, \dots, \bar{b}'_{2\,(n-1)}\}, \end{aligned}$$

from the other. That is, computing $\rho(\bar{a}'_i, \bar{b}'_i)$ will compute the correlation coefficient between all the combinations of points indicated by $C$.

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

$$\begin{aligned} D &= d_1 \dots d_{n-1} \\ &= \{\rho(\bar{a}'_2, \bar{b}'_3), \rho(\bar{a}'_4, \bar{b}'_5), \dots, \rho(\bar{a}'_{2n-4}, \bar{b}'_{2n-3})\} \end{aligned}$$

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 as before with the results are summarized in Table 1.

In applying the collision attack to traces taken while a 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 form hypotheses on the bits of the exponent as described in Section 5.2. The attack was 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. An 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 $2^{48}$ before the key can be broken.

**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:

$$T = \{\alpha_1, \delta_1, \alpha_2, \delta_2, \ldots, \alpha_{n-1}, \delta_{n-1}\}$$

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 collisions 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 the creation of the output of an addition with the power consumption during the processing of the input to a doubling operation. We shall assume that the power consumption corresponding to a group doubling and a group addition operation consist of $u_d$ and $u_a$ points respectively. We determine a matrix of correlation coefficients $C$

$$C = \begin{pmatrix} \bar{c}_{1,1} & \bar{c}_{1,2} & \cdots & \bar{c}_{1,u_a} \\ \bar{c}_{2,1} & \bar{c}_{2,2} & \cdots & \bar{c}_{2,u_a} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{c}_{u_d,1} & \bar{c}_{u_d,2} & \cdots & \bar{c}_{u_d,u_a} \end{pmatrix}$$

where $\bar{c}_{i,j}$ is the correlation of the $i$-th point from the doubling operation with the $j$-th point from the addition 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 likely 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 $v$ 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 $T$, giving

$$\begin{aligned} A' &= \{a'_{1,1} \ldots a'_{1,v}, a'_{2,1} \ldots a'_{2,v}, \ldots, a'_{2\,(n-1),1} \ldots a'_{2\,(n-1),v}\} \\ &= \{\bar{a}'_1, \bar{a}'_2, \ldots, \bar{a}'_{2\,(n-1)}\}, \end{aligned}$$

and another trace from points indicated on the other dimension

$$\begin{aligned} B' &= \{b'_{1,1} \ldots b'_{1,v}, b'_{2,1} \ldots b'_{2,v}, \ldots, b'_{2\,(n-1),1} \ldots b'_{2\,(n-1),v}\} \\ &= \{\bar{b}'_1, \bar{b}'_2, \ldots, \bar{b}'_{2\,(n-1)}\}. \end{aligned}$$

Hence, computing $\rho(\bar{a}'_i, \bar{b}'_i)$ 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

$$\begin{aligned} D &= d_1 \dots d_{n-1} \\ &= \{\rho(\bar{a}'_2, \bar{b}'_3), \rho(\bar{a}'_4, \bar{b}'_5), \dots, \rho(\bar{a}'_{2n-4}, \bar{b}'_{2n-3})\} \end{aligned}$$

as described in Section 4.1 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 for our SASEBO implementation. We assume that the leakage required to conduct this attack is not present. That is, the data does not leak in a similar at two different points in time.

# 6 Other Exponentiation Algorithms

In the previous section we demonstrate that an attacker is able to analyze implementations that use a variety of methods to compute a group exponentiation. In the following we shall define what we consider an attacker is able to do based on our experiments. Using this model we shall demonstrate how an attack can be applied to algorithms that have previously been considered resistant to attacks exploiting collisions.

## 6.1 Attack Model

We shall consider an attacker capable of the following

1. An attacker can take traces during the computation of a group exponentiation algorithm. However, the exponent used each time is considered to be distinct, i.e. it is a nonce or blinded.
2. An attacker is able to distinguish individual group operations and extract these as subtraces.
3. An attacker is only able to distinguish if an operation is a squaring where the algorithm is different (as with a doubling operation in elliptic curve arithmetic). For ease of expression we shall not take into account any information available via this side-channel. However, an attacker is able to compare dissimilar operations as described in Section 5.2.
4. An attacker is able to detect collisions where the same input is used in two different operations with a high probability (as shown in Sections 5.2 and 5.2). Note that ordering is important as the variables need to appear in the same points in the trace. That is, for a given group operation $\otimes$ and group elements $x$, $y$ and $z$, an attacker can detect a collision between $x \otimes z$ and $x \otimes y$, but not $x \otimes z$ and $y \otimes x$.
5. An attacker is able to detect a collision where any input or output of an operation is used in another operation with a high probability (as shown in Section 5.2), but less than for Property 4. In the above example an attacker would, for example, be able to detect a collision between $x \otimes z$ and $y \otimes x$, or where an operation uses the output of some previous operation.

Hence, an attacker will seek to determine collisions where the same input can be seen in more than one operation. Where necessary an attacker will attempt to determine collisions between arbitrary elements using the method described in Section 5.2. However, the observing the same variable in the same position is preferable since that form of the attack will succeed with a higher probability.

**Table 2.** An Example of a Table

| $(r, r')$ | Triples |
|---|---|
| $(2, 0)$ | $(111)$ |
| $(2, 1)$ | $(112, 133)$ |
| $(3, 0)$ | $(112, 121)$ |
| $(3, 1)$ | $(112, 133, 121)$ |
| $(3, 2)$ | $(112, 233, 121)$ |
| $(5, 0)$ | $(112, 121, 121)$ |
| $(5, 1)$ | $(112, 133, 121, 121)$ |
| $(5, 2)$ | $(112, 233, 121, 121)$ |
| $(5, 3)$ | $(112, 121, 133, 121)$ |
| $(5, 4)$ | $(112, 222, 233, 121)$ |

### 6.2 MIST

The MIST algorithm [21] was proposed by Walter as a side-channel resistant exponentiation algorithm. Walter studied the problem of how much information one could determine on a hidden exponent from a single trace [15]. Here we summarize and extend his work.

The MIST algorithm is defined as a random-ary exponentiation method similar to a right-to-left m-ary exponentiation algorithm. However, digits are be read from the exponent with a randomly varying base. This produces Algorithm 4 where Walter defines $S = \{2, 3, 5\}$.

**Algorithm 4 MIST** *[21]*

***Require:*** $\boldsymbol{P}$ *a point on elliptic curve* $\mathcal{E}$*, an* $n$*-bit scalar* $k$
***Ensure:*** $\boldsymbol{Q} = k\,\boldsymbol{P}$

1: $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$
2: $\boldsymbol{R}_1 \leftarrow \boldsymbol{\mathcal{O}}$
3: ***while*** $k > 0$ ***do***
4: *Choose a random base* $r$ *from set* $S$
5: $r' \leftarrow k \bmod r$
6: ***if*** $r' \neq 0$ ***then***
7: $\boldsymbol{R}_1 \leftarrow r'\,\boldsymbol{R}_0 + \boldsymbol{R}_1$
8: ***end if***
9: $\boldsymbol{R}_0 \leftarrow r\,\boldsymbol{R}_0$
10: $k \leftarrow k/r$
11: ***end while***
12: ***return*** $\boldsymbol{R}_1$

Given our attack model we shall assume that an attacker is able to extract the group operations and determine how many operations there are in one loop, given that the effect of line 10 should be visible in a trace.

When studying the side-channel resistance of MIST, Walter carefully constructed sequences of operations using three registers to limit the information available to an attacker. These operations can be expressed as a triplet $\{i, j, k\}$, denoting register $i$ combined with register $j$ using the group operation, and writing the result to register $k$. A sequence of these triplets correspond to each possible combination of $r$ and $r'$ as shown in Table 2. In order to extract an exponent one needs to be able to identify all the pairs of values $(r, r')$ used.

Walter details how one could attack Algorithm 4 by detecting collisions between variables being reused in operations. We omit the details for brevity and refer the reader to Walter [15]. However, Walter focuses on the reuse of

variables, Property 4 in our model, which limits the amount of information an attacker can deduce, resulting in the following Theorem:

**Theorem 1.** *(Theorem 8 [15]) If an attacker can determine operand re-use from side-channel leakage, then he can almost certainly deduce the sequence of pairs $(r, r')$ used in the MIST exponentiation scheme up to an ambiguity between $(2, 1)$ and $(3, 0)$. This reduces the search space for the current exponent $k$ to about $k^{1/3}$.*

Given that Property 5 of our model allows an attacker to find collisions between the output of one operation and the input of another Theorem 1 is not sufficient. In order to distinguish between $(2, 1)$ and $(3, 0)$ one needs to distinguish the sequences

$$(112, 133) \quad \text{and} \quad (112, 121)$$

respectively. We note that the output of the first operation, 112 in both cases, is used as the input to the second operation when $(r, r') = (3, 0)$ and not when $(r, r') = (2, 1)$. Hence, an attacker would be able to use this difference to detect a collision on the use of the second register. This would reduce the number of hypotheses for $k$ from $k^{1/3}$ to 1 with a probability determined by the platform being attacked.

### 6.3 Belenky et al.

A more recent attempt to define algorithms that are resistant to attacks exploiting collisions has been proposed by Belenky et al. [22]. In the following we detail how their algorithms can be attacked using the attack model defined in Section 6.1. The first algorithm defined by Belenky et al. is given in Algorithm 5.

**Algorithm 5 Belenky et al. Exponentiation Algorithm I** *[22]*
***Require:*** $\boldsymbol{P}$ *a point on elliptic curve $\mathcal{E}$, an addition chain $e(j)$ ($\in \{0, 1\}$) of a scalar $k$ for $j \in \mathbb{Z}$, $0 \leq j \leq n-2$*
***Ensure:*** $\boldsymbol{Q} = k\,\boldsymbol{P}$
1: $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$
2: $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$
3: ***for*** $i \leftarrow 0$ **to** $n-1$ ***do***
4: $\quad \boldsymbol{R}_{e(i)} \leftarrow \boldsymbol{R}_{1-e(i)} + \boldsymbol{R}_{e(i)}$
5: ***end for***
6: ***return*** $\boldsymbol{R}_0 + \boldsymbol{R}_1$

A collision characteristic for Algorithm 5 can be defined as follows:

**Lemma 5.** *If the bit values $e(i)$ and $e(i+1)$, for some $i \in \{0, \ldots, n-3\}$, treated in two consecutive loops are same, the first input operands of two operations are the same.*

*Proof.* If we consider that two consecutive loops of Algorithm 5, say rounds $\ell$ and $\ell + 1$, we define the contents of register $R_{i,j}$ as the contents of register $i$ in loop $j$. Then for $i \in \{0, 1\}$ and $j \in \{0, 1\}$,

$$t(u) = i, t(u+1) = j \implies \begin{cases} \boldsymbol{R}_{i,\ell+1} \leftarrow \boldsymbol{R}_{\neg i,\ell} + \boldsymbol{R}_{i,\ell}\,, \\ \boldsymbol{R}_{j,\ell+2} \leftarrow \boldsymbol{R}_{\neg j,\ell+1} + \boldsymbol{R}_{j,\ell+1}\,. \end{cases}$$

Hence, $e(\ell) = e(\ell+1) \implies \boldsymbol{R}_{\neg e(\ell),\ell} = \boldsymbol{R}_{\neg e(\ell+1),\ell+1}\,.$ □

Given Lemma 5 and using Property 4 from our attack model, an attacker will be able to determine hypotheses for all $e(\ell)$ for $\ell \in \{1, \ldots, n-1\}$ given a guess for $e(0)$.

The second algorithm proposed by Belenky et al. is shown in Algorithm 6, which is a right-to-left exponentiation algorithm using dummy operations.

**Algorithm 6 Belenky et al. Exponentiation Algorithm II** *[22]*

***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:* $\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$
*2:* $\boldsymbol{R}_1 \leftarrow \boldsymbol{\mathcal{O}}$
*3:* $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$
*4:* ***for*** $i \leftarrow 0$ **to** $n-2$ ***do***
*5:* $\quad \boldsymbol{R}_{k_i} \leftarrow \boldsymbol{R}_{k_i} + \boldsymbol{R}_2$
*6:* $\quad \boldsymbol{R}_2 \leftarrow \boldsymbol{R}_2 + \boldsymbol{R}_2$
*7:* ***end for***
*8:* $\boldsymbol{R}_0 \leftarrow \boldsymbol{R}_0 + \boldsymbol{R}_1$
*9:* $\boldsymbol{R}_0 \leftarrow \boldsymbol{R}_0 + \boldsymbol{R}_1$
*10:* $\boldsymbol{R}_1 \leftarrow \boldsymbol{R}_1 + \boldsymbol{R}_2$
*11:* ***return*** $\boldsymbol{R}_0 + \boldsymbol{R}_1$

A collision characteristic for Algorithm 6 can be defined as follows:

**Lemma 6.** *If the bit values* $k_i$ *and* $k_{i+1}$*, for some* $i \in \{0, \ldots, n-3\}$*, treated in two consecutive loops are the same, them the result in line 5 of loop* $i$ *will be used in the input to in line 5 of loop* $i+1$*.*

*Proof.* Trivially, we assume that an adversary knows the $\ell$ least significant bits of the scalar. Then in the $(\ell+1)$-th loop $\boldsymbol{R}_{k_i}$ remains the same when $(\ell+1)$-th least significant bit of the scalar is equal to $\ell$-th least significant bit. □

Given Lemma 6 and using Property 5 from our attack model, an attacker can form hypotheses on the bits of $k$ and implement a practical attack.

## 7 Countermeasures

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 values are reused once created 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* $\mathcal{H} = g_i$ *for* $i \in \{1, \ldots, \kappa\}$ *and some function* $\mathcal{H}$ *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 7.** *A totally regular addition chain-based exponentiation algorithm that can compute* $s^\chi$ *for any* $s, \chi \in \mathbb{Z}_{>0}$*, assuming* $\chi$ *has a fixed bit length, does not exist.*

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

$$\mathcal{F} : \mathbb{Z}_{>0} \longrightarrow \mathbb{Z}_{>0} : s \longmapsto s^{\chi}$$

that uses addition chain $\mathcal{A}$ to compute $\mathcal{F}(n)$ for some arbitrary $\chi$ of a given bit length. If $\mathcal{F}$ is totally regular this would imply that $\mathbf{A}$ would remain unchanged for all possible exponents that have the same bit length as $\chi$. However, by Definition 1, the last element of $\mathcal{A}$ is $s^{\chi}$, and we note that $\chi$ cannot be changed by modifying $s_0, s_1$. Hence

$$\mathcal{F} \iff \mathcal{A}$$

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 [41] for a thorough treatment of this topic.

## 8 Conclusion

We describe attacks based on collisions of arbitrarily chosen 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 practical attacks described in the literature that typically require numerous traces to be acquired [8–11, 14]. 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 [23] and Coron's double-and-add-always [25] 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. Furthermore, 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. Hence, demonstrating that the attacks we define can be implemented using a single trace, justifying the novel extended attack we define for the Montgomery ladder.

We define an attack model based on our practical work and demonstrate that this model can be used to describe attacks on exponentiation algorithms otherwise considered resistent to collision attacks. Specifically, the exponentiation algorithm MIST proposed by Walter [21] and two exponentiation algorithms recently proposed by Belenky et al. [22]. Furthermore, 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 such that an attack is not practical.

In the case of the Joye's add-only [23] 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 $1$. We do not present this in the main body of our work since we believe it is not likely to be reproduced on a different platform, and 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 simple, effective 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.

## Acknowledgments

The authors would like to thank Georg Fuchsbauer, Seokhie Hong and Elisabeth Oswald for their helpful comments. This research was supported by the MSIP(Ministry of Science, ICT & Future Planning), Korea, under the C-ITRC (Convergence Information Technology Research Center) support program (NIPA-2013-H0301-13-3007) supervised by the NIPA(National IT Industry Promotion Agency). The work described in this paper has also been supported in part the European Commission through the ICT Programme under Contract ICT-2007-216676 ECRYPT II and the EPSRC via grant EP/I005226/1.

## References

1. R. Rivest, A. Shamir, and L. M. Adleman, "Method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
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. P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *CRYPTO '99*, ser. LNCS, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 388–397.
4. M. Joye and M. Tunstall, "Exponent recoding and regular exponentiation algorithms," in *AFRICACRYPT 2009*, ser. LNCS, B. Preneel, Ed., vol. 5580. Springer, 2009, pp. 334–349.
5. K. Schramm, T. Wollinger, and C. Paar, "New class of collision attacks and its application to DES," in *FSE 2003*, ser. LNCS, T. Johansson, Ed., vol. 2887. Springer, 2003, pp. 206–222.
6. C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil, "Improved collision-correlation power analysis on first order protected AES," in *CHES 2011*, ser. LNCS, B. Preneel and T. Takagi, Eds., vol. 6917. Springer, 2011, pp. 49–62.
7. C. D. Walter, "Sliding windows succumbs to big mac attack." in *CHES 2001*, ser. LNCS, Ç. K. Koç, D. Naccache, and C. Paar, Eds., vol. 2162. Springer, 2001, pp. 286–299.
8. F. Amiel and B. Feix, "On the BRIP algorithms security for RSA," in *WISTP 2008*, ser. LNCS, J. A. Onieva, Ed., vol. 5019. Springer, 2008, pp. 136–149.
9. H. Kim, T. H. Kim, J. C. Yoon, and S. Hong, "Practical second-order correlation power analysis on the message blinding method and its novel countermeasure for RSA," *ETRI Journal*, vol. 32, no. 1, pp. 102–111, 2010.
10. H. Mamiya, A. Miyaji, and H. Morimoto, "Efficient countermeasures against RPA, DPA, and SPA," in *CHES 2004*, ser. LNCS, M. Joye and J.-J. Quisquater, Eds., vol. 3156. Springer, 2004, pp. 343–356.
11. M. F. Witteman, J. G. J. van Woudenberg, and F. Menarini, "Defeating RSA multiply-always and message blinding countermeasures," in *CT-RSA 2011*, ser. LNCS, A. Kiayias, Ed., vol. 6558. Springer, 2011, pp. 77–88.
12. P. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *Mathematics of Computation*, vol. 48, no. 177, pp. 243–264, 1987.
13. M. Joye and S.-M. Yen, "The montgomery powering ladder," in *CHES 2002*, ser. LNCS, B. S. K. Jr., Ç. K. Koç, and C. Paar, Eds., vol. 2523. Springer, 2003, pp. 291–302.
14. K. Okeya and K. Sakura, "A second-order DPA attack breaks a window-method based countermeasure against side channel attacks," in *ISC 202*, ser. LNCS, A. H. Chan and V. Gligor, Eds., vol. 2433. Springer, 2002, pp. 389–401.
15. C. D. Walter, "Some security aspects of the MIST randomized exponentiation algorithm," in *CHES 2002*, ser. LNCS, B. S. K. Jr., Ed., no. 2523. Springer, 2003, pp. 276–290.
16. P.-A. Fouque and F. Valette, "The doubling attack — Why upwards is better than downwards," in *CHES 2003*, ser. LNCS, C. D. Walter, Ç. K. Koç, and C. Paar, Eds., vol. 2779. Springer, 2003, pp. 269–280.
17. S.-M. Yen, L.-C. Ko, S.-J. Moon, and J. Ha, "Relative doubling attack against Montgomery ladder," in *ICISC 2005*, ser. LNCS, D. Won and S. Kim, Eds., vol. 3935. Springer, 2005, pp. 117–128.
18. N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Shamir, "Collision-based power analysis of modular exponentiation using chosen-message pairs," in *CHES 2008*, ser. LNCS, E. Oswald and P. Rohatgi, Eds., vol. 5154. Springer, 2008, pp. 15–29.
19. S.-M. Yen, W.-C. Lien, and C.-N. Chen, "Modified doubling attack by exploiting chosen ciphertext of small order," *IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences*, vol. E94, no. A, pp. 1981–1990, 2011.
20. A. X9.62, "Public key cryptography for the financial services industry, the elliptic curve digital signature algorithm (ECDSA)," 1999.
21. C. D. Walter, "MIST: Efficient, randomized exponentiation algorithm for resisting power analysis," in *CT-RSA 2002*, ser. LNCS, B. Preneel, Ed., no. 2271. Springer, 2002, pp. 53–66.
22. Y. Belenky, Z. Geyzel, M. Kara-Ivanov, and A. Entelis, "Two exponentiation algorithms resistant to cross-correlation power analysis and to other known attacks," Cryptology ePrint Archive, Report 2012/723, 2012, http://eprint.iacr.org/.

23. M. Joye, "Highly regular right-to-left algorithms for scalar multiplication," in *CHES 2007*, ser. LNCS, P. Paillier and I. Verbauwhede, Eds., vol. 4727. Springer, 2007, pp. 135–147.
24. Wireless Application Protocol (WAP) Forum, "Wireless transport layer security (WTLS) specification," Available from http://www.wapforum.org.
25. J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *CHES 1999*, ser. LNCS, C. K. Koç and C. Paar, Eds., vol. 1717. Springer, 1999, pp. 292–302.
26. B. Chevallier-Mames, M. Ciet, and M. Joye, "Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity," *IEEE Transactions on Computers*, vol. 53, no. 6, pp. 760–768, 2004.
27. A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir, "Key recovery attacks of practical complexity on AES-256 variants with up to 10 rounds," in *EUROCRYPT 2010*, ser. LNCS, H. Gilbert, Ed., vol. 6110. Springer, 2010, pp. 299–319.
28. D. Stinson, "Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem," *Mathematics of Computation*, vol. 71, no. 237, pp. 379–391, 2002.
29. E. D. Win, S. Mister, B. Preneel, and M. Wiener, "On the performance of signature schemes based on elliptic curves," in *ANTS 1998*, ser. LNCS, J.-P. Buhler, Ed., vol. 1423. Springer, 1998, pp. 252–266.
30. T. Izu and T. Takagi, "Fast elliptic curve multiplications resistant against side channel attacks," *IEICE Transactions*, vol. 88-A, no. 1, pp. 161–171, 2005.
31. J. Großschädl, E. Oswald, D. Page, and M. Tunstall, "Side channel analysis of cryptographic software via early-terminating multiplications," in *ICISC 2009*, ser. LNCS, D. Lee and S. Hong, Eds., no. 5984. Springer, 2009, pp. 176–192.
32. P. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.
33. A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.
34. Research Center for Information Security, "Side-channel Attack Standard Evaluation Board (SASEBO)," http://www.risec.aist.go.jp/project/sasebo/, 2002, online; accessed 05-Feb-2013.
35. C. K. Koc, T. Acar, and B. S. Kaliski Jr, "Analyzing and comparing montgomery multiplication algorithms," *Micro, IEEE*, vol. 16, no. 3, pp. 26–33, 1996.
36. H. Cohen, A. Miyaji, and T. Ono, "Efficient elliptic curve exponentiation using mixed coordinates," in *ASIACRYPT '98*, ser. LNCS, K. Ohta and D. Pei, Eds., vol. 1514. Springer, 1998, pp. 51–65.
37. D. J. Bernstein and T. Lange, "Faster addition and doubling on elliptic curves," in *ASIACRYPT 2007*, ser. LNCS, K. Kurosawa, Ed., vol. 4833. Springer, 2007, pp. 29–50.
38. M. Keller, A. Byrne, and W. P. Marnane, "Elliptic curve cryptography on fpga for low-power applications," *TRETS*, vol. 2, no. 1, 2009.
39. C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil, "Horizontal correlation analysis on exponentiation," in *ICICS 2010*, ser. LNCS, M. Soriano, S. Qing, and J. López, Eds., vol. 6476. Springer, 2010, pp. 46–61.
40. S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks — Revealing the Secrets of Smart Cards*. Springer, 2007.
41. J. Fan and I. Verbauwhede, "An updated survey on secure ECC implementations: Attacks, countermeasures and cost," in *Cryptography and Security*, ser. LNCS, D. Naccache, Ed., vol. 6805. Springer, 2012, pp. 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 $\alpha$ and $\beta$.*

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

**Lemma 8.** *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 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 [28].

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 $i$th 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 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 $n$-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 [28], 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 8 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.