Exploiting Collisions in Addition Chain-based Exponentiation Algorithms

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
[2] Department of Computer Science, University of Bristol
Merchant Venturers Building, Woodland Road
Bristol BS8 1UB, United Kingdom
[3] Center for Information Security Technologies, Korea University
Seoul, Republic of Korea

Abstract. Public key cryptographic algorithms are typically based on group exponentiation algorithms, and many algorithms have been proposed in the literature based on addition chains. We describe attacks based on collisions of variables manipulated in group operations extending attacks described in the literature. The advantage of our attacks over previous work is that the attacks can be applied to a single trace and do not require any knowledge of the input to the exponentiation algorithm. Moreover, we prove that our attacks are applicable to all addition chain-based exponentiation algorithms. 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.

1 Introduction

It has been shown in the literature that an attacker 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 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 attacker 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 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 a recent results and a survey of the relevant publications).

Amiel and Feix [7] and Kim et al. [8] describe an attack applicable to RSA [1] where a signature is generated using the BRIP exponentiation algorithm [9]. Similarly, recent work by Witteman et al. [10] describe an attack applicable to RSA where a signature is generated using Coron's multiply-always algorithm [11] where the message does not need to be known. We demonstrate that this attack can be applied to a scalar multiplication where the addition and doubling operations use different algorithms.

All of the attacks described in this paper assume that the base point is blinded and an attacker is therefore unaware of the input to the scalar multiplication under attack. Moreover, we assume that the exponent is blinded, or a random value, so that an attacker is obliged to use one trace to conduct an attack. This is a substantial improvement over previous work, where implementations of attacks have required that numerous traces are averaged [8, 10] or random values have the same value [7]. We describe attack strategies for three exponentiation algorithms and then prove that any addition chain-based exponentiation algorithm is vulnerable to these side channel attacks. Therefore, the only countermeasure is to add sufficient noise that the time complexity of an an attack increases to the point where it is impractical.

The remainder of this paper is organized as follows: in Section 2 we introduce some background on exponentiation algorithms, elliptic curves and Stinson's Baby Step/Giant Step algorithm. In Section 4 we detail our attacks including practical results. We discuss countermeasures in Section 5, after which we conclude.

2 Preliminaries

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.*

In this paper we will discuss addition chain-based exponentiation in terms of the integers $\mathbb{Z}_{>0}$ and scalar multiplication of a point over an elliptic curve (described in more detail below). While the group operation in elliptic curve arithmetic is an addition rather than a multiplication it is analogous to multiplication over the integers. It should be clear from the context to which we are referring.

In order to compute $z = n^{\lambda}$ over the integers $\mathbb{Z}_{>0}$ efficiently it is, for example, possible to use an algorithm that is described in [12]. In this algorithm one precomputes some values that are small powers of n and the exponent is broken up into ℓ words in base m (this is called the m-ary expansion and ℓ is called the m-ary length). Typically, m is chosen to be equal to 2^k, for some convenient value of k, to enable the relevant digits to simply be read from the exponent. The m-ary algorithm is shown in Algorithm 1, where we define the function `digit(`z, i`)` as a function returning the i-th radix-m digit of z. If m is set to 2 then the resulting algorithm is typically referred to as the binary exponentiation algorithm.

Algorithm 1: m-ary Exponentiation Algorithm

Input: $n, \lambda \in \mathbb{Z}_{>0}$, μ the m-ary length of n
Output: $z = n^{\lambda}$

1 $R[1] \leftarrow n$;
2 **for** $i \leftarrow 2$ **to** $m - 1$ **do** $R[i] \leftarrow R[i-1] \otimes n$;
3 $b \leftarrow$ `digit(`$\lambda, \mu - 1$`)` ;
4 $A \leftarrow R[b]$;
5 **for** $i = \mu - 2$ **down to** 0 **do**
6 $\quad A \leftarrow A^m$;
7 $\quad b \leftarrow$ `digit(`λ, i`)` ;
8 $\quad$ **if** $(b \neq 0)$ **then** $A \leftarrow A \otimes R[b]$;
9 **end**
10 **return** A

In all instances of Algorithm 1 a given element in the addition chain is either produced by squaring the element a certain number of times, i.e., raising the previous element to the power of m, or it is multiplied by one of the $2^m - 1$ first elements representing $n, n^2, n^3, \ldots, n^{2^m-1}$.

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, e.g. [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 do not present examples of highly regular exponentiation algorithms here since they are discussed 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 $a_i \in \mathbb{F}_q$ $(1 \leq i \leq 6)$, 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

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

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

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 the attacks described in this paper since it is used in standards such as the FIPS 186-3 [2], WTLS [15] and ANSI X9.62 [16].

When implementing a scalar multiplication using elliptic curve arithmetic on a device that could potentially be attacked using side-channel analysis, there are a variety of considerations that need to be taken into account. The simplest such attacks involve observing the time required to compute an operation [17] or by inspecting a power consumption trace to observe implementation differences [3]. A scalar multiplication is typically implemented such that an attacker cannot distinguish a point addition from a point doubling operation in a given instance of scalar multiplication. This is typically achieved by implementing a highly regular addition chain-based as defined in Definition 2 (a survey is given in [4]).

2.3 The Discrete Logarithm Problem

We recall the discrete logarithm problem:

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

An Elliptic Curve Discrete Logarithm Problem (ECDLP) can be posed by considering scalar multiplication of some point $\boldsymbol{Q} = k\,\boldsymbol{P}$ where one will attempt to recover k given $\boldsymbol{P}$ and $\boldsymbol{Q}$. For ease of expression we shall discuss this problem in terms of the integers.

In a side channel analysis of a given instance of an exponentiation algorithm the results can only give the best guess of the exponent. The number of bits that are correct will depend on the attack conducted. In the case we consider in this paper we assume that side channel analysis returns a guess where a small number of bits of the exponent are incorrect.

Stinson describes a variant of the Baby-Step/Giant-Step algorithm where it is assumed that the exponent has a small Hamming weight [18]. Stinson's algorithm require the existence of a means of splitting a string of bits into two sets of equal Hamming weight.

Lemma 1. *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 weigh 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 [18].

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 $e_{i,1}$. After precomputing α^{2^i} and α^{-2^i}, for $i = 0, \ldots, n-1$, computing the these Giant-Steps requires $2^{t/2}$ group operations via a recursive procedure. The Giant-Steps are, as usual, held via a hash table which allows quick searching for entries on the first index.

The Baby-Steps can then be computed so as to match them against the Giant-Steps. Again we produce a list of error vectors of Hamming weight $t/2$ where we define the j-th error from the set of possible errors e_2 as $e_{j,2}$. 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 defined by Stinson [18], and has time complexity of $\mathcal{O}(m \binom{m/2}{t/2})$. However, this assumes that t is known. If t is not known then an attacker has to start with $t = 1$ and increase the Hamming weight until t is found, giving a time complexity of $\mathcal{O}(m\, t \binom{m/2}{t/2})$.

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 [19], which requires an attacker 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 attacker could then focus on adjacent operations executed during the computation of $k\,\boldsymbol{P}$ and $k(2\,\boldsymbol{P})$. If we consider, for example, the double and add always algorithm [11] (see Algorithm 4), where we define the function $\texttt{bit}\,(n, i)$ as a function returning the $(i-1)$-th least significant bit of n), an attacker 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 same input point. This would allow an attacker 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 [11] 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 (e.g. as used in the Montgomery ladder) exponentiation algorithms.

Algorithm 2: Double and add always algorithm [11]

Input: $n \in \mathbb{G}$ for a group $\mathbb{G}$ under operation $\otimes$, $\lambda \in \mathbb{Z}_{>0}$
Output: $z \in \mathbb{G}$ where $z = n^{\lambda}$
Uses: $R[i]$ for $i \in \{0, 1\}$

```
1 R[0] ← n ;
2 for i ← n − 2 down to 0 do
3 |   R[0] ← R[0]^2 ;
4 |   b ← bit(λ, i) ;
5 |   R[¬b] ← R[¬b] ⊗ n ;
6 end
7 return R[0]
```

Yen at al. [20] extended the attack proposed in [19] to apply it to the Montgomery ladder. The attack functioned using the same chosen input as described above, i.e. points $\boldsymbol{P}$ and $2\,\boldsymbol{P}$, but an attacker is only able to deduce whether two adjacent bits of the scalar are equal (if a collision occurred) or not (otherwise). Another very specific attack described by Yen et al. [21] uses chosen messages X and $-X$ as inputs to exponentiation in $\mathbb{Z}_n$. Collisions will occur at the same point in time when exponent bits are equal to zero as the output of the squaring operations will be identical.

A more recent result on the doubling attack has been presented by Homma et al. [22]. 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 $\beta \leq \alpha$ 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 an FPGA.

The above attacks require that an attacker 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 attacker. Okeya and Sakura [23] 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. Similar attacks have been applied to block ciphers applied with some success (see [6] for a recent results and a survey of the relevant publications).

Amiel and Feix [7] and Kim et al. [8] describe some attacks and implementations of the strategy proposed by Okeya and Sakura. Thy describe a collision attacks on the BRIP exponentiation algorithm [9]. 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 attacker can then compute the cross-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. [10] applicable to Coron's double and add always exponentiation algorithm. Witteman

et al. demonstrate that this attack works on an ASIC implementation. However, the attacks referenced above require numerous acquisitions to be taken to reduce noise to the point where an attack will function.

Kim et al. [8] also describe how one could apply such an attack the the Montgomery Ladder [14]. 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 attacker 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.

4 Attacking Addition Chain-based Exponentiation Algorithms

In this section we describe how one can attack different group exponentiation algorithms using collisions between group operations using one acquisition. 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 homogenous 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 exponentiation algorithm we apply an attack based on collisions to an implementation on an ARM7TDMI microprocessor, where 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 cut-off frequency set to 7.37 MHz. Experimentation while conducting the attacks described below determined that this removed any high frequency noise and improved the success rate of any subsequent side channel analysis.

In each case the attack starts by dividing the acquired traces into individual group operations. If we inspect the beginning of a power consumption trace the individual modular multiplications are readily visible. For example, Figure 1 shows a portion of a power consumption trace where nine such multiplications are visible as a repeating pattern. Given the possible group operations one can determine how many multiplications are required to compute each group operation and thus identify a trace corresponding to an individual group operation. Moreover, the gaps between the multiplication operations will confirm hypotheses since these will contain additions, subtractions, etc. required to compute a group operation. This can be observed in Figure 1, where the gaps between the last three multiplications are larger than the other gaps between multiplications.

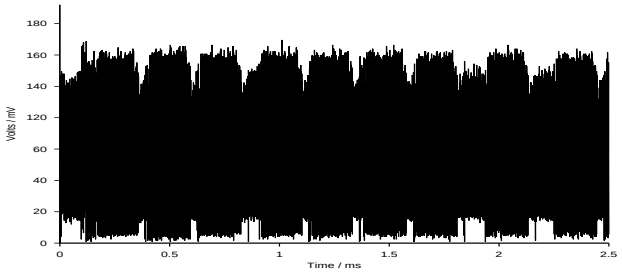
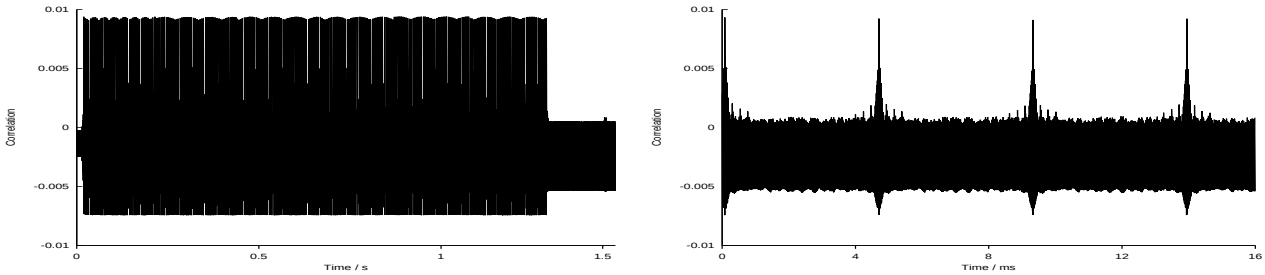


Fig. 1. An example power consumption trace showing the individual multiplication operations in a group operation.

Various methods of extracting the group operations from an acquisition were explored based on using the cross-correlation, Euclidean distance and least-squares. All the methods gave similar results, however the cross-correlation was significantly faster to execute since it can be implemented in the frequency domain.

We extracted the portion of a trace where one group operation has been identified from a set of 100 power consumption traces acquired while computing one of the group exponentiation algorithms described below. This produced a set of 100 small traces that were then used to find corresponding to features in the traces showing the power consumption traces corresponding to the entire group exponentiation algorithms. The small traces were compared to a

window of the same size from the large traces with every possible offset. This would generate a trace of correlation values showing the location of group operations. In Figure 2(a) we show the resultant trace for a scalar multiplication with a random 192-bit scalar. Figure 2(b) is a small portion of this trace where one can clearly observe peaks that indicate the location of a group operation.



(a) Cross-correlation of a mean trace for one operation with a window of the same size starting from every possible point within an acquisition.

(b) Zoomed image showing distinct peaks.

Fig. 2. Determining the location of individual operations in a power consumption trace.

From this analysis we can break a trace up into smaller traces, where each small trace corresponds to a single group operation. If we consider the i-th trace $S^{(i)}$ from a set of N traces we can describe the trace as a series of n traces

$$S^{(i)} : O_1^{(i)}; O_2^{(i)}; O_3^{(i)}; \ldots; O_{n-1}^{(i)}; O_n^{(i)} ,$$

where the exponentiation algorithm consists of n groups operations. Each $O_j^{(i)}$ will be the power consumption of the j-th operation in the i-th trace. We can therefore identify the power consumption while a microprocessor is computing an arbitrary group operation.

4.1 Highly-Regular Right-to-Left Scalar Multiplication

The first attack that we will consider was applied to Joye's Add-Only group exponentiation algorithm [13]. In the additive group formed over an elliptic curve the resulting algorithm is shown in Algorithm 3. We have chosen this algorithm to demonstrate a simple version of this type of attack and we discuss more complex variants below. The attack described below is a straightforward variant of the attacks described by Kim et al. [8] and Witteman et al. [10] applied to a different algorithm. While a set of traces is used to determine the location of individual group operations, as described above, we assume that an attacker can only attack one trace.

Algorithm 3: Joye's Add-Only Right-to-Left Scalar Multiplication [13]

Input: $\boldsymbol{P}$ a point on elliptic curve $\mathcal{E}$, an n-bit scalar $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Output: $\boldsymbol{Q} = k\,\boldsymbol{P}$

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

One can observe that there will be collisions between variables:

Lemma 2. *In one loop of Algorithm 3, $\boldsymbol{R}_0$ remains the same when $k_{\ell+1} = 0$ and $\boldsymbol{R}_1$ remains the same when $k_{\ell+1} = 1$.*

Proof. We assume that an attacker knows the ℓ least significant bits of the scalar, and define $k' = (k_{\ell-1} \dots k_0)_2$ then we wish ti 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 ℓ-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, at the end of the ℓ-th round $\boldsymbol{R}_0 = k' \boldsymbol{P}and\boldsymbol{R}_1 = (2^\ell - k')\,\boldsymbol{P}$ changes 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.

If $k_{\ell+1} = 1$ then, at the end of the ℓ-th round $\boldsymbol{R}_0 = k' \boldsymbol{P}and\boldsymbol{R}_1 = (2^\ell - k')\,\boldsymbol{P}$ changes to

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

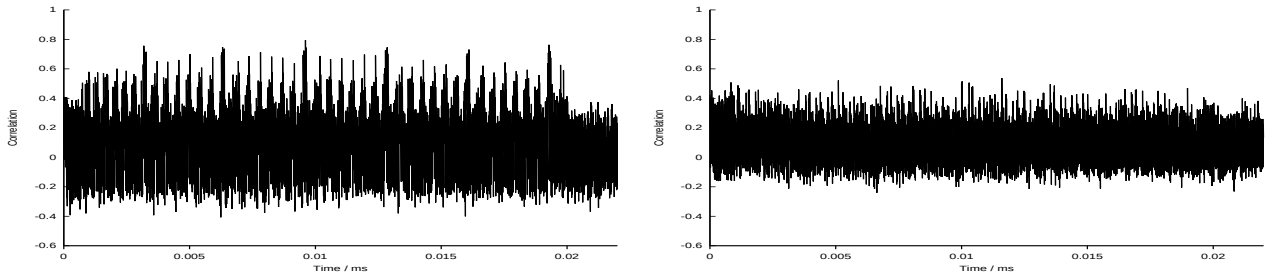
in the $(\ell+1)$-th round.

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

Given Lemma 2, one needs to determine the points within a given group operation that can be used to observe these collisions. We can note that for a given group operation the inputs are typically used several times. For example, using homogenous projective coordinates in a point addition algorithm, such as the x-only operations proposed by Izu and Takagi [25], the z-coordinate is used three times as an operand in a field multiplication.

To begin determining how to observe a collision we need to characterize the points within an individual multiplication that can be exploited. Within one group operation there are typically several multiplications that will have the same value for one operand. We can compare two sets of n-traces by computing the correlation between the points of the traces in the two sets. That is, for two sets of n traces of m points $A = \{A_{i,1}, A_{i,2}, \dots, A_{i,m}\}$ and $B = \{B_{i,1}, B_{i,2}, \dots, B_{i,m}\}$ for $i \in \{1, 2, \dots, n\}$, we compute the correlation as a trace $\{\zeta_1, \zeta_2, \dots, \zeta_m\} = \{\rho(A_{i,1}, B_{i,1}), \rho(A_{i,2}, B_{i,2}), \dots, \rho(A_{i,m}, B_{i,m})\}$ for $i \in \{1, 2, \dots, n\}$. In our case m is the length of a multiplications trace and n is taken to be 192, i.e. the bit length of the scalar.

In our implementation there are 15 multiplications giving 105 possible ways of comparing two multiplications, of which seven combinations show some significant correlation, an example is shown in Figure 3(a). If two multiplications show no correlation we observe a trace as shown in Figure 3(b).



(a) Correlation traces where one operand is the same in both multiplications.

(b) Correlation traces where no operands are the same.

Fig. 3. Correlation traces for identifying the interesting points in a multiplication.

These traces indicate which points can be compared to determine whether a collision has occurred. We chose the points from the correlation trace where the correlation coefficient is superior to 0.55.

One then has to determine which multiplications within a group operation contain useful information that can then be used to identify where collisions have occurred. We conducted the attack outlined above independently, using the

points identified as information bearing, where for each instance we used one of the fifteen multiplications required to compute a group operation. This produced fifteen estimates for the scalar used in the scalar multiplication. We assume that the first estimate is somewhat correct since the first multiplication will have to operate on at least one of the inputs to the group operation so a collision will occur. We then computed the correlation between the first key estimate and the estimates derived from the other multiplications, the results of which are shown in Figure 4. The multiplications that are of interest are clearly indicated.

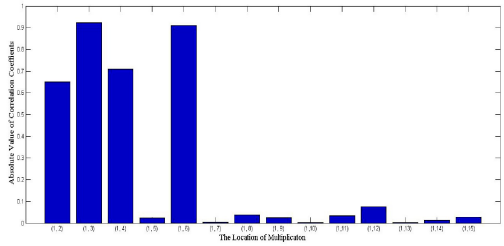


Fig. 4. Determining useful multiplications in an addition operation.

The points within the indicated multiplications can then be used to derive a scalar using the power consumption while a target device is computing a scalar multiplication following the attack outlined above. We conducted this attack on 50 power consumption traces acquired while the target ARM microprocessor was computing a scalar multiplication with a 192-bit scalar, and we compared the operations to find collisions to conduct the attack outlined above. A collision was determined to exist using the Euclidean distance and the correlation as a distinguisher, and the results of these analyses are shown in Table 1. We show the expected time complexity of the resulting discrete logarithm problem as given in Section 2.3. We also show the expected time complexity in the mean case, i.e. the time complexity one could expect if several traces are taken and analysed separately.

Table 1. The expected number of wrong key bits produced from 50 instances of the attack.

Matching Method	E (#Errors)	σ	E (time complexity)	E (time complexity for mean case)
Euclidean distance	1.06	1.43	2^{26}	2^{15}
Correlation Coefficient	0.36	0.66	2^{16}	2^{13}

4.2 Scalar Multiplication with Dummy Operations

As described in Section 3, the first countermeasure proposed to prevent the identification of group operations was proposed by Coron [11]. In the additive group formed over an elliptic curve the resulting algorithm is shown in Algorithm 4. Witteman et al. [10] 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.

Lemma 3. *In one loop of Algorithm 4, 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 attacker knows the ℓ 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 1 and changes when it is equal to 0. □

Algorithm 4: Coron's Dummy Method [11]

Input: $\boldsymbol{P}$ a point on elliptic curve $\mathcal{E}$, an n-bit scalar $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Output: $\boldsymbol{Q} = k\,\boldsymbol{P}$

```
R_0 ← P ; R_1 ← P ;
for i ← n − 2 down to 0 do
    R_0 ← 2 R_0 ;
    b ← 1 − k_i ;
    R_b ← R_0 + P ;
end
return R_0
```

Given Lemma 3, an attacker can make a comparison between two power consumption traces taken while computing these operations to determine if the same value was used. Witteman et al. used a set of traces where each execution used the same exponent to ensure that the correlation coefficient was a good distinguisher, and demonstrated that this attack works on an ASIC implementation of RSA. However, their attack does require numerous acquisitions to be taken to reduce noise to the point where an attack will function.

The attack described in the previous section only needs to consider collision between point additions. That is, the power consumption traces corresponding to different operations can be superposed. However, when computing a scalar multiplication one would use a different algorithm to compute and addition and a doubling operation (as described in Section 2.2).

In applying the attack defined by Witteman et al. is not straightforward where the algorithms to be compared are not the same, since one cannot directly compare two operations. That is, we wish to observe collisions between values manipulated in adjacent additions and doubling operations which use different algorithms to compute their respective group operations. However, one can compare the field multiplications in the same way we describe above. However, given that the operations required to compute the group operation are not the same we compare the correlation between key estimates produced by conducting an attack comparing every multiplication in one operation with every multiplication in the other. The correlation coefficients are not as clear as those given above (see Figure 5). This means that we are obliged to select the points with the largest correlation using an arbitrary threshold as no clear distinction is given.

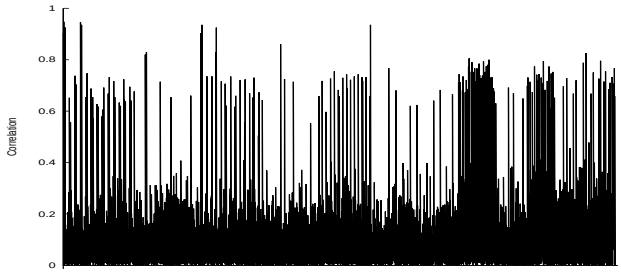


Fig. 5. Finding useful multiplications between an addition and a doubling operation.

Again, we conducted a collision attack on 50 power consumption traces while the target ARM microprocessor was computing a scalar multiplication with a 192-bit scalar. Again we used the Euclidean distance and the correlation coefficient as distinguishers. The results of these analyses are shown in Table 2. We show the expected time complexity of the resulting discrete logarithm problem as given in Section 2.3. As above, we also show the expected time complexity in the mean case.

Table 2. The expected number of wrong key bits produced from 50 instances of the attack.

Matching Method	E (#Errors)	σ	E (time complexity)	E (time complexity for mean case)
Euclidean distance	6.66	7.51	2^{66}	2^{32}
Correlation Coefficient	3.38	5.00	2^{54}	2^{22}

4.3 Montgomery Ladder

Another variant of addition chain-based exponentiation is the Montgomery powering ladder [14] as shown in Algorithm 5. The combination of $\boldsymbol{R}_0$ and $\boldsymbol{R}_1$ will produce an addition chain containing an addition chain similar to that produced by the binary exponentiation algorithm. Indeed, the addition chain created by a binary exponentiation algorithm will contain a subset of the elements of the addition chain created by the Montgomery powering ladder.

Algorithm 5: Montgomery Powering Ladder

Input: $\boldsymbol{P}$ a point on elliptic curve $\mathcal{E}$, an n-bit scalar $k = (k_{n-1}, k_{n-2}, \ldots, k_0)_2$
Output: $\boldsymbol{Q} = k\,\boldsymbol{P}$

1 $\boldsymbol{R}_0 \leftarrow \boldsymbol{\mathcal{O}}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
2 **for** $i = n-1$ **down to** 0 **do**
3 $\quad \boldsymbol{R}_{\neg k_i} \leftarrow \boldsymbol{R}_{k_i} + \boldsymbol{R}_{\neg k_i}$;
4 $\quad \boldsymbol{R}_{k_i} \leftarrow 2\,\boldsymbol{R}_{k_i}$;
5 **end**
6 **return** $\boldsymbol{R}_0$

A method of attacking the Montgomery powering ladder is described by Kim et al. [8]. However, it requires that Line 3 in Algorithm 5 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. 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. If an attacker applies this attack to the implementation of a scalar multiplication the result is similar to that described in Section 4.2.

This attack cannot be applied to Algorithm 5 as described since Line 3 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 4. *In Algorithm 5, if the bits treated in two consecutive loops are the same then the output of the operation in Line 4 in the first loop will be the input to the operation in Line 4 in the second loop.*

Proof. We assume that an attacker 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

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

Let $k' = 2\,u$, 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, let $k' = 2\,u' + 1$, 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 5. *In Algorithm 5, if the bits treated in two consecutive loops are different then the output of the operation in Line 3 in the first loop will be the input to the operation in Line 4 in the second loop.*

Proof. We assume that an attacker 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

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

Let $k' = 2\,u+1$, for some $u \in \mathbb{Z}_{>0}$, then the bit $k_{n-\ell} = 1$ and $\boldsymbol{R}_0 = k'\,\boldsymbol{P}$ is computed by the addition $u\,\boldsymbol{P} + (u+1)\,\boldsymbol{P}$ and then, if the next bit is also equal to zero, the following loop will compute 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, Let $k' = 2\,u'$, for some $u' \in \mathbb{Z}_{>0}$, then the bit $k_{n-\ell} = 0$ and $\boldsymbol{R}_1 = (k'+1)\,\boldsymbol{P}$ is computed by the addition $u\,\boldsymbol{P} + (u+1)\,\boldsymbol{P}$ and, then if the next bit is also equal to one, the following loop will compute

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

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 4 and Lemma 5, the attack described in Section 4.2 cannot be used to directly compare operations since an addition does not consist of the same operations as a doubling operation. Moreover, one cannot compare field operations directly since one wishes to compare the input of one operation with the output of another operation.

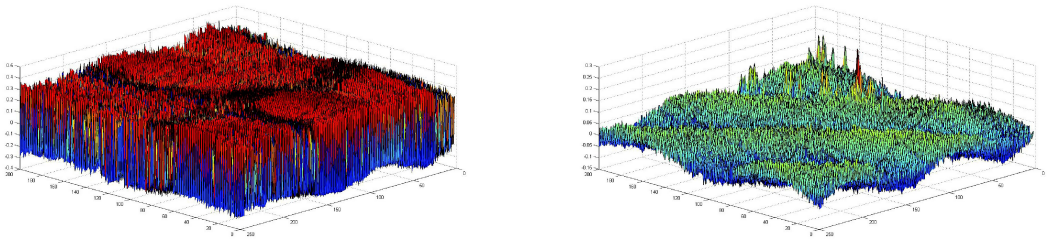
In the remainder of this section we describe a collision attack based on techniques used to second-order differential power analysis. We wish to compare points in a power consumption trace where correlated data is manipulated by different operations in a microprocessor. In order to make such comparisons one needs to change each acquired trace by subtracting a mean trace, generated from a set of acquisitions, so that the acquired traces just contain noise and variations caused by the manipulated data. This is a required strategy for conducting second-order differential power analysis. Moreover, the attack described below is computationally intensive and all the acquisitions used were compressed by only keeping the maximum point per clock cycle [26].

The first step of our attack is to find the corresponding points of time in two traces which we wish to compare. For example, when using the characteristic defined in Lemma 5, we have to find the points at corresponding to the the output of an addition so that they can be compared tot he input of an a doubling operation. To find these points, we collected $N = 50$ power consumption traces while our target microprocessor was computing a scalar multiplication $S^{(i)}$ $(1 \le i \le N)$ each with a different scalar. For each $S^{(i)}$ we denote the 192 addition and doubling operations as $A_j^{(i)}$ and $D_j^{(i)}$ respectively where j refers to the loop in the algorithm $(1 \le j \le n$, where $n = 192)$. That is, a given trace $S^{(i)}$ consists of smaller traces as follows:

$$S^{(i)} : A_1^{(i)}; D_1^{(i)}; A_2^{(i)}; D_2^{(i)}; A_3^{(i)}; D_3^{(i)}; \ldots; A_{n-1}^{(i)}; D_{n-1}^{(i)}; A_n^{(i)}; D_n^{(i)}$$

By the first collision characteristic, two signals $A_j^{(i)}$ and $D_{j+1}^{(i)}$ include power consumed by operating same value when $k_{\ell+1} \neq k_\ell$, for some $\ell \in \{1, \ldots, n\}$. Even without knowing the values of the exponents, one can say that half of the comparisons between $A_j^{(i)}$ and $D_{j+1}^{(i)}$ $(1 \le j \le n-3)$ will demonstrate a collision. However, one cannot know which pairs are of interest.

To find the points in these traces that are of interest we compute a r-by-s matrix containing the pairwise correlation coefficient between each pair of columns in the $N(n-3)$-by-r and $N(n-3)$-by-s matrices X and Y. We define X and Y as the sets of power consumption traces corresponding to acquisitions taken during the computation of addition



(a) Small part of the matrix containing pairwise correlation coefficient between two signals

(b) Small part of the differential matrix

Fig. 6. Sample correlation matrices from our analyses.

and doubling operations of length r and s respectively. However, such a matrix is extremely noisy, a sample of this matrix is shown in Figure 6(a).

To reduce the noise level so that one can determine which points one can use to detect collisions, we generated a similar matrix where there is no possible collision between each A_j and D_j used. We subtracted this noise-level matrix from the original one to produce a matrix with a stronger signal. A sample of the resulting differential matrix is shown in Figure 6(b). This procedure for finding the corresponding points of time with regards to the first collision characteristic is schematically depicted in Figure 7.

From the differential matrix we chose all the points where the correlation coefficient was superior to a threshold of 0.15. With the same threshold, the procedure for finding the interesting combinations of points was repeated for the characteristic defined in Lemma 4. This provided two sets of points that can be compared to detect the collision characteristics described above. Using these points the characteristics defined in Lemma 4 and Lemma 5 can be used to determine if two consecutive bits of the exponent are the same.

Given a guess for the exponent used in the Montgomery Ladder, one cannot apply the algorithm given in Section 2.3. This is because each time one wishes to flip a bit all the bits that are less significant also need to be flipped. One will therefore need to store approximately twice as many entries to account for this difference. However, the time complexity of the attack will still be $\mathcal{O}(m\,t\,\binom{m/2}{t/2})$.

As previously, we conducted a collision attack on 50 power consumption traces while the target ARM microprocessor was computing a scalar multiplication with a 192-bit scalar. We used the Euclidean distance and the correlation coefficient as distinguishers. The results of these analyses are shown in Table 3. We show the expected time complexity of the resulting discrete logarithm problem as given in Section 2.3. As above, we also show the expected time complexity in the mean case.

Table 3. The expected number of wrong key bits produced from 50 instances of the attack.

Matching Method	E (#Errors)	σ	E (time complexity)	E (time complexity for mean case)
Euclidean distance	12.38	3.61	2^{57}	2^{44}
Correlation Coefficient	6.62	2.32	2^{39}	2^{32}

5 Countermeasures

In the above we show that an attacker can determine if two variables 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 the order that they are used should not reveal any information to an attacker. That is,

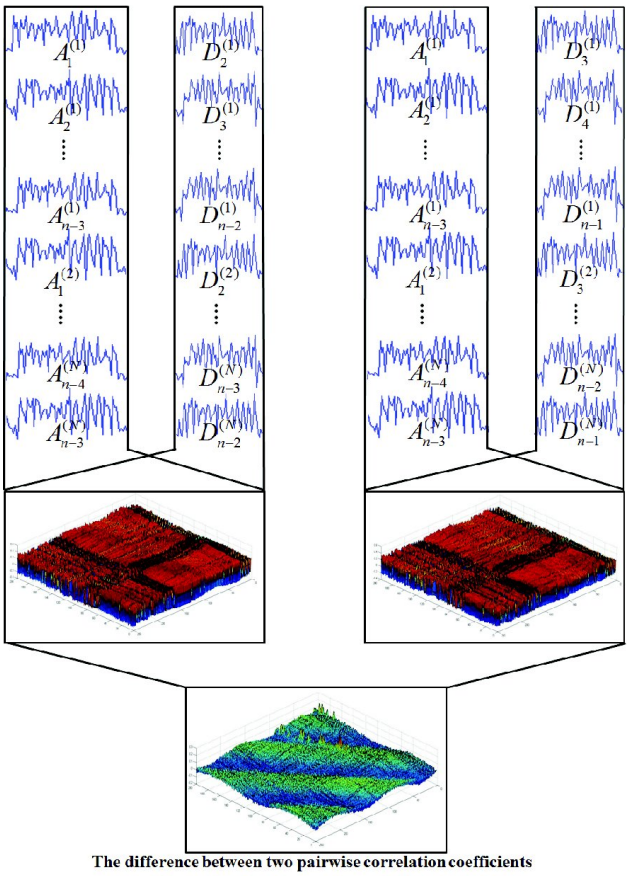


Fig. 7. Finding the corresponding points of time with regards to the first collision characteristic

an attacker cannot find meaningful collisions based on comparing the output of one operation with the input or output of another operation. That is, we define the following property:

Definition 4. *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 6. *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 $\mathbf{A}$ 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 $\mathbf{A}$ to compute $f(n)$. If f is totally regular this would imply that $\mathbf{A}$ would remain unchanged for all possible exponents that have the same bit length as λ. However, by Definition 1 the last element of $\mathbf{A}$ 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. In the following we discuss some countermeasures that can be used to increase the noise that would be introduced in the analyses given above. We discuss some of the commonly cited countermeasure briefly below. The reader is referred to Fan and Verbauwhede [27] for a thorough treatment of this topic.

Random Curve Isomorphisms

A given $\boldsymbol{P}$ on elliptic curve $\mathcal{E}$ can be randomized by computing $\boldsymbol{P}^* \leftarrow \psi(\boldsymbol{P})$ on $\mathcal{E}^* \leftarrow \psi(\mathcal{E})$ for a random curve isomorphism ψ. Then $\boldsymbol{Q} = [k]\,\boldsymbol{P}$ can be computed by calculating $\boldsymbol{Q} = \psi^{-1}([k]\,\boldsymbol{P}^*)$ [28]. A given isomorphism ψ will be fixed for a single execution so the attacks described in this paper apply with no modification. However, if the choice of isomorphism affects the execution time it may make characterising an implementation more difficult.

Randomized Algorithms

A variety of group exponentiation algorithms have been proposed where a randomised sequence of operations is executed, such as described by Oswald and Aigner [29], or storing and treating exponent digits [30]. As with random curve isomorphisms, the structure of a group exponentiation algorithm becomes fixed for one execution and will not prevent collision-based attacks.

Point Blinding

As described above, an affine point $\boldsymbol{P} = (x, y)$ can, for example, be represented as a homogeneous projective point $(\theta\, x, \theta\, y, \theta)$ for all $\theta \in \mathbb{F}_p \setminus \{0\}$ [24]. Before computing a scalar multiplication one can choose some random $\theta \in \mathbb{F}_p \setminus \{0\}$ and use the point $(\theta\, x, \theta\, y, \theta)$ as a random representation of the affine point (x, y) [11].

The attacks described in Section 4 assume that the input to an exponentiation algorithm is not known so this countermeasure cannot be applied just once at the beginning of the exponentiation as described by Coron [11]. In order to prevent collision-based attacks one would need to replace the output point of any operation with a randomised point. While this would not prevent all information leakage, it will minimise the information available to an attacker. In all the attacks described in Section 4 we compare power consumption traces taken during the computation of group operations. Randomising a point can be done far quicker than computing a group operation. For example, to randomise the representation of a homogenous projective point one could multiply each coordinate by a small random value. Comparing a trace taken while a point is being randomised will be harder to compare to a trace taken while a group operation is being computed.

6 Conclusion

We describe attacks based on collisions of variables manipulated in group operations extending attacks described in the literature. Furthermore, we demonstrate that our attacks result in an instance of the discrete logarithm problem that can be solved in a reasonable amount of time in all cases. The advantage of our attacks over previous work is that the attacks can be applied to a single trace and do not require any knowledge of the input to the exponentiation algorithm. However, an attacker may have to apply the attack to more than one trace to succeed. 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.

Our current work on consists of applying the attacks defined above to hardware implementations of a scalar multiplications. We are also evaluating the impact of the countermeasures described in Section 5 on the time complexity of our attacks.

Acknowledgements

The authors would like to thank Georg Fuchsbauer, Seokhie Hong and Elisabeth Oswald for their helpful comments. 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. 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. 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
8. 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
9. 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
10. 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
11. 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
12. Knuth, D.E.: The Art of Computer Programming. 2nd edn. Volume 2 / Seminumerical Algorithms. Addison-Wesley (1981)
13. 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
14. 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
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. Kocher, P.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Koblitz, N., ed.: CRYPTO '96. Volume 1109 of LNCS., Springer (1996) 104–113
18. Stinson, D.: Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem. Mathematics of Computation **71**(237) (2002) 379–391
19. 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
20. 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
21. Yen, S.M., Lien, W.C., Moon, S.J., Ha, J.: Power analysis by exploiting chosen message and internal collisions — Vulnerability of checking mechanism for RSA-decryption. In Dawson, E., Vaudenay, S., eds.: Mycrypt 2005. Volume 3715 of LNCS., Springer (2005) 183–195
22. 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
23. 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
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. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks — Revealing the Secrets of Smart Cards. Springer (2007)
27. 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
28. Joye, M., Tymen, C.: Protections against differential analysis for elliptic curve cryptography: An algebraic approach. In Ç. K. Koç, Naccache, D., Paar, C., eds.: CHES 2001. Volume 2162 of LNCS., Springer (2001) 377–390

29. Oswald, E., Aigner, M.: Randomized addition-subtraction chains as a countermeasure against power attacks. In Ç. K. Koç, Naccache, D., Paar, C., eds.: CHES 2001. Volume 2162 of LNCS., Springer (2001) 39–50
30. Tunstall, M.: Random order m-ary exponentiation. In Boyd, C., Nieto, J.M.G., eds.: ACISP 2009. Volume 5594 of LNCS., Springer (2009) 437–451