Dismantling real-world ECC with Horizontal and Vertical Template Attacks

Margaux Dugardin[1,2], Louiza Papachristodoulou[3], Zakaria Najm[1], Lejla Batina[3], Jean-Luc Danger[1], Sylvain Guilley[1], Jean-Christophe Courrège[2], and Carine Therond[2] *

[1] TELECOM ParisTech, COMELEC, 46 rue Barrault, 75014 Paris, France
firstname.lastname@telecom-paristech.fr
[2] Thales Communications & Security, CESTI, 3 avenue de l'Europe, 31000 Toulouse, France
[3] Radboud University Nijmegen, Digital Security Group, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
lejla@cs.ru.nl, louiza@cryptologio.org

Abstract. Recent side-channel attacks on elliptic curve algorithms have shown that the security of these cryptosystems is a matter of serious concern. The development of techniques in the area of Template Attacks makes it feasible to extract a 256-bit secret key with only 257 traces. This paper enhances the applicability of this attack by exploiting both the horizontal leakage of the carry propagation during the finite field multiplication, and the vertical leakage of the input data. As a further contribution, our method provides detection and auto-correction of possible errors that may occur during the key recovery. These enhancements come at the cost of extra traces, while still providing a practical attack. Finally, we show that the elliptic curve technology developed in PolarSSL running on a ARM STM32F4 platform is completely vulnerable, when used without any modifications or countermeasures.
Keywords: Side-channel analysis, horizontal leakage, vertical leakage, scalar multiplication, Brainpool curves, NIST curves, PolarSSL.

1 Introduction

Implementing security protocols for embedded devices is a constant challenge for the cryptographic community, due to the development of new and powerful side-channel attack techniques. By measuring the power consumption or the electromagnetic emanation of a device during the execution of a cryptographic algorithm, it is possible to derive secret data from a single or multiple traces.

Within the area of side-channel attacks there exist different methods of analysis; either by using a single trace (Simple Analysis) or a large number of traces (Differential and Correlation Analysis) from the target device [20, 6]. Template

* This work was supported in part by the Technology Foundation (STW) through project 12624 – SIDES, the ICT COST action IC1204 TRUDEVICE and the COST action IC1306 Cryptography for Secure Digital Interaction, Date: 2015-10-14

Attacks belong to yet another kind of attacks and are considered to be the most powerful method from the information-theoretic point of view, since they take advantage of most information available in a side-channel observation [8]. The attacker is assumed to have one or limited number of side-channel measurements from the target device, i.e. power, EM traces or timing, but he has access to a similar device, on which he can simulate the computations of the target (template building phase). Rechberger and Oswald presented the first practical template attack on RC4 running on an 8-bit micro-controller in [27].

Most notably, the work of De Mulder et al. [25] showed the first practical attack using electromagnetic emanation of Elliptic Curve Cryptosystems (ECC) on an FPGA implementation. The main idea of the attack presented in this paper is a collision attack exploiting the doubling operation during an ECC computation. Collision attacks exploit the leakage between two portions of the same or different traces, when the same intermediate values are reused. In [4, 10, 9, 14], these attacks are presented as theoretical *horizontal* attacks using collisions. Our work is a practical horizontal attack. The idea of attacking the doubling operation in the elliptic curve setting was originally proposed by Fouque and Valette in [13]. Their "Doubling Attack" is based on the fact that similar intermediate values may be manipulated when working with points $\mathcal{P}$ and $2\mathcal{P}$. However, in most cases, the intermediate values during ECC scalar multiplications are different than the input point. The most efficient result in practical template attacks on ECC is the Online Template Attack (OTA), presented in [3], which requires one full target trace and one template trace per key-bit. With 256 templates, Batina et al. retrieve a 256-bit key on the twisted Edwards curve used for the Ed25519 signature scheme [15].

Our contribution. PolarSSL (recently bought by ARM and renamed to *mbed TLS* [21]) is an open-source library designed initially for servers and PCs, but easy to adapt to embedded environments, like smart-phones. We demonstrate an attack on PolarSSL with Brainpool BP256r1 and NIST SecP256r1 curves running on an ARM Cortex-M4 micro-controller on a STM32F4 platform [23]. Our work shows that the countermeasure of the input randomization implemented in PolarSSL must be activated on embedded platforms.

For the demonstration of our attack, we extend the idea of Online Template Attacks (OTA) by Batina et al. presented in [3]. These authors used one full target trace and one template trace to determine the correct key bit. However, this method requires an identification phase, in order to compute the threshold between matching and non-matching templates. In our case, there are two different leakage models, a horizontal and a vertical one. Therefore, a threshold as described in [3] can not be established. We propose a more generic method to distinguish the matching templates.

The horizontal leakage in PolarSSL is a consequence of the software implementation during multiplication of large numbers (256-bit field elements). In most cases, the multiplication of large numbers leaks due to the potential propagation of carry. This carry occurs during the register addition between two words

(defined by the length of register). We observed that in OpenSSL (a widely used open-source library) the different propagation of carry leaks in the same way as in PolarSSL, making this library vulnerable not only to our attack, but also to more trivial timing attacks. The timing side-channel leakage due to the different propagation of carry can be eliminated by using a dummy operation, such as addition by zero. But in side-analysis an addition by zero can be detected using vertical leakage. Therefore, this method may create a constant time implementation, but it is still not really efficient to avoid the problem of the propagation of carry.

The vertical leakage that we exploited comes from the Hamming weight of the value stored in the register or the Hamming distance between two values stored in the same register. Despite the fact that the levels of noise (from the USB power supply and the general purpose input/output slots) on the STM32F4 platform are high, the vertical leakage can still be exploited to retrieve the scalar bits.

The advantage of our adaptive template attack over the original OTA is the fact that it detects and corrects errors. Making one assumption for each key bit and deciding according to the established threshold if this bit is the correct one, does not always give the correct result. In some instances of our attack, the templates obtained for a "0" key-bit assumption was very similar to the template made for the assumption that the key-bit is 1. To increase the success rate of our attack and to determine wrong assumptions, we decided to obtain two template traces for each key-bit. The choice to use both assumptions to create template traces allows to detect and to adjust any possible error to get back the whole scalar.

Organization of the paper. This paper is organized as follows: We describe the elliptic curves and the scalar multiplication algorithms used for our attack in Section 2. Section 3 gives an overview of the OTA methodology with vertical and horizontal leakage. Section 4 presents our practical adaptive template attack on Brainpool and NIST curves on a STM32F4 micro-controller and the error correction technique used to improve OTA. Section 5 proposes a discussion about the efficiency of certain countermeasures against our attack. Finally, Section 6 summarizes our results and concludes the paper.

2 Mathematic background

2.1 Preliminaries on Elliptic Curves

In 1985, Miller [24] and Koblitz [19] introduced the use of elliptic curves for asymmetric cryptography. The main advantage of using Elliptic Curve Cryptography (ECC) over RSA is the memory and the length of the computations; two important factors for embedded devices. The curves defined over a 256-bit field provide security level of 128-bits, which is equivalent to a 2048-bit RSA key.

An elliptic curve $\mathcal{E}$ over $\mathbb{F}_p$ can be defined in terms of solutions to the reduced Weierstrass equation $y^2 = x^3 + ax + b$ over $\mathbb{F}_p$. The pairs (x, y) that verify the previous equation represent the affine coordinate of a point over the curve $\mathcal{E}$. For our experiments, we used the Brainpool curve BP256r1 recommended by BSI [7] (noted BP) and the NIST curve SecP256r1 recommended by the NIST standard [26]. These curves are defined over a 256-bit field and have security level of 128 bits (see [16] for more details).

2.2 Scalar Multiplication Algorithm

The scalar multiplication is the main operation in cryptographic protocols using ECC, such as ECDSA signatures [1] or the Diffie-Hellman key-exchange protocol (ECDH) [2]. The scalar multiplication is an expensive operation that a designer wants to optimize, yet at the same time a sensitive operation, because it manipulates the secret key or it allows to recover the secret key from a secure component. There are a lot of scalar multiplication algorithms used for efficiency and/or resistance against side-channel attacks. In this paper, we perform an attack against the binary left-to-right double-and-add-always algorithm (see in [12, 17]), which is considered to be resistant against simple power analysis (SPA). Our attack applies to other regular algorithms as well, similarly to the original OTA [3].
The double-and-add-always algorithm takes as input a point $\mathcal{P} = (x_P, y_P)$ in affine coordinates and the scalar k. For our experiments the scalar is 256-bit long. For every iteration the computation block performs a doubling operation and an addition with $\mathcal{P}$. The output is the result of the scalar multiplication $[k]\mathcal{P}$ depending on the current bit of k_i at the specific iteration.

2.3 Scalar multiplication module of PolarSSL

PolarSSL is an open-source cryptographic library [21] recently acquired by ARM. The source code is nicely decomposed into modular blocks and it can be used in embedded devices. Our implementation is a modified version of PolarSSL 1.3.7. PolarSSL contains C and assembly code to speed up the computations over the finite field. For ECC operations, it uses the module `ecp`. To be more efficient the main functions used are doubling in Jacobian coordinates (DBL) and mixed addition [11] between a point in Jacobian and a point in affine coordinates (ADD). The cost of these operations is explained and detailed by Bernstein, Lange et al. in [5]. PolarSSL is intended to be used in embedded systems which include hardware multiplier, like smart-phones. Hence it relies on two steps: multiplication, then modular reduction.

3 Attack Description

3.1 The main idea of Online Template Attack

The *Online Template Attack*(OTA) is an adaptive template attack. The main difference with the classical template attacks as described in [27] is the absence of the building phase. The attack consists of two phases:

1. The attacker first obtains a *target trace* from the target device.
2. For each key bit, he obtains *template traces* for $[m]\mathcal{P}$, the m is chosen according to the algorithm used in the target device, and the first $m-1$ bits found by the attacker. He decides which key bit has the highest probability by matching the relevant template trace to the target trace.

For more details, see the App. A or [3]. The main difference between our attack and the attack described by Batina et al. [3] is that we use two template traces to retrieve one key bit e.g. $2\mathcal{P}$ and $3\mathcal{P}$ for the first bit. By using more template traces, we can detect and correct an error to increase the success rate of the attack.

3.2 Horizontal leakage due to propagation of carry

In case the whole doubling operation is used to construct templates, it is not possible to achieve high similarity between our templates and the target, mainly due to the noise and the non-constant time implementation. As explained later in Sec. 4.3, we can not use the intermediate values (in Jacobian coordinates) as input point (in affine) for the templates. However, by focusing on the operations in the first doubling of the double-and-add-always algorithm to construct the template traces, we achieve more accurate results. For the template pattern, we need only the pattern of the first finite-field multiplication in the doubling. In our implementation (Alg. 13 in [28]), the first operations during the doubling of point $\mathcal{P} = (X, Y, Z)$ are the following:

$$\begin{aligned} D_1 &\leftarrow X \times X \mod p \\ D_2 &\leftarrow Y \times Y \mod p \\ &\vdots \end{aligned} \tag{1}$$

In PolarSSL a multiplication between two elements in the finite field is computed as described in [22]. The result of the multiplication is stored in a 512-bit element, called *"multiplication-before-reduction"*; then the result is reduced modulo p (the characteristic of the finite field). For the curves defined in Sec. 2, one element in the finite field is 256-bit long. The micro-controller is Cortex-M4 (see Sec. 4.1 for more details). Therefore, one element corresponds to 8 words of 32-bits.

Let A and B be two 256-bit elements in the finite field. Then, A (resp. B) can be written as 8 words A_i for all $i \in \{0, 1, ..., 7\}$ (resp. B_i) of 32-bits. A_0 is the least significant word (LSW) of A and A_7 is the most significant word

Algorithm 1: Multiplication in PolarSSL

Require: A and $B_7..B_0$ two elements of 256-bits long.
Ensure: $X = A \times B$
1: $X \leftarrow 0$
2: **for** i from 7 down to 0 **do**
3: $(C, X_{i+7}, X_{i+6}, .., X_i) \leftarrow (X_{i+7}, .., X_i) + A \times B_i$
4: $j \leftarrow i + 8$
5: **repeat**
6: $(C, X_j) \leftarrow X_j + C$
7: $j \leftarrow j + 1$
8: **until** $C \neq 0$
9: **end for**
10: return X

(MSW) of A. Let X be the result of the multiplication $A \times B$ before reduction; X can be represented by 16 words of 32-bits ($X_{15}X_{14}...X_0$). The Alg. 1 shows how multiplication is performed in PolarSSL. The result $A \times B_i$ is stored in eight 32-bit words and there is a potential carry C, which needs to be stored separately (see step 3). This potential overflow creates a significant pattern that can be distinguished from its high amplitude when $C = 1$; this pattern is the propagation of carry as depicted in Fig. 1. The leakage due to the propagation of

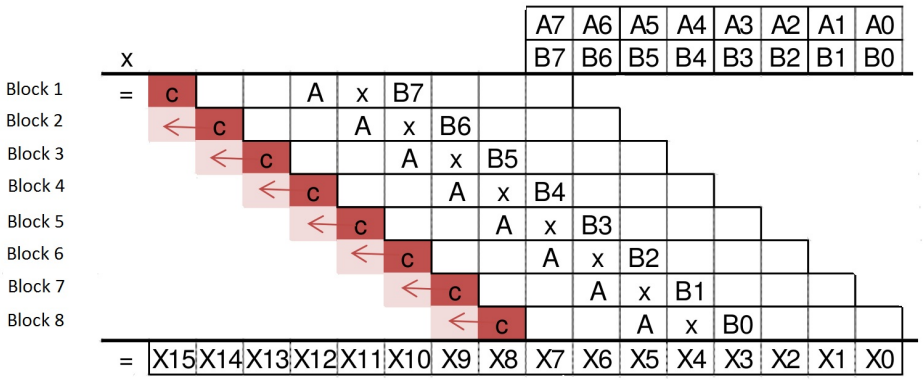

Fig. 1. Propagation of carry during multiplication in the field

carry depends on the MSW of the input data A_7. For the BP curve $\max\{A_7 | A \in \mathbb{F}_p\} =$ `0xA9FB57DA` and the probability of having a propagation of carry is close to $p = 0.17$. For the NIST curve, $\max\{A_7 | A \in \mathbb{F}_p\}$ equals $2^{32} - 1$, so this probability is close to $p = 0.25$. The full proof of this computation of probability is described in App. B. As shown in Fig. 1, we can have 7 propagations during the

multiplication, but we can not detect the last propagation. So, the probability to have two templates with the same propagation of carry, denoted by $\mathcal{P}(C)$, is:

$$\mathcal{P}(C) = \sum_{i=0}^{6} \binom{6}{i} p^{2i}(1-p)^{2(6-i)} \tag{2}$$

where p is the probability to have an internal propagation of carry. For the BP curve, the probability to have horizontal leakage is 0.86 using $p = 0.17$. For the NIST curve, the probability to have horizontal leakage is 0.95 using $p = 0.25$.

However, it is more interesting from the OTA point of view to find out when a difference in the propagation of carry occurs between the target and template traces. This is the only part of PolarSSL that is non-constant time and we take advantage of this timing difference, every time it occurs. In this case, there is an obvious horizontal leakage between the target and the template traces, as depicted in the Fig. 2.

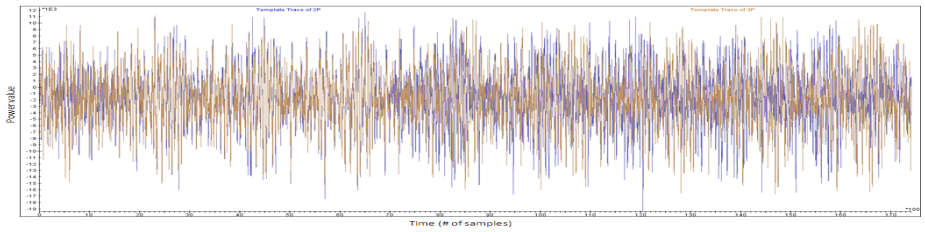

Fig. 2. Squaring of two random data with different propagation of carry

3.3 Vertical leakage due to signal amplitude

In constant time executions of our implementation, there is no difference in the propagation of carry and the template traces are synchronized with the target trace. In those cases, we observe only a vertical leakage due to the amplitude of the signal and the same method as described in [3] can be used. To observe this leakage, we use the pattern matching technique, with the Pearson correlation computation.

4 Detailed Phases of the Attack in Practice

4.1 Acquisition Setup

The target device is an STM32F4 micro-controller, which contains an ARM Cortex-M4 processor running at its maximum frequency (168MHz). We imported the assembly code originally included in PolarSSL to ARM Cortex-M4 and implemented the double-and-add-always procedure as described in [12, 17]. For the

acquisition, we used a 54855 Infiniium Agilent oscilloscope and a Langer EMV-TECHNIK RF-U5-2 near field probe. The sampling frequency is 1GSa/s with 50MHz hardware input low-pass filter enabled. The position of the probe was determined to maximize the signal related to the activity of the hardware 32×32 multiplier.

For the curves defined in Sec. 2, one element in the finite field is 256-bit long. Thus, each operation over the field consists of manipulating eight processor words (8×32 bits). In our implementation, a multiplication-before-reduction consists of eight multiplications between a 256-bit element by each 32-bit words of the second element. It leads to eight easily identifiable patterns of eight blocks on EM traces. The length between two blocks can be different depending on the propagation of carry, as explained in Sec. 3.2.

4.2 Pre-proccessing Phase

The pre-processing phase starts with choosing an input point $\mathcal{P}$ and obtaining the target trace from our target device; this is depicted in Fig. 3. In this trace,

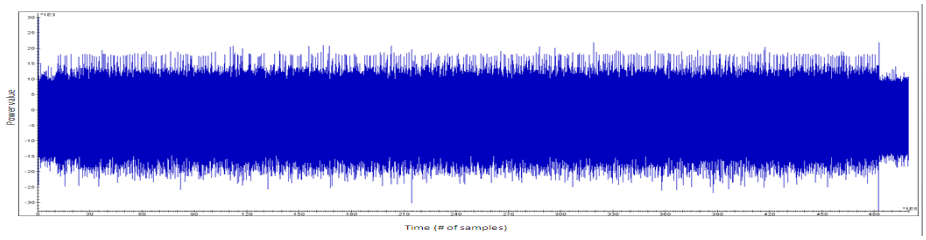

Fig. 3. EM acquisition for scalar multiplication on NIST curve with $k =$ `0xA5A5`

we need to spot the multiplication patterns, which are eight blocks of 256×32 multiplications depicted in Fig. 4. The multiplication procedure is described in Sec. 3.2.

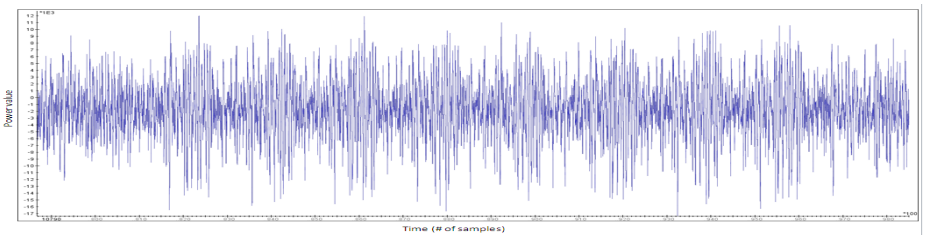

Fig. 4. Pattern of multiplication-before-reduction

When we have a clear pattern for the multiplication, we cross correlate this pattern with our target trace and we obtain the cross-correlation pattern with one peak at the position of every multiplication. Fig. 5 shows the cross correlation of the target trace with the multiplication pattern. By counting the peaks in

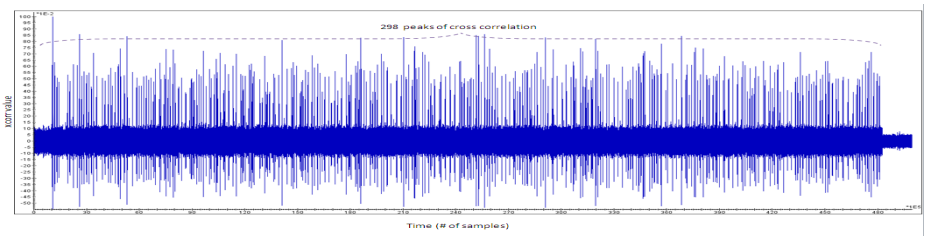

Fig. 5. Cross correlation between the pattern of the multiplication and the target trace

the cross-correlation trace, we can find the part of the computation that we are interested in. For BP, as explained in [5], the doubling consists of 10 multiplications (except for the first doubling, where there are only 7 multiplications[4]), and the mixed addition consists of 11 multiplications. For NIST, there is a particular parameter equal to $(-3 \mod p)$, so the multiplication by a in the doubling can be optimized. The doubling consists of 9 multiplications and the mixed addition of 11 multiplications[5].

In this way, we can "cut" the target trace in sections according to the loop of the scalar multiplication operation (as in Fig. 6). The first iteration of the double-

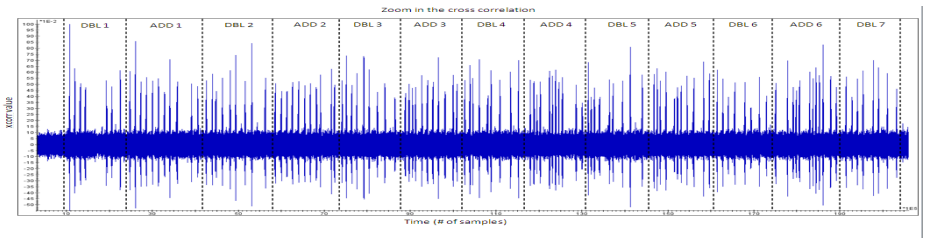

Fig. 6. The first seven iterations of the scalar multiplication algorithm on the curve

and-add always algorithm is completed after 18 peaks of cross-correlation. For the next iterations, we take into account that each doubling consists of 9 or 10 multiplications and each addition of 11. For the first bit, the interesting section on the target is the 19th multiplication. For the second bit, the interesting section

[4] Because in the beginning $Z = 1$ and we computed aZ^4 with 3 multiplications.

[5] The fact that doubling is performed faster for NIST curves, allows us to recover 7 bits of the scalar at once

is the 39th multiplication for NIST curve or 40th multiplication for BP curve. For the third bit, the interesting section on the target is the 59th multiplication for NIST curve or 61th multiplication for BP curve, and so on for all the other bits of the scalar.

As the last step of this phase, we calculate multiples of the point $\mathcal{P}$ using our PolarSSL implementation. We explain this in detail in the next section.

4.3 Template acquisition

In PolarSSL every input point is represented in affine coordinates and then converted to Jacobian coordinates. The intermediate values are represented in Jacobian coordinates. The input points to the device are given in affine coordinates. To create templates, we need to find an input point in affine corresponding to an intermediate value in Jacobian coordinates.

The *target trace* is obtained with input point $\mathcal{P} = (x_P, y_P)$ given in affine coordinates. In order to compute the intermediate values of the points $2\mathcal{P} = (X_{2P}, Y_{2P}, Z_{2P})$ and $3\mathcal{P} = (X_{3P}, Y_{3P}, Z_{3P})$ with PolarSSL, we use the formulas defined in [5]. Note that this does not correspond to the point $2\mathcal{P}$ and $3\mathcal{P}$ in affine coordinates, because $Z_{2P}, Z_{3P} \neq 1$. Therefore, we can not compare directly the templates with input point $2\mathcal{P}$ (resp. $3\mathcal{P}$), since they are not in affine form.

We create our templates with a specific input point $\mathcal{Q}_i$ such that the first field multiplication D_1 in $2\mathcal{P}$ or $3\mathcal{P}$ is the same with the one attacked on the target trace. The squaring of the X-coordinate of the intermediate value is not affected by the change of coordinates system.

The way to construct the input point for templates is more sophisticated. Let us assume that we have the input point $\mathcal{Q}_0 = (x_{Q_0}, y_{Q_0})$ in affine coordinates associated to the point value $2\mathcal{P}$ and $\mathcal{Q}_1 = (x_{Q_1}, y_{Q_1})$ corresponding to $3\mathcal{P}$. We need to analyze the squaring of X-coordinate in Jacobian coordinates. The input point $\mathcal{Q}_0 = (x_{Q_0}, y_{Q_0})$ should be a solution in $\mathbb{F}_p \times \mathbb{F}_p$ of the following system:

$$\begin{cases} x_{Q_0} = X_{2P} \\ y_{Q_0}^2 = x_{Q_0}^3 + a x_{Q_0} + b \end{cases} \tag{3}$$

with a, b the parameters of the curve as defined in [7, 26].

The number X used as input in the squaring is random, so $X^3 + aX + b$ is also random. If $x_{Q_0}^3 + a x_{Q_0} + b$ is not a square in the finite field, we can change one bit in X as proposed in [3] we get another point on the curve that satisfies Eq.(3).

We locate the first multiplication in the template trace corresponding to the squaring of the X-coordinate of the input point $\mathcal{Q}_0$ or $\mathcal{Q}_1$, depicted in Figs. 8, 9 respectively. With these two patterns and the target trace (Fig. 7), we can perform template matching.

4.4 Template Matching

In this section, we present how to perform template matching by making the right hypothesis on a scalar-bit. This procedure is described for the cases of

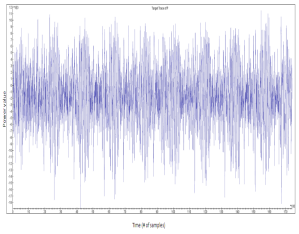

Fig. 7. Pattern of the 19^{th} multiplication in trace with input $\mathcal{P}$

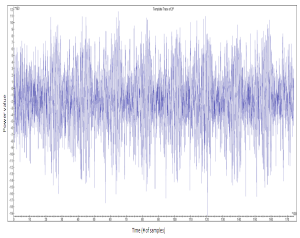

Fig. 8. Pattern of the 1^{st} multiplication in trace with input $\mathcal{Q}_0$

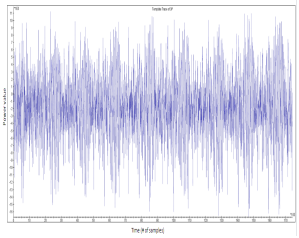

Fig. 9. Pattern of the 1^{st} multiplication in trace with input $\mathcal{Q}_1$

horizontal and vertical leakage. The probability of having a horizontal leakage corresponds to the probability of having different propagation of carry between the two templates.

Horizontal leakage When the traces are not synchronized (86% of cases in BP curve, and 95% in NIST curve), cross correlation between the multiplication pattern and the target trace is performed before template matching, in order to choose the correct part of the target trace. Then we align the template and target traces and decide what is the correct key-bit guess.

Horizontal leakage is observed when there is different propagation of carry between two multiplications 256×32 in the field. In Fig. 10 we see the misalignment of the traces due to propagation of carry.

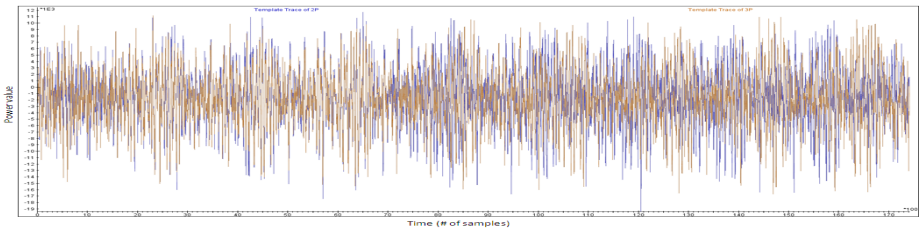

Fig. 10. Misalignment of two template traces due to propagation of carry

Vertical leakage When the implementation is executed in constant time and the template traces are synchronized with the target trace, the same method as described in [3] can be used (14% of cases in Brainpool curve and 5% in NIST curve). The propagation of carry is the same between the two templates and the target as depicted in Fig. 11; therefore, we can only observe a *vertical* leakage in our traces. In our experiments, we use the Pearson correlation coefficient $\rho(X, Y)$

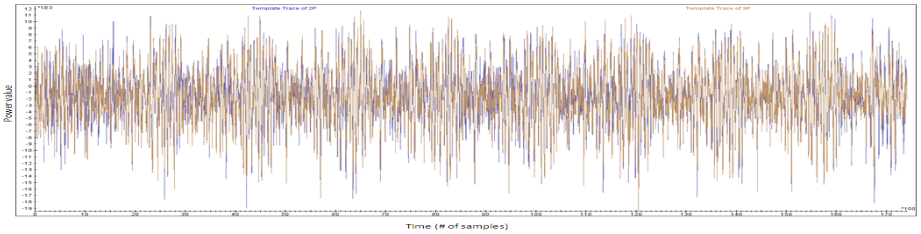

Fig. 11. Two templates with the same propagation of carry

as described in Sec. A.3 and we get a correlation of $0,81$ for the multiplication obtained from the target trace and the template trace of $2\mathcal{P}$. The same value drops to $0,78$ for the correlation of the target trace with the template trace of $3\mathcal{P}$.

4.5 Success Rate for one key-bit

In this part, we describe the method used to calculate and increase the success rate of our attack. As explained in Sec. 3.2 for the NIST curve the probability to have a different propagation of carry between the two templates is 95% and for BP 86%. The horizontal attack scenario is easy, since if two templates have different propagation of carry, then the success rate of finding this bit is 100%.

For the vertical attack scenario, the success rate depends on the input data. So, we keep only the input data, when the propagation of carry is the same for the two template traces. We perform the computation of the success rate, using random points on each curve for the target trace. We acquire 30 target traces to attack, and for the template trace, we acquire 1000 traces for each assumption when the propagation of carry is the same. We compute the two Pearson correlations between one target trace and one template by each assumption. Each bit k_i is the notation for the correct key-bit of k. If the highest Pearson correlation gives the assumption k_i, then the counter corresponding to the success of the attack increases, otherwise, the counter corresponding to the failure of the attack increases. The success rate to retrieve one key-bit in vertical leakage for the NIST curve is $76,23\%$ and for the BP curve is 69%. To conclude, the success rate to find one bit is $1 \times 0,95 + 0,76 \times 0,05 = 98,8\%$ for the NIST curve, and $1 \times 0,86 + 0,69 \times 0,14 = 95,66\%$ for the BP curve.

To increase the vertical information leakage the average of traces can be used. When the scalar is randomized, we can not perform the attack with more than one target traces. But we can still acquire more than one template traces. So, by using only one target trace with an average of a few template traces, the success rate increases as shown the Tab. 1 on the BP curve. To conclude, the success rate for BP curve is $1 \times 0,86 + 0,99 \times 0,14 = 99,86\%$

4.6 Error-correcting bit from the template traces

The advantage of our method is the possibility of detection and correction of errors. As we described in the previous section, the success rate to retrieve one bit

Number of average traces	1	10	50	100
Success Rate	69%	80,70%	91,60%	99,80%

Table 1. Different success rates according to the number of average template traces on BP curve.

using OTA is close to 99%, which means that there is a 1% probability of having an unsuccessful attack due to a wrong key-guess. For a 256-bit scalar, if an error is not detected, the success rate for the original OTA is $7,6\%$ ($0.99^{255} \simeq 0.076$), since this error will propagate and affect all the bits after the wrong guess. Therefore, it is very important to detect and correct errors before making new templates. An error can be made when both template traces have the same propagation of carry. If we want to be sure of the key-bit value, then we can automatically compute the value of this bit and the following one. For instance, if the two templates for $2\mathcal{P}$ and $3\mathcal{P}$ have the same propagation of carry, then we create templates for $4\mathcal{P}$, $5\mathcal{P}$, $6\mathcal{P}$ and $7\mathcal{P}$. We have the following four cases:

1. Let us assume that only one template has the same propagation as the target, so we can conclude for both key-bits. For example, if the template of $7\mathcal{P}$ gives the highest correlation, then the second key-bit is 1, and the third bit is 1.
2. Let us assume that two templates have the same propagation as the target, so we compute 4 templates, and start looking the next bit with the 4 template traces.
3. Let us assume that three templates have the same propagation as the target, so we compute 6 templates, and start looking the next bit with the 6 template traces.
4. Let us assume that four templates have the same propagation as the target, so we compute the 8 templates, and start looking the next bit with the 8 template traces.

For BP curve, the probability to have one template trace with the same propagation of carry with the target trace is, 70%, two template traces is 14%, 3 template traces is 3,9% , 4 template traces is 1,2%, 5 template traces is 0,4%, 6 template traces is 0,1%, the probability for more template trace with the same propagation of carry is very low. For NIST curve, the probability to have one template trace with the same propagation of carry with the target trace is, 90%, two template traces is 5,9%, 3 template traces is 0,7%, 4 template traces is 0,1%, 5 template traces is 0,01%. For both curves, this probability reduces significantly for more template traces. To conclude the number of template traces cannot increase exponentially. At the end of the attack, in order to retrieve the 256-bit scalar, we can have an uncertainty for the 2 or 3 last bits; we find the 2^2 or 2^3 scalar key by comparing with the output $\mathcal{Q} = [k]\mathcal{P}$ and in this way we can find those last bits.

5 Countermeasure

At this point, it is clear that both OTA and our adaptive template attack are very efficient methods to attack scalar the double-and-add always algorithm during the execution of ECC protocols. These methods can be easily adapted to other scalar multiplication algorithms as described in [17, 28]. For the binary algorithm Montgomery Ladder [18], we can compare the doubling operation and find the correct key bit. For the non-binary algorithm using windows, we can obtain templates for all hypotheses and make the same attack with more template traces. Since the most commonly used scalar multiplication algorithms are vulnerable to our attack, it is interesting to see which countermeasure can be applied against it. We hereby give a list of the possible countermeasures and their efficiency against our attack:

- *Randomization of the scalar.*
  The different ways of scalar randomization are :
  1. $[k]\mathcal{P} = [k - r]\mathcal{P} + [r]\mathcal{P}$, two scalar multiplications are computed $\mathcal{Q} = [k - r]\mathcal{P}$ and $\mathcal{R} = [r]\mathcal{P}$;
  2. $[k]\mathcal{P} = [k \times r^{-1}]([r]\mathcal{P})$, two scalar multiplications are computed $\mathcal{Q} = [r]\mathcal{P}$ and $\mathcal{R} = [k \times r^{-1}]\mathcal{Q}$;
  3. $[k]\mathcal{P} = [k \mod r]\mathcal{P} + [\lfloor \frac{k}{r} \rfloor]([r]\mathcal{P})$, three scalar multiplications are computed $\mathcal{Q} = [k \mod r]\mathcal{P}$, $\mathcal{R} = [r]\mathcal{P}$ and $\mathcal{S} = [\lfloor \frac{k}{r} \rfloor]\mathcal{R}$.

  For all these randomization techniques, our attack can be applied; the *target trace* requires one acquisition on the second or third scalar multiplication. This acquisition is possible using an oscilloscope with big memory depth. For the template traces, we make assumptions for each part of the scalar multiplication. We retrieve a random scalar part for each scalar multiplication part. And to retrieve the scalar, we compute the addition (randomization 1.) or the multiplication (randomization 2.) of the scalar or the both addition and multiplication (randomization 3.). So, using the scalar randomization is no efficient against this kind of attack.
- *Randomization of the point.*
  The Jacobian representation of the point can be easily randomized as the projective coordinate. For the Jacobian coordinate, the randomization consists in selecting a random r in the finite field $\mathbb{F}_p$, and compute: $(X, Y, Z) \mapsto (r^2 \times X, r^3 \times Y, r \times Z)$. In most cases, the input point is in affine coordinates, so the randomization of the point is reduced to compute: $(x, y) \mapsto (r^2 \times x, r^3 \times y, r)$. The supplementary cost of this countermeasure is 5 finite field multiplications. For comparison, the cost of one scalar multiplication using 256-bit scalar with a regular algorithm such as double-and-add-always is 5100 multiplications. Applying randomization of the input point does not allow to predict intermediate values of the calculation and prevents the construction in a deterministic way for the template traces. This countermeasure is implemented in PolarSSL, and must be activated to protect the embedding device.

– *Random field isomorphism.*
This countermeasure prevents our attack, in the same way as described in [3]. Random field isomorphisms can be performed by changing the prime of the finite field that our computations take place. If we use random primes to generate our curves, our attack will not work.

6 Conclusion

In this paper we presented a practical extension of OTA on Brainpool Br256r1 and NIST Sec256r1 curves implemented on an ARM Cortex M4 micro-controller. A modified version of OTA is applied with the Pearson correlation coefficient as distinguisher for the correct hypothesis on the key-bit. Error detection and correction of a wrong key-bit guess is possible for our modified version of the attack, mainly due to the fact that we expect to have 99.8% success rate with averaging 100 template traces.

Most of the countermeasures applicable to the original OTA attack should also work against our attack. Blinding the coordinates for every execution of the attacked algorithm is the most efficient countermeasure against OTA, though incurring with some cost for the performance of the implementation. Point blinding is also efficient against our attack, since we need to know the input point and to actually be able to choose input points for our templates. To use PolarSSL in ARM embedding device, the countermeasure of point blinding must be activated.

References

1. ANSI-X9.62. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), 1998.
2. ANSI-X9.63. Public Key Cryptography for The Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography, 1998.
3. Lejla Batina, Lukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. Online template attacks. In *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, pages 21–36, 2014.
4. A. Bauer, E. Jaulmes, E. Prouff, and J. Wild. Horizontal collision correlation attack on elliptic curves. In T. Lange, K. Lauter, and P. Lisonek, editors, *Selected Areas in Cryptography*, volume 8282 of *LNCS*, pages 553–570. Springer, 2014.
5. Daniel J. Bernstein and Tanja Lange. Explicit formulas database. http://www.hyperelliptic.org/EFD/.
6. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.
7. BSI. RFC 5639 - Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation. Technical report, Bundesamt für Sicherheit in der Informationstechnik (BSI), 2010.
8. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, pages 13–28, 2002.

9. C. Clavier, B. Feix, G. Gagnerot, C. Giraud, M. Roussellet, and V. Verneuil. ROSETTA for single trace analysis. In Steven Galbraith and Mridul Nandi, editors, *Progress in Cryptology – INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 140–155. Springer, 2012.
10. C. Clavier, B. Feix, G. Gagnerot, M. Roussellet, and V. Verneuil. Horizontal correlation analysis on exponentiation. In Miguel Soriano, Sihan Qing, and Javier Lopez, editors, *Information and Communications Security*, volume 6476 of *LNCS*, pages 46–61. Springer, 2010.
11. Henri Cohen, Atsuko Miyaji, and Takatoshi Ono. Efficient elliptic curve exponentiation using mixed coordinates. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings*, volume 1514 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 1998.
12. Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.
13. Pierre-Alain Fouque and Frédéric Valette. The doubling attack — Why upwards is better than downwards. In Colin D. Walter, Çetin K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *LNCS*, pages 269–280. Springer, 2003.
14. N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Shamir. Collision-based power analysis of modular exponentiation using chosen-message pairs. In E. Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *LNCS*, pages 15–29. Springer, 2008.
15. Michael Hutter and Peter Schwabe. NaCl on 8-bit AVR microcontrollers. In Amr Youssef and Abderrahmane Nitaj, editors, *Progress in Cryptology – AFRICACRYPT 2013*, volume 7918 of *LNCS*, pages 156–172. Springer, 2013.
16. N. Smart I. Blake, G. Seroussi. *Advances in Elliptic Curve Cryptography*, volume 317. Cambridge University Press, 1999.
17. Marc Joye. *Elliptic curve cryptosystems and Side Channel Analysis*, volume 4, pages 17–21. ST J. Syst. Res., 2003.
18. Marc Joye and Sung-Ming Yen. The montgomery powering ladder. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2002.
19. Neal Koblitz. *Elliptic curve cryptosystems*, volume 48, pages 203–209. Mathematics of Computation, 1987.
20. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.
21. ARM mbed. Polarssl version 1.3.7. https://tls.mbed.org/.
22. Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
23. ST Microelectronics. RM0090 Reference Manual. DocID018909 Rev 8, 2014.
24. Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August*

18-22, 1985, Proceedings, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.
25. Elke De Mulder, Pieter Buysschaert, Sıddıka Berna Örs, Peter Delmotte, Bart Preneel, Guy Vandenbosch, and Ingrid Verbauwhede. Electromagnetic Analysis Attack on an FPGA Implementation of an Elliptic Curve Cryptosystem. In *IEEE International Conference on Computer as a tool*, pages 1879–1882, November 2005. Belgrade, Serbia & Montenegro. DOI: 10.1109/EURCON.2005.1630348, http://www.sps.ele.tue.nl/members/m.j.bastiaans/spc/demulder.pdf.
26. NIST. FIPS publication 186-4 - Digital Signature standard (DSS). Technical report, National Institute of Standards and Technology (NIST), July 2013.
27. Christian Rechberger and Maria Elisabeth Oswald. Practical template attacks. In Chae Hoon Lim and Moti Yung, editors, *Information Security Applications*, volume 3325 of *Lecture Notes in Computer Science*, pages 440 – 456. Springer, 2004.
28. Matthieu Rivain. Fast and regular algorithms for scalar multiplication over elliptic curves. *IACR Cryptology ePrint Archive*, 2011:338, 2011.

A Description for Online Template Attack

A.1 Attack model for OTA

Online Template Attacks (OTA), introduced in [3], is an adaptive template attack technique, which can be used to recover the secret scalar in a scalar multiplication algorithm. The main assumption in the OTA attacker model is in his ability to choose an input point to the scalar multiplication algorithm, in order to generate template traces. As it is demonstrated in the original paper, OTA works with one *target trace* from the device under attack and one *template trace* per key-bit obtained from the attacker's device that runs the same implementation. Performing OTA in practice requires the following assumptions to be made regarding the attacker:

- The attacker knows the input $\mathcal{P}$ of the target device.
- He knows the implementation of the scalar multiplication algorithm and he is able to compute the intermediate values.
- He can choose the input points on a device similar to the target device.

Furthermore, we work with the following assumptions related to the device:

- The scalar can be randomized.
- The intermediate values are deterministic.

The OTA is then performed as follows:

1. The attacker first obtains a target trace with input point $\mathcal{P}$ from the target device.
2. He obtains template traces with input points $[m]\mathcal{P},\ m \in \mathbb{Z}$ for multiples of the point $\mathcal{P}$, e.g. $2\mathcal{P}$ or $3\mathcal{P}$.
3. He compares the correlations between the target and each pair of template traces. The correct guess is most likely to be the highest correlation.

The OTA technique is originally described for binary algorithms, but it can be easily adapted to the windows method by creating one template for a hypothesis made for each window.

The attacker model for OTA is more suitable for the Diffie-Hellman key-exchange protocol, because the input point can be selected. Nevertheless, this attack can be applied against the ECDSA algorithm, if the input point of the target device is known.

A.2 Constructing template traces for OTA

At this point, it is important to explain precisely how the interesting points to generate the template traces are chosen. With the term *interesting points* we mean the multiples of the point $\mathcal{P}$ that are expected to be the outputs of every iteration of the scalar multiplication algorithm, i.e. $2\mathcal{P}$ and $3\mathcal{P}$ for the first bit of the scalar. This is demonstrated with a graphical example depicted in Fig. 12.

Let us assume that the initial input point to the double-and-add-always algorithm is $\mathcal{P}$ and the most significant bit (K_{MSB}) of our secret scalar is 1. Then, the output of the second iteration (operations for K_{MSB-1}) is either $2\mathcal{P}$ or $3\mathcal{P}$. For example, if $K_{MSB-1} = 0$, then the output of the second iteration is $2\mathcal{P}$ and consequently the template trace for $2\mathcal{P}$ gives higher correlation to the target trace than the template for $3\mathcal{P}$. We compute the correlations between the template traces $2\mathcal{P}, 3\mathcal{P}$, and the target trace, in order to find the most likely key-bit. The highest correlation value is considered to be the right key guess.

We continue the same procedure of calculating the two possible outcomes for bit K_{MSB-2}, which are the template traces for $4\mathcal{P}$ or $5\mathcal{P}$, and then finding the highest correlation between the templates and the target trace. Fig. 13 shows how the templates for the third bit K_{MSB-2} can be generated. In general, for

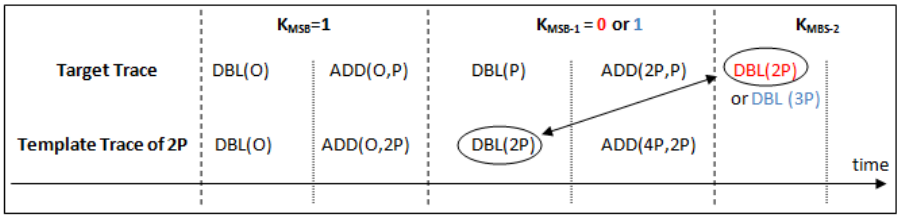

Fig. 12. How to find the second MSB K_{MSB-1} in the target trace with the template trace of $2\mathcal{P}$

each iteration of the scalar multiplication algorithm, we compare the i^{th} iteration of the scalar multiplication execution in the template trace with the $(i+1)^{\text{th}}$ execution of the target trace.

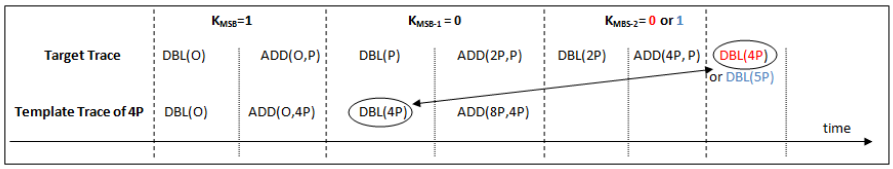

Fig. 13. How to find the third MSB K_{MSB-2} in the target trace with the template trace of $4\mathcal{P}$

A.3 Template Matching Phase

Template matching is performed at suitable parts of the traces, where key-bit related assignments take place. Our pattern matching technique, in order to distinguish the right hypothesis on the attacked bit of the scalar, is based on the Pearson correlation coefficient $\rho(X, Y)$ between the target trace and the template traces.

$$\rho(X,Y) = \frac{\sum_i (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2}\sqrt{\sum (Y_i - \bar{Y})^2}} = \frac{\langle X - \bar{X},\ Y - \bar{Y}\rangle}{||X - \bar{X}||\ ||Y - \bar{Y}||} \tag{4}$$

We chose this metric, since it is both scale and offset-shift invariant.

B Probability of the propagation of carry

Computing the probability of having an inner carry is the same as computing the probability of $(X \times Y + R \times 2^{32}) \geq 2^{64}$ with X a random value between $[0, \max\{A_7 | A \in \mathbb{F}_p\}]$, with Y a random value between $[0, \max\{B_i | B \in \mathbb{F}_p, i \in \{0, \cdots, 7\}\}]$ and with R a random value between $[0, \max\{X_i | X \in \mathbb{F}_{(p-1)^2}, i \in \{7, \cdots, 15\}\}]$. For all curves, $\max\{B_i | B \in \mathbb{F}_p, i \in \{0, \cdots, 7\}\}$ and $\max\{X_i | X \in \mathbb{F}_{(p-1)^2}, i \in \{7, \cdots, 14\}\}$ equal $2^{32} - 1$. The value $\max\{A_7 | A \in \mathbb{F}_p\}$ depends on the MSW of the characteristic of the finite field. The probability can be computed as follows:

$$\mathbb{P}(X \times Y + R \times 2^{32}) \geq 2^{64}) = \frac{1}{4}\frac{\max\{A_7 | A \in \mathbb{F}_p\}^2}{2^{64}} \tag{5}$$

We hereby give a the complete computation of the probability of an inner-carry propagation (Eq. 5)

$$\mathcal{P}(XY + 2^{32}R \geq 2^{64}) =$$

$$\sum_{x=0}^{A_7-1}\sum_{y=0}^{2^{32}-1}\sum_{r=0}^{2^{32}-1} \mathcal{P}(XY + 2^{32}R \geq 2^{64} \mid X = x, Y = y, R = r)\mathcal{P}(X = x)\mathcal{P}(Y = y)\mathcal{P}(R = r) =$$

$$\sum_{x=0}^{A_7-1}\sum_{y=0}^{2^{32}-1}\sum_{r=0}^{2^{32}-1} \mathcal{P}(xy + 2^{32}r \geq 2^{64})\frac{1}{A_7}\frac{1}{2^{32}}\frac{1}{2^{32}} =$$

$$\frac{1}{A_7}\frac{1}{(2^{32})^2}\sum_{x=0}^{A_7-1}\sum_{y=0}^{2^{32}-1}\sum_{r=0}^{2^{32}-1} 1_{xy+2^{32}r\geq 2^{64}} \text{ ,where 1 is the indicator, i.e.,} 1_z = \begin{cases} 0 & \text{if } z \text{ is false,} \\ 1 & \text{otherwise} \end{cases}$$

which can be approximated by:

$$\begin{aligned}
&\frac{1}{A_7}\frac{1}{(2^{32})^2}\int_{x=0}^{A_7-1}\int_{y=0}^{2^{32}-1}\int_{r=0}^{2^{32}-1}\delta_{xy+2^{32}r\geq 2^{64}}\mathrm{drdydx} \\
&\quad\simeq \frac{1}{A_7}\frac{1}{(2^{32})^2}\int_{x=0}^{A_7}\int_{y=0}^{2^{32}}\int_{r=0}^{2^{32}}\delta_{xy+2^{32}r\geq 2^{64}}\mathrm{drdydx} \\
&\quad= \frac{2^{32}}{A_7}\int_{x=0}^{a_7}\int_{y=0}^{1}\int_{r=0}^{1}\delta_{xy+r\geq 1}\mathrm{drdydx}
\end{aligned}$$

with $x \leftarrow x/2^{32}$, $y \leftarrow y/2^{32}$, $r \leftarrow r/2^{32}$ and $a_7 = A_7/2^{32}$.

It holds, $\delta_{xy+r\geq 1} = \delta_{r\geq 1-xy}$. Besides, $1 - xy \in [1 - a_7, 1] \subset [0, 1]$. Indeed,

$$0 \leq x \leq a_7, 0 \leq y \leq 1 \implies 0 \leq xy \leq a_7, \text{ hence } 1 - a_7 \leq 1 - xy \leq 1 \ .$$

Therefore,

$$\begin{aligned}
&\frac{2^{32}}{A_7}\int_{x=0}^{a_7}\int_{y=0}^{1}\int_{r=0}^{1}\delta_{xy+r\geq 1}\mathrm{drdydx} && = \frac{2^{32}}{A_7}\int_{x=0}^{a_7}\int_{y=0}^{1}\int_{r=1-xy}^{1}\mathrm{drdydx} = \\
&\frac{2^{32}}{A_7}\int_{x=0}^{a_7}\int_{y=0}^{1}xy\mathrm{dydx} && = \frac{2^{32}}{A_7}\int_{x=0}^{a_7}x\mathrm{dx}\times\int_{y=0}^{1}y\mathrm{dy} = \\
&\frac{2^{32}}{A_7}\left[\frac{x^2}{2}\right]_0^{a_7}\times\left[\frac{y^2}{2}\right]_0^{1} && = \frac{2^{32}}{A_7}\frac{a_7^2}{2}\times\frac{1}{2} = \frac{2^{32}}{A_7}\frac{1}{4}a_7^2 = \frac{1}{4}\frac{A_7}{2^{32}} \ .
\end{aligned}$$

For $A_7 = 2^{32-1}$, this yields ≈ 0.25. For $A_7 = \texttt{0xA9FB57DA}$, this yields ≈ 0.166.