Online Template Attack on ECDSA:
Extracting Keys via the Other Side

Niels Roelofs, Niels Samwel(✉), Lejla Batina, and Joan Daemen

Radboud University, PO Box 9010, 6500 GL Nijmegen, The Netherlands
nielsroelofs95@gmail.com, {nsamwel,lejla,joan}@cs.ru.nl

Abstract. We retrieve the ephemeral private key from the power trace of a single scalar multiplication in an ECDSA signature generation and from that the signing private key using an online template attack. The innovation is that we generate the profiling traces using ECDSA signature verification on the *same* device. The attack can be prevented by randomization of the (projective) coordinates of the base point.

Keywords: Online template attacks · Scalar multiplication · ECDSA

1 Introduction

Template attacks are a very powerful type of side channel attacks, typically featuring two phases: profiling and key recovery [5]. In the profiling phase, the device's leakage is characterized as a probability distribution to make optimally use of the information present in each time sample of a leakage trace. In the attacking phase, the key is identified using the maximum likelihood principle. In the public-key cryptography application scenario, template attacks can be used to extract a private key from a single power consumption or electromagnetic emanation trace of a device performing an exponentiation of scalar multiplication [15].

In this paper we concentrate on the scalar multiplication with a known base point and a secret scalar. The goal of the adversary will be to extract the latter one. Typically, an attacker derives some bits of the secret scalar based on some well-defined positions in the trace and thereafter recovers the remainder of the scalar via lattice reduction techniques. The first phase of the attack (profiling) requires the generation of *templates* and for that the attacker ideally must be able to take traces from the same device, running scalar multiplications with chosen scalars. We call those *profiling* traces.

With classical template attacks, the required number of profiling traces to perform a successful attack can be large, depending on the noise level, set-up, implementation characteristics etc. An important innovation in this respect was the technique of the Online Template Attack (OTA) [1] that reduces the number of profiling traces to less than two times the number of scalar bits.

In real-world attack scenario's it rarely happens that an attacker can get profiling traces from the device that contains the target key. For that reason,

A. Nitaj and A. Youssef (Eds.): AFRICACRYPT 2020, LNCS 12174, pp. 323–336, 2020.
https://doi.org/10.1007/978-3-030-51938-4_16

one often obtains the profiling traces from a device of the same type running the same software. This is, for instance, the case if the target device is a JavaCard used for banking or used as an ID card, assuming that the attacker is able to obtain an open JavaCard from the same type. However, profiling traces obtained from a device different than the target device are not necessarily accurate due to variations between individual chips. This is the so-called issue of *portability* that has received wide coverage in many papers the last few years [3,6]. For dedicated security products such as smart cards it is often even infeasible to obtain an open device similar to the target device and profiling is simply out of the question.

In this paper we make use of a simple observation to generate profiling traces using the target device itself, thereby completely bypassing the portability discussion and the non-availability of open devices. This observation is the following: products that support the Elliptic Curve Digital Signature Algorithm (ECDSA) algorithm typically support both signature generation that includes a scalar multiplication with an ephemeral private key and signature verification that includes two scalar multiplications with scalars that are provided to the device and can hence be chosen by an attacker. The idea is simply to use scalar multiplications in the signature verification to generate profiling traces. There are still some obstacles: the scalar multiplications in signature generation and verification may make use of different algorithms. In a way, the issue of portability between devices is now replaced by a portability problem between algorithms. In this paper, we report on an attack that overcomes the difference in algorithms and succeeds in extracting the private ephemeral key from a single ECDSA signature generation trace using profiling traces from ECDSA signature verification traces on the same device.

The remainder of the paper is structured as follows. We discuss related work in Sect. 1.1 and summarize the contributions of this paper in Sect. 1.2. We provide the necessary background in Sect. 2. Section 3 focuses on the vulnerability identified and how it might actually be exploited and in Sect. 4 we introduce the actual exploitation process itself.

1.1 Related Work

There are two research directions that should be considered as relevant previous work. The first one is on portability and the other one is the work on online template attacks.

Considering the importance of portability, Elaabid and Guilley [9] show that, when precharacterized templates are outdated, the consequence can be as drastic as ranking the correct key last. The work of Choudary and Kuhn focuses on differences between devices when performing portable template attacks (considering mainly DC offset caused by temperature changes) [7]. Bhasin et al. recently compared different machine learning techniques on the efficacy with respect to portability [3]. A common point in all those works is they identified the problem and try to amortize the penalty due to portability.

Online template attacks are an adaptive template-attack technique that can recover a complete scalar from only one power trace of a scalar multiplication

using this scalar [1]. The attack is characterized as online, because the templates are created after the acquisition of the target trace. The attack is not a typical template attack; i.e. no pre standard profiling template-building phase is necessary. The attack is demonstrated by acquiring one target trace from the device under attack and comparing patterns of certain identical operations from this trace with templates obtained from the attacker's device that runs the same implementation. Our work examines another dimension of online templates, as we address the portability issue at the same time by applying OTA on the same device. This strategy requires the templates creation phase to use another algorithm than the one which implementation we attack. This minor but subtle difference requires some creativity with the new attack we propose as the templates to compare and identify within the actual trace are not readily available. Our strategy is initiated by a common scenario in real-world applications where one often has only one device available and it solves the portability problem.

1.2 Contributions

In this work we show the feasibility of OTA using a single device, so templates collection and the key recovery are performed in identical conditions. This use case puts the portability discussion in perspective. We successfully extract the ephemeral private key from a single power consumption trace of an ECDSA signature generation implemented on an 8-bit microcontroller. We use the ECDSA signature verification on the same device for the online templates. The fact that the scalar multiplication in signature verification uses another algorithm than the one in signature generation does not prevent our attack.

2 Background

In this section we discuss some principles needed to understand our attack: the ECDSA signature scheme, the relevant scalar multiplication algorithms and the principle of OTA.

2.1 ECDSA

By now it is almost twenty years ago that Johnson et al. proposed the elliptic curve variant of the Digital Signature Algorithm [13]. Next to RSA [18] and Ed25519 [2] it is one of the most commonly used signature algorithms. We specify the ECDSA signing algorithm in Algorithm 1 and its verifying counterpart in Algorithm 2, along with the symbol clarification in Table 1.

In the context of side-channel analysis, in relation to elliptic curves, scalar multiplications are the standard go-to, especially if the scalar is a secret value. Namely, in the case of ECDSA, if an adversary somehow learns the value for k by exploiting some side channel during the scalar multiplication on line 3 of Algorithm 1, he can compute the private key d via line 8 of Algorithm 1 since it is the only unknown variable left.

Table 1. Symbol clarification for ECDSA related algorithms.

Symbol	Meaning	Element of
B	Base point	The elliptic curve
l	Order of base point	$\mathbb{N}$
k	Ephemeral private key	$\mathbb{Z}/l\mathbb{Z}$
d	Long-term private key	$\mathbb{Z}/l\mathbb{Z}$
D	Public key ($D = [d]B$)	$\langle B \rangle$
$H(...)$	Hash of input	$\mathbb{Z}/l\mathbb{Z}$
M	Message	$\{0,1\}^*$
(r, s)	Signature	$(\mathbb{Z}/l\mathbb{Z}, \mathbb{Z}/l\mathbb{Z})$

Algorithm 1: ECDSA signature generation.

Input: d, l, B, M
Output: (r, s)
$z \leftarrow H(M)$
while *true* **do**
  $k \leftarrow create\_nonce(1, l-1)$
  $(x, y) \leftarrow [k]B$
  $r \leftarrow x \ mod \ l$
  if $r == 0$ **then**
    continue `// new k needed`
  end
  $s \leftarrow k^{-1}(z + rd) \ mod \ l$
  if $s == 0$ **then**
    continue `// new k needed`
  end
  break
end
return (r, s)

Algorithm 2: ECDSA signature verification.

Input: l, r, s, B, D, M
Output: *signature_accepted*
$signature\_accepted \leftarrow 0$
$z \leftarrow H(M)$
$w \leftarrow s^{-1} \ mod \ l$
$u_1 \leftarrow zw \ mod \ l$
$u_2 \leftarrow rw \ mod \ l$
$(x, y) \leftarrow [u_1]B + [u_2]D$
if $x == r \ mod \ l$ **then**
  $signature\_accepted \leftarrow 1$
end
return *signature_accepted*

2.2 Double-and-Add

One of the simplest fast methods for scalar multiplication is the double-and-add algorithm, see Algorithm 3.

This algorithm iterates over the scalar k, starting from the most significant bit, doubles the intermediate result in every round and conditionally adds the elliptic curve point if the scalar bit processed equals 1.

Algorithm 3: Double-and-add algorithm.

Input: $(k_{n-1}, k_{n-2}, ..., k_0), P$
Output: $Q = kP$

```
1 Q ← O
2 for i ← n − 1 down to 0 do
3 |   Q ← 2Q
4 |   if k_i == 1 then
5 |   |   Q ← Q + P
6 |   end
7 end
8 return Q
```

However, this procedure is insecure to use in a cryptographic setting when k has to be kept private if a side-channel vector exists in the form of time due to the conditional statement in line 4. If different rounds within the execution of the algorithm can be distinguished from one another the secret scalar can be reconstructed bit by bit. This is not necessarily trivial. Nevertheless, to prevent such time-based attacks, one should avoid branching on secret data altogether.

2.3 Montgomery Ladder

An alternative for the double-and-add algorithm is the Montgomery ladder [16], as specified in Algorithm 4. Peter Montgomery invented it back in 1987 to speed up factorization using elliptic curves and it offers resistance against the type of side-channel attacks as discussed above.

Due to the regular structure of the algorithm, and therefore the lack of branches, the scalar multiplication occurs in constant time and even provides protection against some power analysis techniques, such as simple power analysis [14].

Still, the algorithm needs to somehow distinguish between 0 and 1 scalar bits being processed. It does so by working with two shares, X_0 and X_1. Based on the key bit, the shared a swapped in constant time by the function *cswap*. The value shares are swapped if a bit is set.

Algorithm 4 only gives a high level overview of the Montgomery ladder. In the last few decades quite some research has been done in optimizing it for speed and/or memory usage and improving its resistance against side-channel analysis. For the remainder of the paper we refer to the Montgomery ladder implementation done by Hutter et al. [12].

Algorithm 4: Montgomery ladder.

Input: $(k_{n-1}, k_{n-2}, ..., k_0), P$
Output: $Q = kP$
1 $X_0 \leftarrow O$
2 $X_1 \leftarrow P$
3 **for** $i \leftarrow n-1$ **down to** 0 **do**
4 $\quad X_1 \leftarrow X_0 + X_1$
5 $\quad X_0 \leftarrow 2X_0$
6 $\quad (X_0, X_1) = cswap(X_0, X_1, k_i)$
7 **end**
8 **return** X_0

2.4 Online Template Attack

OTA [1] can be seen as a combination a template attack [5] and doubling attacks as introduced for the scalar multiplication operation on an elliptic curve [10].

The main difference between a standard template attack and OTA is that the latter requires at most two power traces per scalar bit to recover, while in the former case it easily can be hundreds or thousands. Additionally, with OTA the process of template generation and matching can be interleaved per bit to recover. So, OTA is faster in template generation building and matching and uses significantly less storage in comparison to a standard template attack.

When looking at Algorithm 3, the key idea of OTA is the following: based on the bit value of k_{n-2}, Q will equal $2P$ or $3P$ before the doubling operation at the start of the third round of the for loop. That is because the most significant bit k_{n-1} will equal 1. Although an adversary does not know which case occurred, he can create his own profile trace, a template, for both cases. Assuming that he has access to a power consumption trace of a target doing the secret scalar multiplication and is able to identify a single doubling operation of an elliptic curve point, he can apply correlation analysis techniques to derive the correct scalar bit. That is, he compares how similar the doubling operation in the beginning of the third round of the for loop is with both of his templates in terms of power consumption. Depending on the scalar multiplication and the hardware used, it might even be possible to only generate one template and accept the corresponding correlation value based on a certain threshold value, which has to be determined empirically.

Once the value of k_{n-2} has been identified, the whole process can be repeated to determine the next bit of the scalar. The only matters that require updating are the possible input values for the doubling operation on line 3 of Algorithm 3 and the location of the doubling operation in the target trace. For example, assuming that $k_{n-2} = 0$, the template traces will represent $4P$ and $5P$.

Note that by itself it is overkill to apply OTA to the double-and-add algorithm because a simple power analysis technique will already suffice. However, in the next section we grasp the key idea of OTA and apply it to an implementation of the Montgomery ladder.

3 Spotting the Attack Vector

This section focuses on the vulnerability identified and how it might actually be exploited.

3.1 Finding the Similarity

As introduced before, for the attack to succeed, we need to find some key dependent operation in the ECDSA signing procedure, that can be mimicked with its verifying counterpart. If such a relation can be found, it allows an adversary to generate templates of that operation, one where the bit of the scalar is 0 and one where it is equal to 1.

In side channel analysis of ECDSA the typical method to achieve key extraction is to use the scalar multiplication of the ephemeral private key k with the base point B, see line 3 of Algorithm 1. If k can be reconstructed, the extraction of the private key is trivial. During the ECDSA verification procedure there are two scalar multiplications taking place as well, so those can serve as an entry point. However here arises a problem: the scalar multiplication method used during signing is the Montgomery ladder, while its counterpart uses the standard double and add algorithm. Not only do their internal computations differ, so do their coordinate systems. In our scenario we assume the usage of standard projective coordinates for the Montgomery ladder and Jacobian coordinates for the double and add algorithm. So in order to find our targeted key-dependent operation we have to dig deeper and look at the underlying field arithmetic in both cases. Once again, solving this kind of issues is the core novelty of this paper since it makes the portability discussion redundant by introducing the new OTA-like method.

Signature generation and signature verification use different scalar multiplication algorithms, thus we are not able to create templates of the whole double-and-add iteration. Instead, we aim to find an operation that is computed in both algorithms. Our requirements are: 1) In the signature generation algorithm the input value of the operation should depend on the key and 2) in the signature verification we should be able to control the input value.

The point that is doubled in the iteration depends on the key bit of the previous iteration, namely $2P$ or $2P+1$ (see line 4 and 5 of Algorithm 4) will be doubled. We locate an operation in that step to create templates of the key bit used in the previous iteration. For the point doubling operation, the x-coordinate is squared. In the signature verification algorithm, there is also a squaring of the x-coordinate. Since the x-coordinate is part of the public key D, an attacker can control it. Therefore, this value of the x-coordinate and the squaring operation is a good candidate for the generation of templates.

For each key bit, two templates are created, one for the case where $2P$ is doubled and one where $2P + 1$ is doubled, with P the resulting point of the previous round in X_0. The template consists of a power trace of the squaring operation from the corresponding point doubling. To determine which key bit was most likely in the target trace, the Pearson correlation [4] is computed with

each template where the template with the highest correlation value corresponds to the correct key bit.

3.2 Preparing the Input

Now that we have found our attack window, the question becomes how to get meaningful data as input for the identified squaring operation so that we can build our wanted templates. A first problem to overcome is to compute the possible intermediate values for X_0 in the Montgomery ladder which may serve as input for the squaring operation in the Jacobian doubling.

The issue can quite easily be solved by implementing our own Montgomery step function in SageMath[1]. After all, the input values for this step function are known, except for the secret scalar itself of course, hence two possible values for X_0 which make up the template.

However, a more serious problem arises with our values just calculated. Namely, when looking at line 6 of Algorithm 2, it only makes sense to feed our template values for X_0 via the public key D, since it is not likely that an outside adversary can manipulate the base point B to use during a signature verification. However, the complication that now arises is that typically in an ECDSA signature verification implementation as a first step a check is made whether the given D is really a public key, that is, whether the point D lies on the curve. The issue here is that it is quite feasible that both computed values for X_0 are indeed not representing a point on the curve.

To circumvent this problem, Papachristodoulou [17] came up with the idea to introduce bit flipping on the least significant bit of X_0. By flipping it, effectively a new value for X_0, say x', is computed. Together with its y' component the coordinate (x', y') might lie on the curve and hence pass the test that D is a valid public key. If (x', y') is still not a point on the curve, the bit flipping is reversed and instead the second least significant bit is flipped. This process continues just as long until a point on the curve is found. Based on experiments of Papachristodoulou and ourselves, a maximum of five tries are needed to find an elliptic curve point that actually lies on the curve.

Since we now know how and what to fill in for D in Algorithm 2, we can focus on the scalar u_2. Looking at Algorithm 3, we can think of line 3 as our Jacobian doubling with our desired squaring operation. In order to get our calculated values for X_0 into the doubling, we need the most significant bit of u_2 to be 1 so that in the beginning of the second round in the for loop our wanted operation happens. It does not really matter what happens afterwards, we only care about the power trace of the squaring operation. u_2 itself is depending on the signature tuple (r, s). So, by some simple brute forcing of either of the values we should have some random value for u_2 for the most significant bit equals 1.

Today's applications typically do not implement a simple double-and-add algorithm and execute it two times to get both scalar multiplications during ECDSA verification. Typically, they apply some kind of optimization, such as

[1] http://www.sagemath.org.

Non-Adjacent Form (NAF) [11] or the Straus-Shamir trick [19]. It is important to note that also these implementations are affected, because the vulnerability identified resides inside the underlying finite field arithmetic and not on the higher level scalar multiplication algorithm.

Furthermore, also realize that when creating our templates, the corresponding signature verification results do not have any meaning. It is a logical consequence of taking an intermediate result of one algorithm and use it as input for another in a different coordinate system. The takeaway here is that the verification results are irrelevant, we only care about the retrieved power traces.

4 Exploiting the Attack Vector

Now that we have introduced the entry point of our attack, we will discuss our measurement setup, after which we describe the procedure of bit extraction. We will finish with an effective countermeasure against OTA.

4.1 Measurement Setup

For a total overview of the measurement setup, see Fig. 1 and Table 2 for the symbol clarification.

The device used for our online template attack is the ChipWhisperer-Lite Classic (CWLC). It uses the Atmel Xmega 128D4 8-bit processor which runs at 32 MHz. Actually, the CWLC board consists of two parts: one with Xmega target and one main board which communicates with the target. We access the Xmega target via the main board via a USB-cable. On our computer we run Jupyter notebook to interact with the board within a specially prepared virtual machine[2] provided by the manufacturers of the CWLC.

The board itself is connected to the Waverunner 8404M-MS from Teledyne LeCroy which uses a sample rate of 25 Msamples/s in order to get a power trace. Besides that, there is also wiring from the board to the oscilloscope to plot any trigger mechanisms used. Even though the Xmega target runs our code in constant time, the triggering makes the identification process of the squaring in the trace a bit more convenient. However, it is not a necessity to use it. Finally, the oscilloscope itself is connected to another computer which runs Inspector[3], a side-channel analysis tool we will discuss in the next section.

On a completely different note, it should be mentioned that the execution time of a 256 bit elliptic curve scalar multiplication on a 8-bit architecture is a costly endeavor: approximately 60 s. In contrast, a typical home computer does it in the order of milliseconds. Additionally, the oscilloscope used can capture up to 128 Msamples. With our sampling rate of 25 Msamples/s this means that around 5 s of the signature generation can be captured. After doing some experimenting with this setup we found out that one single round of the Montgomery ladder

[2] https://github.com/newaetech/chipwhisperer.
[3] https://www.riscure.com/security-tools/inspector-sca/.

takes 192 ms and that the squaring we are interested in happens at a constant offset of 111.5 ms from the start of a round and takes 10.8 ms. So, this means that we can recover around 20 scalar bits with our setup. Remember, the scalar used with ECDSA signing is different and randomly chosen for every single execution so it is useless to take another target trace. However, this problem can simply by bypassed by using an oscilloscope which has bigger storage capabilities. On top of that, an adversary could try to lower the sampling rate. However, we did not test such a scenario.

4.2 Bit Extraction

Once both profiling traces have been taken for the second most significant bit, the post-processing can start. Realize that we start with the second bit, because the most significant bit will always be 1.

The basic idea is to calculate the Pearson correlation coefficient between a profile trace and every single offset of the target trace. To increase the speed of these calculations we applied the principle of window resampling with a window size of 20 with an overlap of 0.15, which leads to a sample reduction of factor 17. In our case window resampling means that the average is taken of 20 samples, which serves as 1 new sample, and that the last 3 samples of a window are reused in the next window. These numbers have been chosen empirically and are a balance between computation time and still being able to distinguish between different templates. Namely, creating the windows too big in size effective leads to a situation where every window starts to look similar which gives as a consequence that further analysis is not possible.

Table 2. Number clarification for OTA setup Fig. 1.

Number	Meaning
1	CWLC Xmega target
2	CWLC main board
3	Micro-USB connector
4	Measuring cable to oscilloscope
5	Wiring for triggering to oscilloscope
6	Computer running Jupyter notebook
7	Waverunner 8404M-MS oscilloscope
8	Processing power trace with Inspector

The concept above can be implemented quite easily with the tool Inspector from Riscure. If done so separately for both templates, we are given Fig. 2, where (a) represents the correlation analysis when the second most significant bit of the scalar equals equals 0 and (b) when it equals 1.

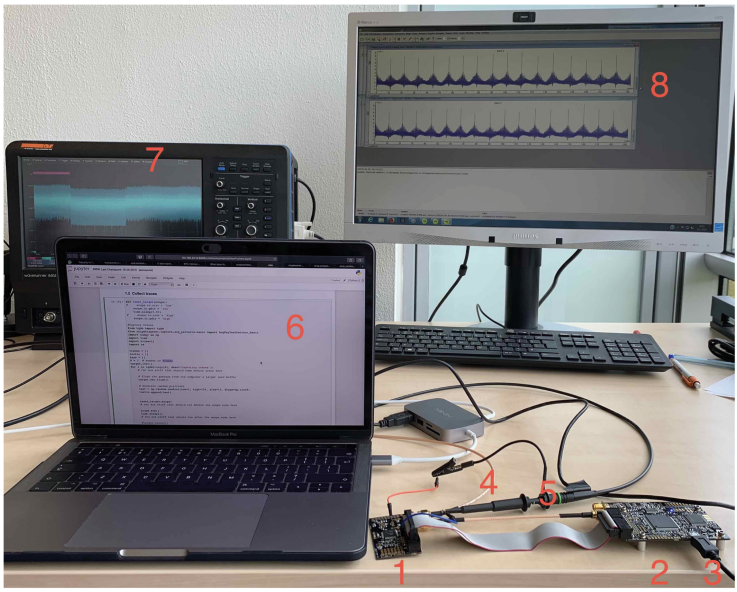


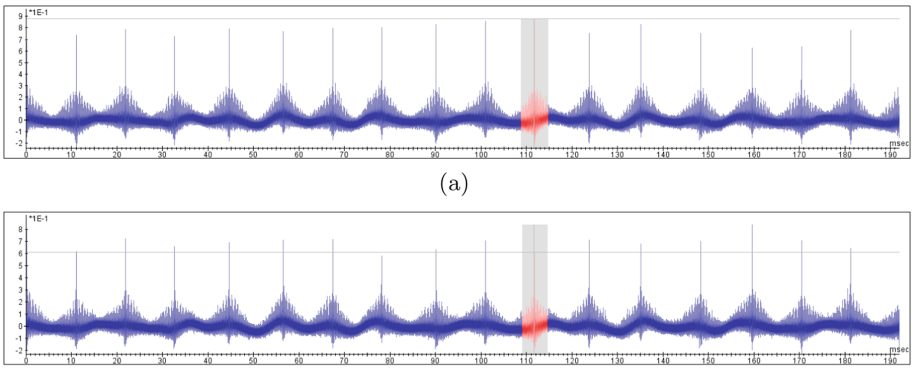
Fig. 1. Overview of the OTA attack setup.

The x-axis of both subfigures captures the time of one complete Montgomery ladder step, 192 ms. The parts in the figure marked in red refer to the area where the correlation coefficient between our template and our targeted squaring in the Montgomery ladder in the target trace should be. And indeed, there is a peak at an offset of 111.5 ms. The other correlation peaks in the figures represent points in time where either another squaring or multiplication operation in the Montgomery ladder occurs. Based on the implementation of Hutter et al. [12], we should have the eleventh peak, which is indeed the case (note the peak at an offset of 0 ms).

As a final step, the only thing that is now left to do is to compare the correlation peak values at the offset of 111.5 ms: (a) gives a correlation of 0.88 and (b) 0.61. Therefore, the conclusion can be drawn that it is more likely that the second most significant bit of the scalar equals 0. And indeed, when creating our setup we set the most significant byte of the scalar to 0×97, which gives in binary 10010111.

From here on onward, the whole procedure can be repeated for the third most significant bit and thereafter the fourth et cetera. For every scalar bit to recover new values have to be computed for X_0. They will serve as input for the templates to generate. Hereby it is important to realize that these values for X_0 depend on the previously scalar bits identified. Additionally, when trying to derive to value for the n^{th} bit, one should look at the correlation peak at an offset of $(n-2)192 + 111.5$ ms.

In total, we repeated this procedure sixteen times and we were able to derive the scalar bit every single time without error, once we applied the appropriate window resampling parameters. Nevertheless, even if somehow a wrong bit would get chosen, the correlation values would significantly drop in successive rounds thereafter, clearly indicating that somewhere something went wrong.



(a)

(b)

Fig. 2. Correlation results second most significant bit templates. (a) represents bit 0 and (b) bit 1.

4.3 Countermeasures

As demonstrated above, simply implementing the elliptic curve scalar multiplication in a time constant way does not provide protection against OTA. However, as Coron wrote in his influential paper [8], there are some countermeasures that can be taken in order to prevent certain types of power analysis side-channel attacks on elliptic curve cryptosystems.

One of those countermeasures described also protects against OTA: the concept of randomized projective coordinates. It is based on the concept that the projective representation of an elliptic curve point is not unique. See Eq. (1) where λ is a random scalar in $\mathbb{F}_p$, the finite field with modulo p.

$$(X, Y, Z) \leftarrow (\lambda X, \lambda Y, \lambda Z) \tag{1}$$

When this concept is applied on the base point while signing, an adversary can no longer create any meaningful templates for OTA. After all, it becomes impossible to calculate the intermediate computation value of X_0 in the Montgomery ladder which he wants to serve as input for the squaring operation during the verification process because the value is randomized.

A big advantage of this countermeasure is that it is very cheap to implement in terms of speed: it only takes three finite field multiplications. In contrast, a full

Montgomery ladder consists of several thousands of such operations. However, there is one big assumption that must not be overlooked: we assume that a device has the capability to generate randomness, an operation that by itself is far from trivial and definitely not implemented on every device out there in the world.

5 Conclusion

In this paper we demonstrated the feasibility of extracting a secret scalar used in the ECDSA verification procedure via its verifying counterpart, which does not use any secret values by itself. By itself, this is nothing the new. However, the novelty lies in the fact that a single device is used for both collecting the target trace and building the profile traces, while different scalar multiplication algorithms with different coordinate systems are used during ECDSA signing and verification.

Although we showed a rather simple but effective attack, so is its countermeasure of randomizing the coordinates of the base point. Granted, hereby we assume that the device has the ability to generate random numbers, which is definitely not always the case.

References

1. Batina, L., Chmielewski, Ł., Papachristodoulou, L., Schwabe, P., Tunstall, M.: Online template attacks. J. Cryptogr. Eng. **9**(1), 21–36 (2017). https://doi.org/10.1007/s13389-017-0171-8
2. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. J. Cryptogr. Eng. **2**(2), 77–89 (2012). https://doi.org/10.1007/s13389-012-0027-1
3. Bhasin, S., Chattopadhyay, A., Heuser, A., Jap, D., Picek, S., Shrivastwa, R.R.: Mind the portability: a warriors guide through realistic profiled side-channel analysis. IACR Cryptol. ePrint Arch. **2019**, 661 (2019)
4. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_2
5. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3
6. Choudary, M.O., Kuhn, M.G.: Efficient, portable template attacks. IEEE Trans. Inf. Forensics Secur. **13**(2), 490–501 (2018)
7. Choudary, O., Kuhn, M.G.: Efficient template attacks. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 253–270. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_17
8. Coron, J.-S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_25
9. Elaabid, M.A., Guilley, S.: Portability of templates. J. Cryptogr. Eng. **2**(1), 63–74 (2012). https://doi.org/10.1007/s13389-012-0030-6

10. 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. LNCS, vol. 2779, pp. 269–280. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45238-6_22
11. Hankerson, D., Vanstone, S., Menezes, A.: Guide to Elliptic Curve Cryptography. Springer, New York (2004). https://doi.org/10.1007/b97644
12. Hutter, M., Joye, M., Sierra, Y.: Memory-constrained implementations of elliptic curve cryptography in co-Z coordinate representation. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 170–187. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21969-6_11
13. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Secur. **1**(1), 36–63 (2001). https://doi.org/10.1007/s102070100002
14. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
15. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Power analysis attacks of modular exponentiation in smartcards. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 144–157. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_14
16. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comput. **48**(177), 243–264 (1987)
17. Papachristodoulou, L.: Masking curves: side-channel attacks on elliptic curve cryptography and countermeasures. Ph.D. thesis, Radboud University Nijmegen (2019)
18. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
19. Straus, E.G.: Addition chains of vectors (problem 5125). Am. Math. Mon. **70**(806–808), 16 (1964)