Breaking Ed25519 in WolfSSL

Niels Samwel[1(✉)], Lejla Batina[1], Guido Bertoni[2], Joan Daemen[1,3], and Ruggero Susella[4]

[1] Digital Security Group, Radboud University, Nijmegen, The Netherlands
{n.samwel,lejla,joan}@cs.ru.nl
[2] Security Pattern, Brescia, Italy
g.bertoni@securitypattern.com
[3] STMicroelectronics, Diegem, Belgium
[4] STMicroelectronics, Agrate Brianza, Italy
ruggero.susella@st.com

Abstract. Ed25519 is an instance of the Elliptic Curve based signature scheme EdDSA that was recently introduced to solve an inconvenience of the more established ECDSA. Namely, both schemes require the generation of a value (scalar of the ephemeral key pair) during the signature generation process and the secrecy of this value is critical for security: knowledge of one such a value, or partial knowledge of a series of them, allows reconstructing the signer's private key. In ECDSA it is not specified how to generate this random value and hence implementations critically rely on the quality of random number generators and are challenging to implement securely. EdDSA removes this dependence by deriving the secret deterministically from the message and a long-term auxiliary key using a cryptographic hash function. The feature of determinism has received wide support as enabling secure implementations and in particular deployment of Ed25519 is spectacular. Today Ed25519 is used in numerous security protocols, networks and both software and hardware security products e.g. OpenSSH, Tor, GnuPG etc.

In this paper we show that in use cases where power or electromagnetic leakage can be exploited, exactly the mechanism that makes EdDSA deterministic complicates its secure implementation. In particular, we break an Ed25519 implementation in WolfSSL, which is a suitable use case for IoT applications. We apply differential power analysis (DPA) on the underlying hash function, SHA-512, requiring only 4 000 traces.

Finally, we present a tweak to the EdDSA protocol that is cheap and effective against the described attack while keeping the claimed advantage of EdDSA over ECDSA in terms of featuring less things that can go wrong e.g. the required high-quality randomness. However, we do argue with our countermeasure that some randomness (that need not be perfect) might be hard to avoid.

Keywords: EdDSA · SHA-512 · Side-channel attack
Real world attack

N. P. Smart (Ed.): CT-RSA 2018, LNCS 10808, pp. 1–20, 2018.
https://doi.org/10.1007/978-3-319-76953-0_1

1 Introduction

Since its invention in the late 80's independently by Koblitz [18] and Miller [22] Elliptic Curve Cryptography (ECC) has established itself as the default choice for classical public-key cryptography, in particular for constrained environments. Especially lightweight Internet of Things (IoT) applications and resources sparse platforms such as RFID tags and sensor nodes consider ECC exclusively for their (exceptional) public-key requirements. This does not come as surprise knowing that working in fields with size 160 bits or so is considered to be at least as secure as RSA using around 1200 bits [1]. This property often results in implementations of smaller memory/area footprints, lower power/energy consumption etc.

A recent initiative is to seriously consider and consequently standardize some post-quantum cryptosystems, i.e., those that could survive a prospect of having a quantum computer that (if built) would break all classical public-key cryptosystems. However, this does not (yet) make research on ECC obsolete as there is still a number of years to go, before the actual transition to post-quantum cryptography might occur.

Research on ECC has evolved from the first proposals to numerous works on protocols, algorithms, arithmetic, implementations aspects including side-channel security etc. Especially, looking into different curves and representations has become a resourceful topic for various optimizations. Twisted Edwards curves [15] were proposed by Bernstein and Lange [8,10] featuring a complete point operation formulae that is proven to be more efficient and secure with respect to side-channel leakages.

All together, the easiness of constant-time implementations and performance boost, together with somewhat reduced confidence in NIST-standardized curves have made many users transitioning to Edwards curve based protocols including OpenSSH, Tor, TLS, Apple AirPlay, DNS protocols etc [3].

In particular, Edwards-Curve Digital Signature Algorithm (EdDSA) is very popular in real-world application of cryptography. An instance of EdDSA using Edwards Curve25519 called Ed25519 is used among others, in Signal protocol (for mobile phones), Tor, SSL, voting machines in Brazil etc. There is an ongoing effort to standardize the scheme, known as RFC 8032.

EdDSA including Ed25519 is claimed to be more side-channel resistant than ECDSA [9], not just in terms of resisting software side-channels i.e. featuring constant timing. The authors rely on the idea to "generate random signatures in a secretly deterministic way" such that "different messages lead to different, hard-to-predict values of ephemeral key r". This aims at the known algorithms using lattice methods to compute the long-term ECDSA key from the knowledge of as few as 3 bits of the ephemeral key r for a few hundred of signatures [23]. This knowledge can be typically obtained from side-channel attacks or from non-uniformity of the distribution from which r is taken, so the authors of EdDSA rightfully point at the fact that the "deterministic feature" results in no obvious side-channel leakage exploits. They also state that "no per-message randomness is consumed", making this additionally attractive due to the notoriously expensive generation of random numbers.

In this work we show that, although expensive, one should possibly retreat to randomness as we are able to break Ed25519, implemented in WolfSSL, by using 1st order differential power analysis. Actually, the combination of the key and the message in the hash computation (without randomness) makes it a classic scenario for DPA as proposed in the seminal paper of Kocher et al. [19]. More in detail, although we exploit the non-linearity of modular addition in the hash computation, EdDSA is a perfect target for this kind of attack as it fully breaks the scheme after collecting as few as 4 000 power or EM traces. We give all the details of the attack later in this paper, including a simple fix that would render the attack infeasible.

The rest of the paper is organized as follows. First, we mention related previous work and specify our contributions. In Sect. 2, we provide background information required for the remainder of the paper. Section 3 gives the ingredients of our attack and dissect the methodology from attacking the signature scheme down to DPA on modular addition. In Sect. 4 we present the practical attack on a 32-bit ARM architecture running WolfSSL and some caveats that had to be overcome before turning the idea into a practical attack. We present the results of the attack with a technique to reduce the number of traces. In Sect. 5 we present a countermeasure and Sect. 6 concludes the paper.

1.1 Related Work

Ed25519 uses SHA-512, a member of the SHA-2 family, for hashing. SHA-512 is used in many applications, often in HMAC mode. Namely, as SHA-1 collisions were expected for years up to now many implementers started already upgrading to alternatives. As a matter of fact, due to the recently found collisions in SHA-1 it is strongly recommended to immediately migrate to SHA-2 or SHA-3.

Several works looked into side-channel vulnerabilities in SHA-1 and SHA-2 hash functions or other symmetric-key primitives using modular addition. McEvoy et al. [21] presented an attack on the compression function of SHA-2. Basically, they present the theory of an attack on an HMAC construction using DPA but a full attack on real traces was not executed. The authors also presented a countermeasure against DPA using masking.

In another attack on the compression function of SHA-2, Belaid et al. [5] target other steps (than McEvoy et al.) and they provide results on simulated traces. The authors also suggest a countermeasure for their specific attack.

In Seuschek et al. [26] the authors discuss an attack on EdDSA. They apply the attack as described in [5,21]. However, they do not execute the attack on either simulated or real traces.

In this work we exploit another aspect of SHA-512. Namely, our attack is the first one to exploit leakage in the computation of the message schedule of SHA-512 (in contrast to the previous paper where they target the addition of part of the message in the round function). More specifically, we target the modular addition operation and exploit the non-linearity of it to attack EdDSA.

Attacking modular addition is done before by several authors. Zohner et al. [27] attack the modular addition in the hash function Skein using real

traces. The authors discuss issues regarding a certain symmetry in the results of an attack on modular addition and present a solution. Namely, the correct result value modified by flipping the most significant bit also shows a correlation. This result is called the symmetric counterpart of the correct result. Lemke et al. [20] and Benoît and Peyrin [6] also attack modular addition in other symmetric ciphers on simulated traces. A similar symmetry in the results was observed.

In our work we actually use the symmetry in the results of the attack in a different manner. More precisely, we use it to reduce the number of traces until the key recovery. Additionally, we provide results of our attack on real traces supporting the hypotheses from the theoretical attack considerations. Except for [27] the previous works only support their theory with simulations.

Recently, a paper was published[1] that discusses several fault attacks on EdDSA [4], it also mentions using DPA on the hash function to recover the key of EdDSA.

1.2 Contributions

Here we summarize the main contributions of this paper:

- We present the first side-channel attack on Ed25519 using real traces. To this end, we extract secret information i.e. a key that allows us to forge signatures on any message using the key obtained. The key recovery is successful after collecting a few thousands of power consumption traces corresponding to signature generation.
- We present the first side-channel attack on the message schedule of SHA-512 targeting the modular addition operation within. The ideas are extendable to other similar constructions. In contrast to previous attacks on SHA-512 we target the extension of the message schedule instead of the addition of a message in the round function.
- Our attack breaks a real-world implementation. The traces were generated by an implementation of Ed25519 from the lightweight cryptographic library WolfSSL on a 32-bit ARM based micro-controller. This kind of implementation particularly targets low-cost and/or resource-constrained environments as in the IoT use cases and similar.
- Finally, we present a countermeasure against this attack. The countermeasure is a result of a small tweak in EdDSA that would not just make the attack infeasible but also does not add much overhead to implementations. A similar countermeasure where randomness is added was presented in the XEdDSA and VXEdDSA Signature Schemes [2] (more details in Sect. 5).

2 Background

2.1 EdDSA

EdDSA [9] is a digital signature scheme. The signature scheme is a variant of the Schnorr signature algorithm [25] that makes use of Twisted Edwards Curves.

[1] This paper was published after the submission deadline of CT-RSA.

The security of ECDSA depends heavily on a good quality randomness of the ephemeral key, which has to be truly random for each signature. Compared to ECDSA, EdDSA does not need new randomness for each signature as the ephemeral key is computed deterministically using the message and the auxiliary key that is derived from the private key. The security depends on the secrecy of the auxiliary key and the private scalar. This does not create a new requirement as we need to keep a private key secret anyway.

In Ed25519, a twisted Edwards curve birationally equivalent to Curve25519 [7] is used. Ed25519 sets several domain parameters of EdDSA such as:

- Finite field F_q, where $q = 2^{255} - 19$
- Elliptic curve $E(F_q)$, Curve25519
- Base point B
- Order of the point B, l
- Hash function H, SHA-512 [24]
- Key length $b = 256$

For more details on other parameters of Curve25519 and the corresponding curve equations we refer to Bernstein [9].

Table 1. Our notations for EdDSA

Name	Symbol
Private key	k
Private scalar	a (first part of $H(k)$)
Auxiliary key	b (last part of $H(k)$)
Ephemeral scalar	r

To sign a message, the signer has a private key k and message M. Algorithm 1 shows the steps to generate an EdDSA signature.

Algorithm 1. EdDSA key setup and signature generation

Key setup.
1: Hash k such that $H(k) = (h_0, h_1, \ldots, h_{2b-1}) = (a, b)$
2: $a = (h_0, \ldots, h_{b-1})$, interpret as integer in little-endian notation
3: $b = (h_b, \ldots, h_{2b-1})$
4: Compute public key: $A = aB$.
Signature generation.
5: Compute ephemeral private key: $r = H(b, M)$.
6: Compute ephemeral public key: $R = rB$.
7: Compute $h = H(R, A, M)$ and convert to integer.
8: Compute: $S = (r + ha) \mod l$.
9: Signature pair: (R, S).

The first four steps belong to the key setup and are only applied the first time a private key is used. Notation $(x, \ldots, y)$ denotes concatenation of the elements. We call a the private scalar and $b = (h_0, h_1, \ldots, h_{2b-1})$ the auxiliary key (see Table 1). In Step 5 the ephemeral key r is deterministically generated.

To verify a signature (R, S) on a message M with public key A a verifier follows the procedure described in Algorithm 2.

Algorithm 2. EdDSA signature verification

1: Compute $h = H(R, A, M)$ and convert to integer.
2: Check if the group equation $8SB = 8R + 8hA$ in E holds.
3: If group the equation holds, the signature is correct.

2.2 SHA-512

SHA-512 is a member of the SHA-2 hashing family, designed by the NSA. The hash functions from the SHA-2 family are named after their digest length. SHA-512 is used several times in the Ed25519 signature scheme. SHA-2 is based on its predecessor SHA-1 and with SHA-1 being broken, implementations change in their usage of hash function from SHA-1 to SHA-2 or SHA-3 [11].

SHA-2 is a Merkle-Damgård construction that uses a compression function based on a block cipher by adding a feed-forward according to Davies-Meyer, see Algorithm 3. Merkle-Damgård iteratively updates a chaining value (CV), this value is initialized to a fixed initial value (IV). The message is padded and split up into blocks. In each iteration a message block is processed. The digest is the value of the CV after all message blocks have been processed. Figure 1 shows an overview of the generation of the ephemeral scalar where the auxiliary key and the message are hashed. The letter K denotes the auxiliary key b, M_i the input message, w the remaining message schedule words and R the compression function. M_0 is the fragment of the message that is in the same block as the key and M_1 a fragment in the second block. We assume here a relatively short message.

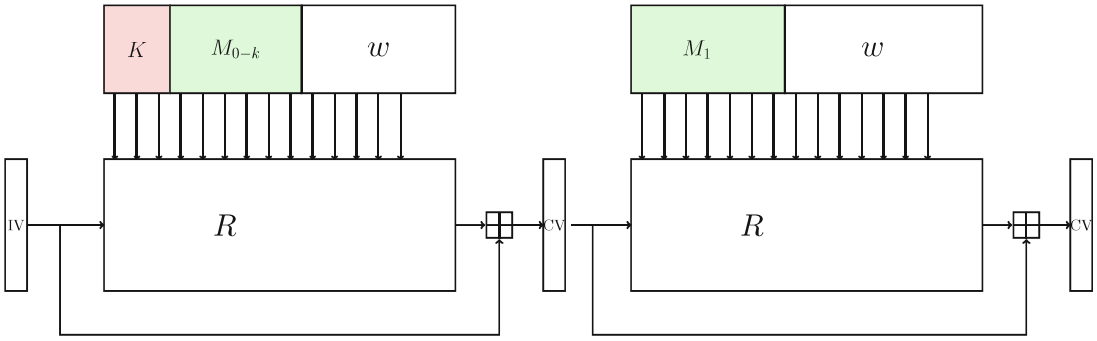


Fig. 1. SHA-512 hashing of K and M.

The compression function has two inputs, the chaining value CV_i and message block M_i. The compression function produces an updated chaining value CV_{i+1}. All the variables in SHA-512 are 64-bit unsigned integers (words). The additions are computed modulo 2^{64}. The algorithm consists of a data path and a message schedule. The data path transforms the CV by iteratively applying 80 rounds on it. The message expansion takes a $16 \times 64 = 1024$-bit message block and expands it to a series of 80 *message schedule words* w_i, each of 64 bits. Each message block consists of 16 64-bit words, that are the first 16 message schedule words. Next, the remaining message schedule words are generated using the 1024-bit message block so there is a word for each round. On a message block 80 rounds are applied, in each round a round constant and a message schedule word is added. As a result a 512-bit message digest is produced.

The compression function of SHA-512 is explained in detail in Algorithm 4 using the notation described in Table 2.

Table 2. Notation for SHA-512

Name	Symbol
Bitwise right rotate	$\ggg$
Bitwise right shift	$\gg$
Bitwise and	$\wedge$
Bitwise xor	$\oplus$
Bitwise not	$\neg$
Addition modulo 2^{64}	$+$
Message schedule word	$w[i]$
Message word	$m[i]$
Message block	$M[i]$
State of the data path	H_i
Compression function	CF

Algorithm 3. Merkle Damgård

Input: Message M with $0 \leq$ bit-length $< 2^{128}$
Output: Hash value of M

1: Pad message M by appending an encoding of the message length
2: Initialize chaining value CV with constant IV
3: Split padded message M into blocks
4: **for all** blocks M_i **do**
5: $\quad CV_{i+1} \leftarrow CF(CV_i, M_i)$
6: **end for**
7: **return** H $\leftarrow$ CV

Algorithm 4. SHA-512 Compression function

Input: CV_i, M_i
Output: $\mathrm{CV}_{i+1} = \mathrm{CF}(\mathrm{CV}_i, M_i)$

Message expansion
for $i = 0; i < 16; i{+}{+}$ **do**
  $w[i] \leftarrow m[i]$
end for
for $i = 16; i < 80; i{+}{+}$ **do**
  $\sigma_0 \leftarrow (w[i\text{-}15] \ggg 1) \oplus (w[i\text{-}15] \ggg 8) \oplus (w[i\text{-}15] \gg 7)$
  $\sigma_1 \leftarrow (w[i\text{-}2] \ggg 19) \oplus (w[i\text{-}2] \ggg 61) \oplus (w[i\text{-}2] \gg 6)$
  $w[i] \leftarrow \sigma_1 + w[i\text{-}7] + \sigma_0 + w[i\text{-}16]$
end for
$H_0, \cdots, H_7 \leftarrow \mathrm{CV}_i$
Copy chaining value to data path
$a \leftarrow H_0, \ldots, h \leftarrow H_7$
for $i = 0; i < 80; i{+}{+}$ **do**
  $\Sigma_1 \leftarrow (e \ggg 14) \oplus (e \ggg 18) \oplus (e \ggg 41)$
  $\Sigma_0 \leftarrow (e \ggg 28) \oplus (e \ggg 34) \oplus (e \ggg 39)$
  $ch \leftarrow (e \wedge f) \oplus ((\neg e) \wedge g)$
  $maj \leftarrow (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$
  $T_1 \leftarrow h + \Sigma_1 + ch + k[i] + w[i]$
  $T_2 \leftarrow \Sigma_0 + maj$
  $h \leftarrow g$
  $g \leftarrow f$
  $f \leftarrow e$
  $e \leftarrow d + T_1$
  $d \leftarrow c$
  $c \leftarrow b$
  $b \leftarrow a$
  $a \leftarrow T_1 + T_2$
end for
Davies-Meyer feed-forward
$H_0 \leftarrow H_0 + a, \ldots, H_7 \leftarrow H_7 + h$
return $\mathrm{CV}_{i+1} \leftarrow H_0, \ldots, H_7$

2.3 Differential Power Analysis

There are different categories of side-channel attacks such as timing attacks, electromagnetic emissions attacks and power attacks, i.e. exploiting different physical information. In this paper we perform a power attack. Power analysis attacks were introduced in 1999 by Kocher et al. [19]. Power attacks exploit the dependency of the power consumption on the data that is processed by a device.

We use a CMOS based micro-controller, so we can model the power consumption by computing the Hamming weight of the assumed intermediate values processed in the device. In our attack, we predict the intermediate values using a selection function. The selection function computes the intermediate value based on a known input, i.e. part of the message and on a hypothesis of an unknown input, part of the key.

In a side-channel attack the adversary typically has to make a hypothesis on all possible candidate values of a subkey. As using the complete key results in an unfeasible amount of key hypotheses, the adversary uses a divide-and-conquer technique by recovering the key in smaller chunks. The size is determined so it is possible to compute the selection function for all possible hypotheses, for instance with a size of 8 bits. We correlate all the Hamming weights of the values processed by the selection function with the traces using the Pearson correlation coefficient. This distinguisher is called Correlation Power Analysis (CPA) [12]. The results are stored in a table. The columns correspond to the time samples, the rows correspond to the key hypotheses. When enough traces are used, the row containing highest absolute correlation value corresponds to the correct key hypothesis.

3 The Attack Components

In this part we elaborate on our strategy and the hierarchy of the attack. Following a top-down approach we examine the Ed25519 signature algorithm looking for vulnerabilities. The way it is composed leads us to identifying the weakness of the modular addition operation in the SHA-512 part.

We start off by explaining what value we need to recover from Ed25519 and how to use it to generate forged signatures. Next, we explain how we recover this value by attacking SHA-512. Finally, we apply DPA on modular addition. To reduce the complexity of the attack we use a divide-and-conquer technique to divide 64-bit key words into 8 bit substrings.

3.1 Attacking Ed25519

We describe a key-recovery attack on Ed25519 by measuring the power consumption of 4 000 signature computations.

We attack the generation of the ephemeral key to retrieve the auxiliary key b. This allows us to compute the ephemeral key r. Once we know the auxiliary key, we extract the private scalar by applying the following computations on an arbitrary signature performed with the key.

1. Compute $r = H(b, M)$.
2. Compute $h = H(R, A, M)$.
3. Compute $a = (S - r)h^{-1} \mod l$.

We can use the private scalar a with any message and any auxiliary key b to generate forged signatures. This is because r, in signature verification is only used in R which is part of the signature.

3.2 Attack on SHA-512

The auxiliary key is prepended to the message and together this is hashed to compute the ephemeral key. In our attack we assume the message has at least

length 512 bits. In this way the first 4 message schedule words contain the constant auxiliary key, the next 8 words contain the variable message and the remaining 4 words can contain more message words or constant padding.

To be able to attack the auxiliary key b, we are looking for steps in the algorithm where a word that only depends on the message part of the input block is added to a constant unknown key-dependent word. If we look at Algorithm 4, we can see that elements from the message schedule are added in two places, namely in message schedule line 7 and in data path line 16. The extending of the message schedule in line 7 seemed like a viable option, as from round 16 to 19 unknown words are added to known words. It depends on the implementation how this can be attacked.

The implementation that we attacked is in WolfSSL, which is a lightweight C-based TLS/SSL library that targets IoT and embedded devices. To determine how to attack the implementation and how to model the leakage we looked at the computation of $w[16]$ in the message schedule, see Fig. 2. The figure shows a single step to compute the message schedule that is applied recursively to the remaining words.

$$w[16] \leftarrow \sigma_1(w[14]) + w[9] + \sigma_0(w[1]) + w[0] \tag{1}$$

σ_0 and σ_1 apply linear transformations that transform a word by taking a word, shifting it over three different offsets and XOR'ing these. They do not play a role in our attack. Of these four words on the right hand side of (1), word $w[14]$

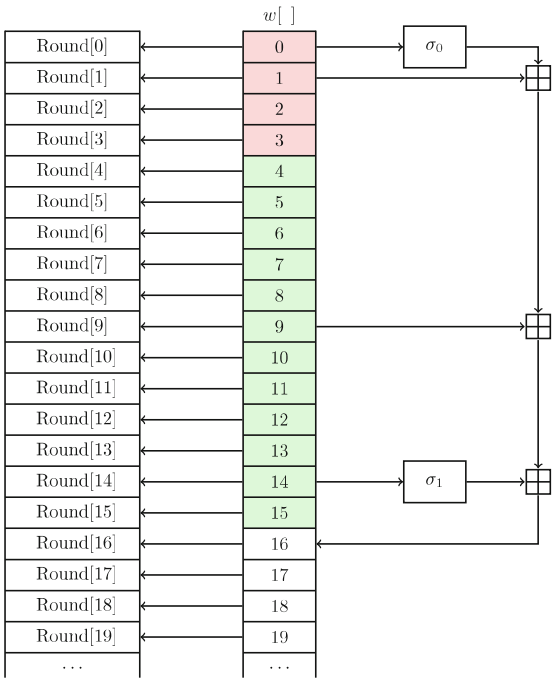


Fig. 2. Single step of message schedule SHA-512

and $w[9]$ are part of the message therefore variable and known (green). Word $w[1]$ and $w[0]$ are part of the auxiliary key value so constant and unknown (red). With the attack we are able to recover $\sigma_0(w[1]) + w[0]$. To be able to recover the words separately, we introduce 4 auxiliary variables that represent the key-dependent part of the message expansion word computation. Knowledge of these four variables allows reconstructing the key.

$$w[17] \leftarrow \sigma_1(w[15]) + w[10] + \sigma_0(w[2]) + w[1] \tag{2}$$
$$w[18] \leftarrow \sigma_1(w[16]) + w[11] + \sigma_0(w[3]) + w[2] \tag{3}$$
$$w[19] \leftarrow \sigma_1(w[17]) + w[12] + \sigma_0(w[4]) + w[3] \tag{4}$$

We call the unknown parts $k_{16}, \ldots, k_{19}$, corresponding to the message schedule entries $w[16], \ldots, w[19]$ respectively.

$$k_{19} = w[3] \tag{5}$$
$$k_{18} = \sigma_0(w[3]) + w[2] \tag{6}$$
$$k_{17} = \sigma_0(w[2]) + w[1] \tag{7}$$
$$k_{16} = \sigma_0(w[1]) + w[0] \tag{8}$$

Equation (3) uses the result of (1). Since we can obtain k_{16}, we can compute $w[16]$ and consider it to be known. This also applies to (4). In (4), $w[19]$ only uses one unknown word as input, so $k_{19} = w[3]$. Once we know $w[3]$, there is only one unknown word in (7), word $w[2]$. Thus we can compute it. The remaining unknown words are computed in a similar way. The words $w[0], \ldots, w[3]$ correspond to auxiliary key $b = (h_b, \ldots, h_{2b-1})$.

3.3 DPA on Modular Addition

To attack a full addition we need to guess 64 unknown bits. This leaves us with 2^{64} possible candidates. As it is not feasible to correlate the traces with this number of key candidates, we apply a divide-and-conquer strategy similar to the one in [27]. We pick an 8-bit part of the computation result called the sensitive variable.

We start the attack on a 64-bit word with the least significant 8 bits of the words. We craft the selection function $S(M, k^*)$ as follows for k_{16}, where M is part of the input message $(w[9], w[14])$ and k^* is the key byte we make a hypothesis on.

$$S(M, k^*)_{k_{16},\ \text{bit}\ 0-7} \leftarrow ((\sigma_1(w[14]) + w[9]) \mod 2^8) + k^* \tag{9}$$

Next, we create the table V containing all possible intermediate values by adding $k^* \in \{0, \ldots, 255\}$ to each 8-bit message. The addition of k^* is not reduced by 2^8, that means the intermediate values have a length of at most 9 bits. The trace set contains T traces, each trace consists of N time samples and there are 256 key

candidates. With table V we model the power consumption by computing the Hamming Weight of each intermediate value and store them in table $H = T \times K$. To find the correct key candidate we compute the Pearson correlation of each column of traces with each column of H. The result is stored in table $R = K \times N$. When a sufficient amount of traces is used, the row with the highest absolute value corresponds to the correct key candidate. We store the value in k'_{16} (the recovered key bits) with the remaining bits 0.

When we know the least significant byte of k_{16} by applying the attack, we use it to obtain the next byte as follows.

$$S(M, K^*)_{k_{16},\ \text{bit}\ 8-15} \leftarrow (((\sigma_1(w[14]) + w[9] + k'_{16}) \gg 8) \mod 2^8) + k^*$$

We add k'_{16} to the messages, shift the result 8 bits to the right and compute modulo 2^8 such that the MSB of the previous result is taken into account. We compute the previous steps again and store the key corresponding to the highest correlation value in k'_{16}. We repeat these steps to obtain the remaining 6 bytes of k_{16}. The remaining words of the auxiliary key, k_{17}, k_{18} and k_{19} are obtained in a similar way as k_{16}.

4 Experimental Setup and Results

4.1 Setup

For our attack we use the Piñata[2] development board by Riscure as our target. The CPU on the board is a Cortex-M4F, working at a clock speed of 168 MHz. The CPU has a 32-bit Harvard architecture with a three-stage pipeline. The board is programmed and modified such that it can be targeted for SCA.

The target is the Ed25519 code of WolfSSL 3.10.2.

The physical leakage of the device that we exploit is the dependency of the current to the data it is processing. To measure this we use a device called the Current Probe[3] by Riscure. The Current Probe provides us with a clean signal we can exploit.

The oscilloscope we use to measure the output of the Current Probe is a Lecroy Waverunner z610i. The oscilloscope is triggered by an I/O pin on the Piñata. We set the pin to a high signal just before SHA-512 is called and to a low signal right after it finishes. Although the clock speed of the CPU is 168 MHz, the oscilloscope is set to sample at a rate of 250 MS/s. With these settings we captured the traces that we attacked. Figure 3 shows a photo of the setup.

[2] Piñata board. Accessed: 18-04-2017. Url: https://www.riscure.com/security-tools/hardware/pinata-training-target.

[3] Current Probe. Accessed: 18-04-2017. Url: https://www.riscure.com/benzine/documents/CurrentProbe.pdf.

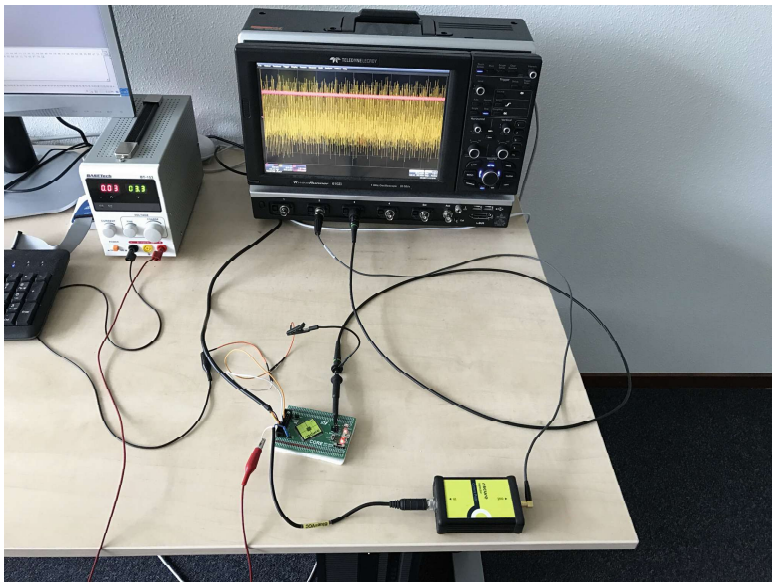


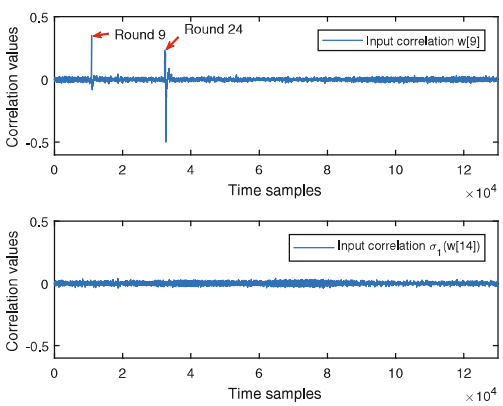
Fig. 3. Setup

4.2 Input Correlation

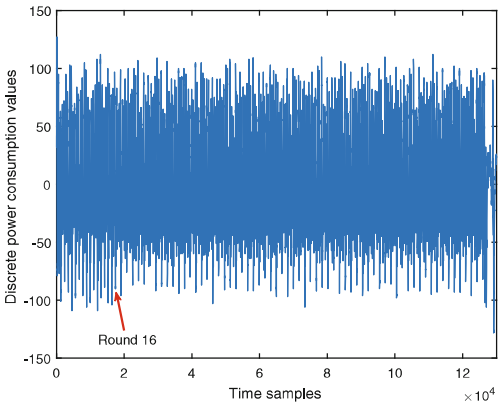
To determine where the computations leak we compute the correlation of values that we know and that are going to be used in the sensitive variable. If we look at Fig. 4a, we see the correlation of the measured power consumption with the Hamming weight of $w[9]$. The same approach was applied for $\sigma_1(w[14])$. For $w[9]$ we observe peaks in the correlation and for $\sigma_1(w[14])$ we only observe noise. The value $w[9]$ is directly loaded from the memory to a register while $\sigma_1(w[14])$ is not loaded from the memory, but $w[14]$ is and has the linear computation σ_1 applied afterwards. We only observe correlation with values directly loaded from the memory. This lead us to the conclusion that the memory bus provided us with the highest observed leakage.

If we look at Fig. 4b we see a power trace of the compression function computation where the message expansion is computed. Each negative peak corresponds to a round. The first 16 rounds are shorter as in WolfSSL the message schedule does not happen before the compression rounds start, but on the fly. The time samples in Fig. 4b correspond to time samples in Fig. 4a, thus we can relate the peaks to the round where they appear. The first peak is when word $w[9]$ is used in the round function at round 9 and the second peak at round 24 when $w[9]$ is used to compute $\sigma_0(w[9])$. There is no input correlation at round 16. The value could be cached and therefore does not appear on the memory bus.

Since the Hamming weight of values on the memory bus provide the best leakage, we choose to attack values that are loaded or stored from a register to the memory or visa versa. That means in (1), $w[16]$ leaks and from that we can recover k_{16}.



(a) Input correlation of $\sigma_1(w[14])$ and $w[9]$.



(b) Power trace of compression function computation.

Fig. 4. Input correlation and power trace figures

4.3 Results of the Attack

In Fig. 5 we see the correlation of the correct key candidate with the traces. Peaks are visible corresponding to the rounds when the value is stored and loaded. The figure also shows the correlation result for an incorrect key candidate where no correlation occurs.

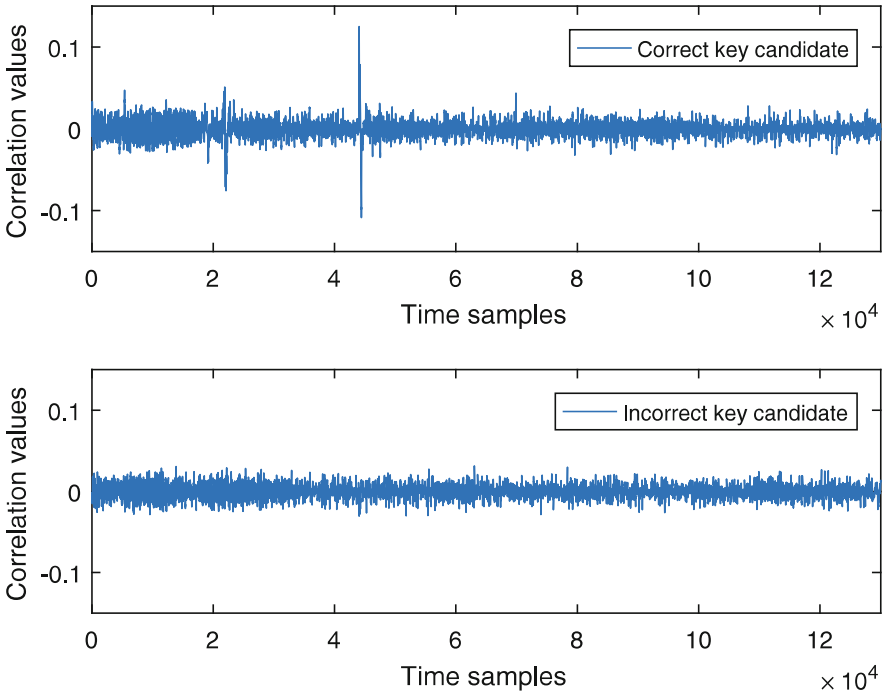


Fig. 5. Pearson correlation of a correct and an incorrect key candidate.

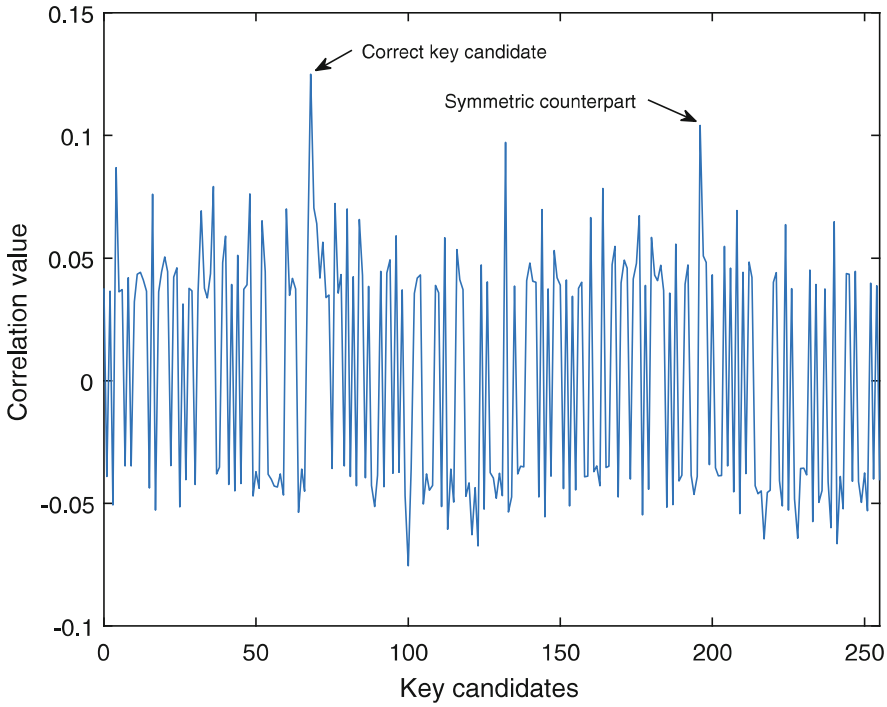


Fig. 6. Correlation result of the least significant byte of k_{16}, with correct key candidate 68.

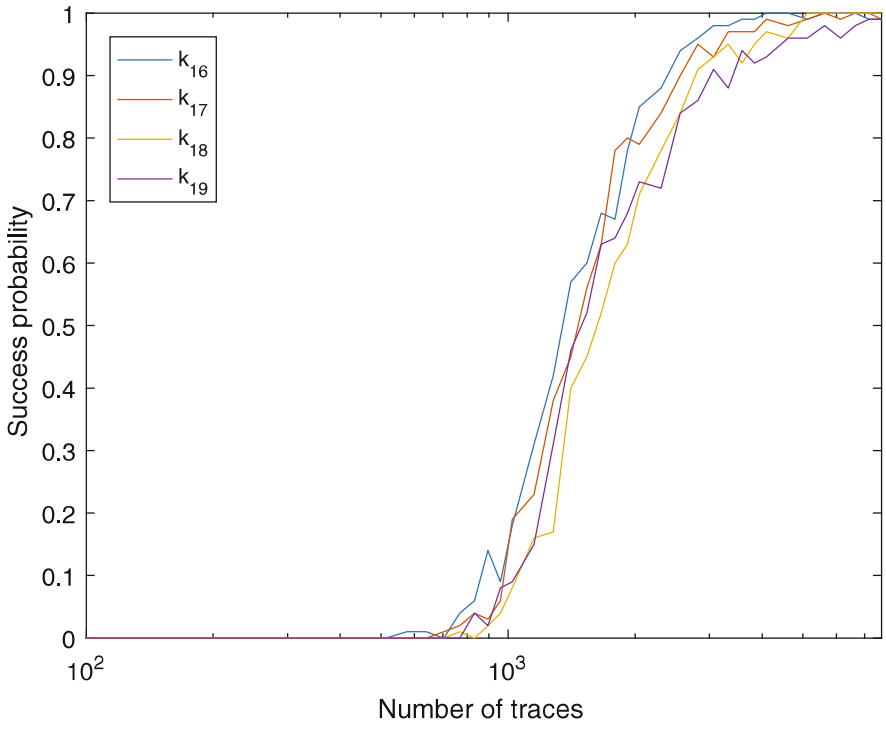


Fig. 7. Success probability of the attack

When we plot the highest correlation value for each key candidate we see a similar effect as in other attacks on modular addition where the Pearson correlation is also used. We also see high correlation values for the symmetric counterpart of the correct key candidate. In Fig. 6 we can observe this with high peaks for the correct key candidate 68 and for its symmetric counter part key candidate 196. In the symmetric counterpart of the key candidate only the most significant bit is different. As all papers describing an attack on modular addition mention this symmetry it seems unavoidable. Compared to the work [27] we can clearly distinguish the correct key candidate from the incorrect ones.

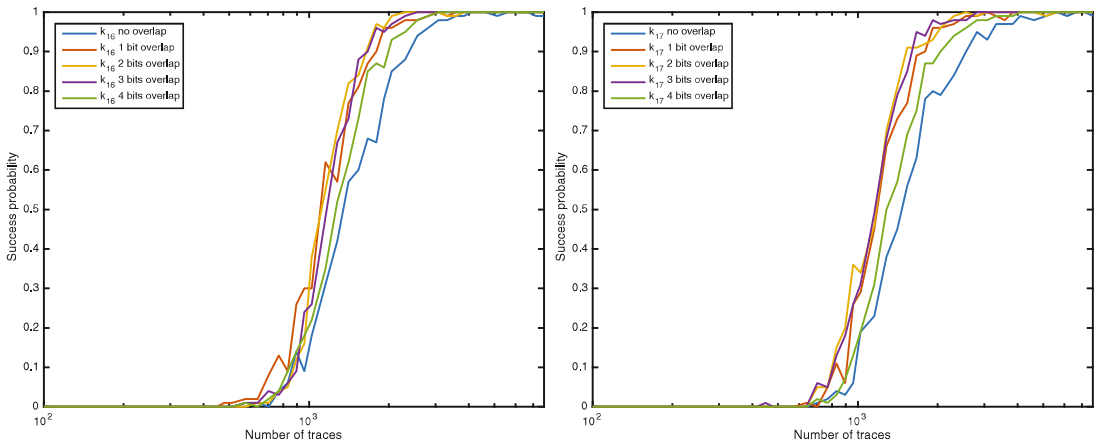
In Fig. 7 we see the success probabilities of the attack on the unknown words $k_{16}, \ldots, k_{19}$. For each data point in the figure we ran the attack 100 times with a certain amount of traces. In Fig. 7, the attack was considered successful if all 64 bits of a word were recovered correctly by applying the attack on a byte 8 times. The figure shows that the success probability of the attack rapidly increases when more than 1000 traces are used. At around 4 000 traces the success probability approaches one making this a practical attack.

4.4 Reducing the Number of Traces

Although we can clearly distinguish the correct key candidate from Fig. 6, we use the symmetry of the result to increase the success probability of our attack such that less traces are required for a successful attack. The most significant bit is the hardest to attack and requires the highest number of traces to distinguish. If we overlap the bytes that we attack by one bit, the most significant bit in one attack will be the least significant bit in the next attack. Using this overlap technique we find all bits of a word except for the most significant bit. In the attack on Ed25519 we attack four words, that means we need to brute force four bits, so 16 possibilities. We do this by recomputing a valid signature with each possible key. We compare the computed signatures with the valid one we have, the key corresponding to the valid signature is the correct one.

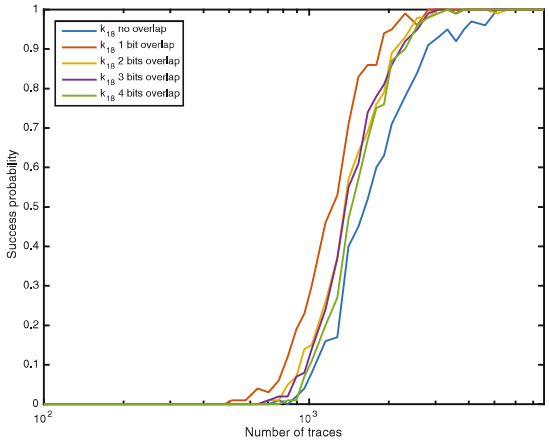
We also overlapped the result with more bits. With 2, 3 and 4 bits overlap we need to brute force four bits for each word. This means we need to brute force 2^{16} possibilities.

Figure 8 shows the results of the different overlap sizes for the different words that we need to attack to recover the key. As we can see, overlapping bits results in a higher success probability. The difference between the amount of overlapped bits seems minimal and not consistent for each word. We already saw that we needed the highest amount of traces to distinguish the most significant bit correctly. Any amount of overlapping bits at least overlaps with the most significant bit. This causes the largest increase of the success probability. Overlapping a larger number of bits does not seem to affect the success probability relevantly.

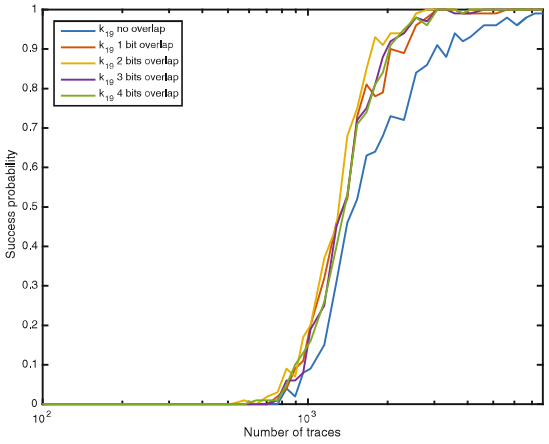


(a) k_{16}

(b) k_{17}



(c) k_{18}



(d) k_{19}

Fig. 8. Success probability of the attack with overlap.

5 Discussion and Countermeasure

With our presented attack, we are able to obtain the private scalar such that we can forge signatures by collecting the power measurements of only 4 000 signatures. This makes it a very practical attack and implementers of Ed25519 should take this into account.

The default protection would be the implementation of a protected version of SHA-512. Due to the use of boolean and arithmetic operations, the protection of SHA-1, SHA-2 and ARX algorithms in general is complex and could be quite costly [16,21]. We have an alternative proposal that requires dropping the deterministic signature feature and adding some randomness in the computation of the ephemeral scalar. We need to create a scenario such that an attacker is not able to make a hypothesis on the constant key value. This can be achieved by padding the key with fresh random bits such that the first 1024-bit block is composed only by key and random value, without any bits known to the attacker.

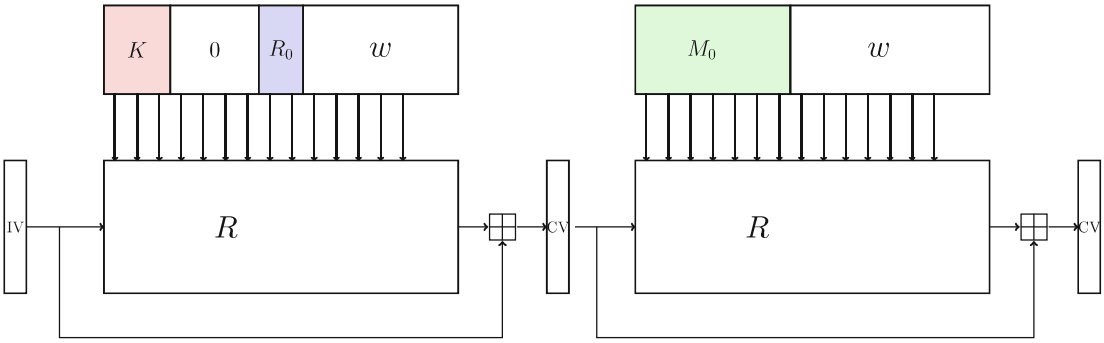


Fig. 9. Generation of the ephemeral key with a countermeasure.

The input message will be processed in blocks after that. Figure 9 visualizes how the input should look. The R_0 block would be a random number of 768 bits. We argue that it is also possible to have an R_0 block composed by 128 bits of randomness and pad the rest of the block with 640 bits with a constant value (e.g. all zero).

The XEdDSA and VXEdDSA [2] signature schemes extend Ed25519 to generate a more robust ephemeral private scalar that is sufficiently random. Although XEdDSA and VXEdDSA also add random values into the signature scheme, XEdDSA is still vulnerable to our attack. As they append a random 64-byte sequence to the key and the message, the vulnerability that we exploit remains the same. VXEdDSA is not vulnerable to our attack but it requires several additional scalar multiplications that add to the computation time.

Obviously, this countermeasure kills the deterministic signature properties, but we do not see this as a dramatic problem. The main motivation for the proposal of deterministic signatures was to avoid a poor management of randomness that can introduce security problems [14,17]. The proposed countermeasure is also not re-introducing the strong security requirement of randomness needed by ECDSA. Basically, even if the same randomness is used to sign two different messages, the attacker will not be able to recover the key as it would be possible with ECDSA. Additionally we want to highlight that the signature verification procedure remains as is.

As our final comment, in the recent developments of the IETF CFRG group for TLS 1.3, the hash function adopted for Ed448 is SHAKE256. In this case the protection against side-channel attacks such as power and EM based would be easier and pretty robust as explained by Chari et al. [13].

6 Conclusion

In this work we presented a side-channel attack on the digital signature scheme Ed25519. By measuring the power consumption of approximately 4 000 signatures we were able to recover the auxiliary key of a signature. We can use the auxiliary key to recover the private scalar that we can use to forge signatures.

We recover the auxiliary key by executing a side-channel attack on SHA-512. We described an attack on the message schedule that is applicable to all applications where a constant secret is hashed together with a variable known input, if the length of the secret is shorter then the block size.

The attack we presented poses a real threat to implementation of the signature scheme such as on embedded devices or devices in IoT, if an attacker is able to measure the power consumption. Additionally, we propose a countermeasure to counteract against this attack.

Acknowledgments. This work was supported in part by a project funded by DarkMatter LLC.

References

1. ECRYPT II key recommendations (2012). https://www.keylength.com/en/3/
2. The XEdDSA and VXEdDSA Signature Schemes (2017). https://signal.org/docs/specifications/xeddsa/xeddsa.pdf. Accessed 11 Sept 2017
3. Things that use Ed25519 (2017). https://ianix.com/pub/ed25519-deployment.html. Accessed 29 Sept 2017
4. Ambrose, C., Bos, J.W., Fay, B., Joye, M., Lochter, M., Murray, B.: Differential attacks on deterministic signatures. Cryptology ePrint Archive, report 2017/975 (2017). https://eprint.iacr.org/2017/975.pdf
5. Belaid, S., Bettale, L., Dottax, E., Genelle, L., Rondepierre, F.: Differential power analysis of HMAC SHA-2 in the Hamming weight model. In: 2013 International Conference on Security and Cryptography (SECRYPT), pp. 1–12. IEEE (2013)
6. Benoît, O., Peyrin, T.: Side-channel analysis of six SHA-3 candidates. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 140–157. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15031-9_10
7. Bernstein, D.J.: Curve25519: new Diffie-Hellman speed records. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 207–228. Springer, Heidelberg (2006). https://doi.org/10.1007/11745853_14
8. Bernstein, D.J., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted edwards curves. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 389–405. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68164-9_26
9. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. J. Cryptographic Eng. **2**(2), 77–89 (2012)
10. Bernstein, D.J., Lange, T.: Faster addition and doubling on elliptic curves. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 29–50. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76900-2_3
11. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak reference (2011). http://keccak.noekeon.org/Keccak-reference-3.0.pdf, http://keccak.noekeon.org/
12. 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
13. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_26

14. Checkoway, S., Maskiewicz, J., Garman, C., Fried, J., Cohney, S., Green, M., Heninger, N., Weinmann, R.P., Rescorla, E., Shacham, H.: A systematic analysis of the juniper dual EC incident. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 468–479. ACM (2016)
15. Edwards, H.M.: A normal form for elliptic curves. Bull. Am. Math. Soc. **44**(03), 393–423 (2007). https://doi.org/10.1090/s0273-0979-07-01153-6
16. Goubin, L.: A sound method for switching between Boolean and arithmetic masking. In: Proceedings of Third International Workshop Cryptographic Hardware and Embedded Systems - CHES 2001, Paris, France, 14-16 May 2001, pp. 3–15 (2001)
17. Hastings, M., Fried, J., Heninger, N.: Weak keys remain widespread in network devices. In: Proceedings of the 2016 ACM on Internet Measurement Conference, pp. 49–63. ACM (2016)
18. Koblitz, N.: Elliptic curve cryptosystems. Math. Comput. **48**(177), 203–209 (1987)
19. 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
20. Lemke, K., Schramm, K., Paar, C.: DPA on n-bit sized boolean and arithmetic operations and its application to IDEA, RC6, and the HMAC-construction. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 205–219. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_15
21. McEvoy, R., Tunstall, M., Murphy, C.C., Marnane, W.P.: Differential power analysis of HMAC based on SHA-2, and countermeasures. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 317–332. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77535-5_23
22. Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-39799-X_31
23. Nguyen, P.Q., Shparlinski, I.E.: The insecurity of the elliptic curve digital signature algorithm with partially known nonces. Des. Codes Cryptogr. **30**(2), 201–217 (2003). https://doi.org/10.1023/A:1025436905711. ISSN: 1573-7586
24. Pub, F.: Secure hash standard (SHS). Technical report, NIST, July 2015
25. Schnorr, C.P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991). http://dx.doi.org/10.1007/BF00196725
26. Seuschek, H., Heyszl, J., De Santis, F.: A cautionary note: side-channel leakage implications of deterministic signature schemes. In: Proceedings of the Third Workshop on Cryptography and Security in Computing Systems, CS2 2016, pp. 7–12. ACM, New York (2016). http://doi.acm.org/10.1145/2858930.2858932
27. Zohner, M., Kasper, M., Stöttinger, M.: Butterfly-attack on Skein's modular addition. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 215–230. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29912-4_16