Differential Power Analysis of XMSS and SPHINCS

Matthias J. Kannwischer[1,2], Aymeric Genêt[3,4], Denis Butin[1], Juliane Krämer[1], and Johannes Buchmann[1]

[1] TU Darmstadt, Darmstadt, Germany
{dbutin,jkraemer,buchmann}@cdc.informatik.tu-darmstadt.de
[2] University of Surrey, Guildford, UK
m.j.kannwischer@surrey.ac.uk
[3] EPFL, Lausanne, Switzerland
aymeric.genet@epfl.ch
[4] Kudelski Group, Cheseaux-sur-Lausanne, Switzerland
aymeric.genet@nagra.com

Abstract. Quantum computing threatens conventional public-key cryptography. In response, standards bodies such as NIST increasingly focus on post-quantum cryptography. In particular, hash-based signature schemes are notable candidates for deployment. No rigorous side-channel analysis of hash-based signature schemes has been conducted so far. This work bridges this gap. We analyse the stateful hash-based signature schemes XMSS and XMSS^{MT}, which are currently undergoing standardisation at IETF, as well as SPHINCS — the only practical stateless hash-based scheme. While timing and simple power analysis attacks are unpromising, we show that the differential power analysis resistance of XMSS can be reduced to the differential power analysis resistance of the underlying pseudorandom number generator. This first systematic analysis helps to further increase confidence in XMSS, supporting current standardisation efforts. Furthermore, we show that at least a 32-bit chunk of the SPHINCS secret key can be recovered using a differential power analysis attack due to its stateless construction. We present novel differential power analyses on a SHA-2-based pseudorandom number generator for XMSS and a BLAKE-256-based pseudorandom function for SPHINCS-256 in the Hamming weight model. The first attack is not threatening current versions of XMSS, unless a customised pseudorandom number generator is used. The second one compromises the security of a hardware implementation of SPHINCS-256. Our analysis is supported by a power simulator implementation of SHA-2 for XMSS and a hardware implementation of BLAKE for SPHINCS. We also provide recommendations for XMSS implementers.

Keywords: Post-quantum cryptography · Hash-based signatures · DPA

1 Introduction

Due to the wide applicability of Shor's algorithm [29], conventional public-key cryptography (e.g., RSA, DSA, and ECDSA) is vulnerable to attacks using

quantum computers. Some cryptographic schemes, known as *post-quantum* [7], are believed to remain safe in the presence of quantum computers. Post-quantum cryptography was already introduced in the 70s, but not deployed at that time. Engineering progress in quantum computing [21] is creating a new sense of urgency. Current standardisation efforts — for instance at NIST [27] and IETF [14] — signal a shift towards real-world use [8]. It is therefore important to further analyse the security of candidate schemes.

In particular, the side-channel resistance of hash-based signature (HBS) schemes has not been evaluated systematically so far. HBS schemes rely on the security of an underlying hash function, and use a binary hash tree structure. While these schemes are conjectured to be "naturally" side-channel resistant [14], a deeper look is desirable to uncover potential weaknesses and increase confidence in them. We provide a side-channel analysis (SCA) of two prominent HBS schemes: XMSS (including its variant XMSS^{MT}) and SPHINCS. We chose them because XMSS is being standardised, SPHINCS is the only practical *stateless* HBS scheme (see Sec. 2 for an explanation of statefulness), and both are recommended by the PQCRYPTO EU project [28].

1.1 Related Work

The side-channel resistance of HBS schemes is rarely addressed. Eisenbarth et al. [11] investigate the side-channel leakage of a customised Merkle-based HBS scheme. Leakage experiments using an AES-based hash function are performed. XMSS is not directly analysed, and only a brief SCA is provided.

For other categories of post-quantum cryptography, SCAs are mainly available for implementations of lattice-based and code-based schemes. In particular, the NTRUEncrypt [20,30] and McEliece [22] schemes have been thoroughly examined. Several differential power analysis (DPA) attacks have been proposed on hash-based message authentication codes (HMACs) based upon SHA-2 [2, 23] and SHA-3 [32, 33]. However, none of them directly applies to HBS.

We only address purely passive attacks. The fault attack vulnerability of SPHINCS was recently analysed by Castelnovi et al. [9].

1.2 Outline

The remainder of the paper is organised as follows. We start by recalling basics about the schemes under consideration and the more elementary schemes they rely upon: W-OTS$^+$, XMSS, XMSS^{MT}, and SPHINCS (Sec. 2). We next analyse the side-channel resistance of XMSS and XMSS^{MT} (Sec. 3) and describe a DPA on a SHA-2-based pseudorandom number generator (PRNG) which applies to both schemes. SPHINCS-256 is then analysed in the same respect (Sec. 4); we introduce a novel DPA on a BLAKE-256-based pseudorandom function (PRF). Impact analyses and discussions of implications for implementers appear in both main sections. We then conclude (Sec. 5).

2 XMSS, XMSS^{MT} and SPHINCS

In HBS schemes, many one-time signature key pairs are combined into a single structure, using a binary hash tree. Numerous improvements upon seminal constructions by Lamport [19] and Merkle [25] have culminated in modern schemes such as XMSS [4], its hierarchical variant XMSS^{MT} [15] and SPHINCS [3].

These are the schemes analysed in this paper; in particular, for XMSS and XMSS^{MT}, we examine the recently proposed IETF standard [14]. XMSS has minimal security requirements, since it only requires a second-preimage resistant hash function for its security. XMSS and XMSS^{MT} are *stateful*: after signing, the secret key is updated. If this update is not carried out properly, the security of the cryptographic scheme degrades or vanishes. As a result, extra care is required [24]. Stateful hash-based signature schemes are particularly suited to the use case of software update authentication, where signing frequency is low. SPHINCS is conveniently stateless, but its signatures are significantly larger and speed also suffers.

We start by recalling a one-time signature scheme which is typically not used on its own, but constitutes a cornerstone of these three schemes: W-OTS$^+$ [13]. Due to space limitations, we only describe parts of the schemes relevant for SCA. Self-contained algorithm descriptions and security proofs can be found in the original papers.

2.1 W-OTS$^+$

W-OTS$^+$ improves upon the W-OTS [10]. It is parametrised by the Winternitz parameter $w = 2^\omega$, which enables a time/space trade-off. Large values of w yield small keys and signatures, but slow down the scheme. Given a keyed hash function $f_k : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, W-OTS$^+$ defines the chaining function c_k^i :

$$c_k^0(x, \mathbf{r}) = x, \qquad c_k^i(x, \mathbf{r}) = f_k(c_k^{i-1}(x, \mathbf{r}) \oplus r_i), \quad \mathbf{r} = (r_1, \ldots, r_j), j > i.$$

We recall the W-OTS$^+$ key generation and signature generation algorithms, which involve secret information and are, thus, relevant for SCA.

W-OTS$^+$ key generation Given the security parameter n and length ℓ, the secret key $\mathbf{X} = (x_0, \ldots, x_{\ell-1}) \in_R \{0,1\}^{n \times \ell}$, the randomisation bitmasks $\mathbf{r} = (r_1, \ldots, r_{w-1}) \in_R \{0,1\}^{n \times w-1}$ and the key $k \in_R \{0,1\}^n$ are chosen uniformly at random. The public key $\mathbf{Y}$ is computed from $\mathbf{X}$ by applying c_k $(w-1)$ times:

$$\mathbf{Y} = (y_0, \ldots, y_{\ell-1}) \in \{0,1\}^{n \times \ell}, \quad y_i = c_k^{w-1}(x_i, \mathbf{r}), \quad 0 \le i < \ell.$$

The secret key is $\mathbf{X}$ and the public key is $(\mathbf{Y}, \mathbf{r}, k)$. To compress the public key, $\mathbf{r}$ and k can also be replaced by an n-bit seed to generate $\mathbf{r}$ and k pseudorandomly.

W-OTS$^+$ signature generation Given the secret key $\mathbf{X}$ and the digest $D \in \{0,1\}^n$ of a message M, the digest D is divided into ℓ_1 blocks of ω bits each: $D = b_{\ell-1} \,||\, \ldots \,||\, b_{\ell-\ell_1}$. Using D, a checksum $C = b_{\ell_2-1} \,||\, \ldots \,||\, b_0$ is calculated. The blocks $b_{\ell-1}, \ldots, b_0$ are then used to calculate the signature:

$$\sigma_{W\text{-}OTS^+} = \left(c_k^{b_{\ell-1}}(x_{\ell-1}, \mathbf{r}), \ldots, c_k^{b_1}(x_1, \mathbf{r}), c_k^{b_0}(x_0, \mathbf{r})\right).$$

2.2 XMSS

XMSS is a stateful digital signature scheme built upon the one-time signature scheme W-OTS (or its optimised version W-OTS$^+$) as a building block. XMSS was introduced by Buchmann et al. in 2011 [4]; it is EU-CMA secure, forward-secure and efficient.

Given the security parameter n, XMSS requires a cryptographic hash function $h : \{0,1\}^{2n} \to \{0,1\}^n$. Denoting H the XMSS tree height, up to 2^H messages can be signed, using as many W-OTS$^+$ key pairs.

XMSS key generation Given H, the key generation algorithm first generates 2^H W-OTS$^+$ key pairs $(\mathrm{sk}_{\text{W-OTS}^+,i}, \mathrm{pk}_{\text{W-OTS}^+,i})$, where $0 \leq i < 2^H$. The W-OTS$^+$ public keys are then used to construct an XMSS tree. The inner nodes of the XMSS tree are computed as

$$v_h[j] = h\left((v_{h-1}[2j] \oplus b_{l,h}) \,||\, (v_{h-1}[2j+1] \oplus b_{r,h})\right),$$

where $b_{l,h}$ and $b_{r,h}$ are *public* randomisation elements derived from a public seed using a PRNG. Each leaf of the XMSS $v_0[i]$ $(0 \leq i < 2^H)$ tree is derived from the corresponding W-OTS$^+$ public keys using another XMSS tree, which is called *L-tree*. An L-tree compresses an $n \times \ell$ bit public key to a single n bit value using the same construction for the inner nodes. Since ℓ is not a power of 2 in general, the rightmost leaves of the L-tree are lifted up to form a binary tree.

The XMSS public key is the root of the XMSS tree $v_h[0]$ and the public seed required to generate the randomisation elements in W-OTS$^+$, the XMSS tree, and the L-trees. The XMSS secret key is comprised of all W-OTS$^+$ secret keys $\mathrm{sk}_{\text{W-OTS}^+}$ and the index s of the next unused leaf (initially $s = 0$). Since storing all W-OTS$^+$ secret keys results in an enormous key ($2^H \cdot \ell \cdot n$ bits), it is recommended to use a PRNG to generate them and just store the n-bit seed.

XMSS signature generation Given the secret key $(\mathrm{sk}_{\text{W-OTS}^+}, s)$ and the digest $D \in \{0,1\}^n$ of a message M, XMSS first computes the W-OTS$^+$ signature $\sigma_{W\text{-}OTS^+}$ for M using $\mathrm{sk}_{\text{W-OTS}^+,s}$. It is imperative to increment s in the XMSS secret key to ensure that this one-time key pair is not used again in subsequent signature generations. In addition to $\mathrm{pk}_{\text{W-OTS}^+,s}$, the verifier requires several nodes of the XMSS tree to reconstruct the root of the hash tree. This is achieved by appending the authentication path $A_s = (a_0, \ldots, a_{h-1})$ to the signature, which contains one node in each layer of the hash tree. The a_h are either left or right neighbours of the nodes in the path from $v_0[s]$ to $v_h[0]$:

$$a_h = \begin{cases} v_h[s/2^h - 1], & \text{if } \lfloor s/2^h \rfloor \equiv 1 \pmod 2 \\ v_h[s/2^h + 1], & \text{if } \lfloor s/2^h \rfloor \equiv 0 \pmod 2. \end{cases}$$

The XMSS signature is, thus, $\sigma = (s, \sigma_{\text{W-OTS}^+}, \text{pk}_{\text{W-OTS}^+,s}, A_s)$.

2.3 XMSS^{MT}

While optimised implementations of XMSS provide sufficient performance during signature generation and signature verification, key generation is slow for high trees, e.g., $H > 20$. Since this is problematic in some use cases, an extension of XMSS was proposed using multiple layers of XMSS trees. This tree chaining idea was initially used in the CMSS scheme [6]. Combined with improved distributed signature generation, it resulted in the XMSS^{MT} scheme [15]. It is also specified in the Internet-Draft by Hülsing et al. [14].

A hyper-tree is used. Its upper layers are used to sign the roots of the layers below, and only the lowest layer is used to actually sign messages. Thus, an XMSS^{MT} hyper-tree consists of $T \geq 2$ layers of XMSS trees with heights $H_0, \ldots, H_{T-1}$, where H_0 is the height of the trees at the lowest level. The Internet-Draft further restricts the heights to be equal, i.e., $H_0 = H_1 = \ldots = H_{T-1}$. The W-OTS$^+$ key pairs corresponding to the leaves of layer i are used to sign the roots of the trees on layer $i-1$. The root of layer $T-1$ is the XMSS^{MT} public key. Using XMSS^{MT} is especially sensible if a large number of messages is to be signed. In that case, the use of a PRNG is recommended. Otherwise, the required storage and slow random number generation outweigh the performance gain of XMSS^{MT}.

2.4 SPHINCS

In 2014, Bernstein et al. introduced SPHINCS [3], a practical *stateless* HBS scheme. SPHINCS uses components as XMSS^{MT}, but includes a layer of few-times signatures named HORST beneath the extended Merkle multi-trees, whose instances are pseudorandomly selected to sign messages.

SPHINCS-256 The SPHINCS authors have suggested a standard instantiation for their scheme in [3] that achieves 128 bits of post-quantum security. This instance is called SPHINCS-256 and requires two secret keys (sk_1, sk_2) of 32 bytes each. Since SPHINCS-256 is stateless, the hash-based instances within the scheme are referred to with an addressing scheme. The i^{th} W-OTS$^+$ instance at the leaf of the j^{th} sub-tree at layer l is addressed with the binary concatenation of all these indices, i.e., $A(i, j, l) = (l \mid\mid j \mid\mid i)$ where the first 4 most significant bits refer to layer l, the next 55 bits to sub-tree j, and the last 5 bits its leaf i.

To sign a given message M, a pseudorandom value R is generated according to M and sk_2. This value represents the selected branch of our hyper-tree, i.e., it allows the computation of the HORST instance address A_{HORST} and the W-OTS$^+$ addresses A_i^l at each layer $0 \leq l < T$ and for each leaf $0 \leq i < 2^{H_i}$. The

secret seeds of these instances are computed using this address. In SPHINCS-256, SEED_A = BLAKE-256(sk_1 || A) where BLAKE-256 is the cryptographic hash function [1] used as a PRF. When fed to a PRNG, this seed generates the secret key of the addressed instance.

3 Side-Channel Analysis of XMSS

3.1 Assumptions

To provide a sound analysis of side-channel resistance that is relatively independent of the actual implementation, we first assume that the used hash functions and PRNG suffer no side-channel leakage at all. Obviously, this assumption does not hold for any real world implementations, but it allows us to separate the analysis of the schemes (Sec. 3.2- 3.4) from the analysis of the hash function and PRNG (Sec. 3.5). We perform a bottom-up SCA, i.e., we start by analysing W-OTS$^+$ and then extend the analysis to XMSS and XMSS^{MT}. An extended version of this analysis is contained in [16].

3.2 W-OTS$^+$

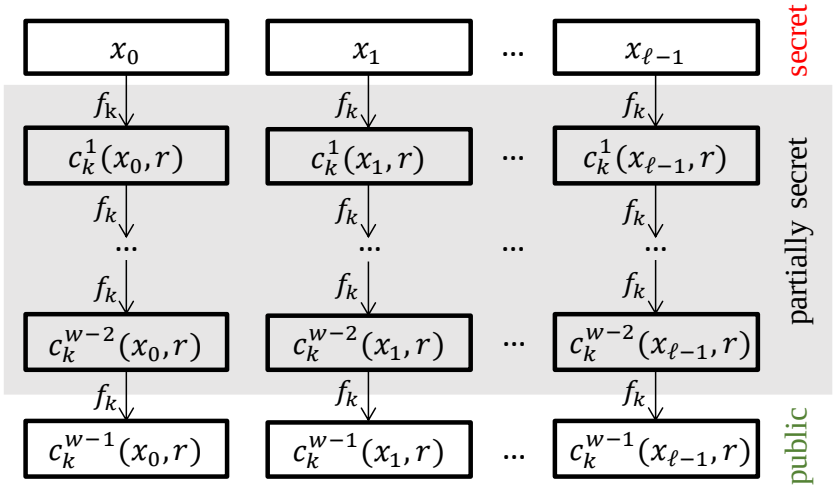

Fig. 1. Parts of W-OTS$^+$ relevant for SCA

As illustrated in Fig. 1, the only secret data processed inside W-OTS$^+$ are the secret key parts x_i. The used randomisation elements $\mathbf{r}$ and keys k are public values and, thus, are of no interest for an attacker. The x_i are only used as input to the chaining function c_k.

To mount a DPA attack, a function depending upon a part of the secret key and a known variable input data must be found. At first sight, $c_k^i(x, \mathbf{r})$ seems to be a perfect target for a power analysis attack. For $i = 1$, the signer calculates $x \oplus r_i$, where x is some secret key block and r_i is a randomisation bitmask known to the adversary. However, this function is only called twice: once during key generation and once during signature generation. This limited number of executions alone prevents the majority of side-channel attacks due to measurement noise. Additionally, r_i is the same for both evaluations. This prevents DPA attacks, which rely upon different inputs to the target function. Also, simple power analysis (SPA) attacks are unable to recover any relevant portion of the secret key.

3.3 XMSS

We just saw that W-OTS$^+$ barely leaks information via power side-channels. Since XMSS is built using many W-OTS$^+$ keys, intuitively XMSS provides this resistance as well. However, one major difference that makes XMSS more vulnerable than W-OTS$^+$ is that W-OTS$^+$ key generation is called much more often during authentication path computation.

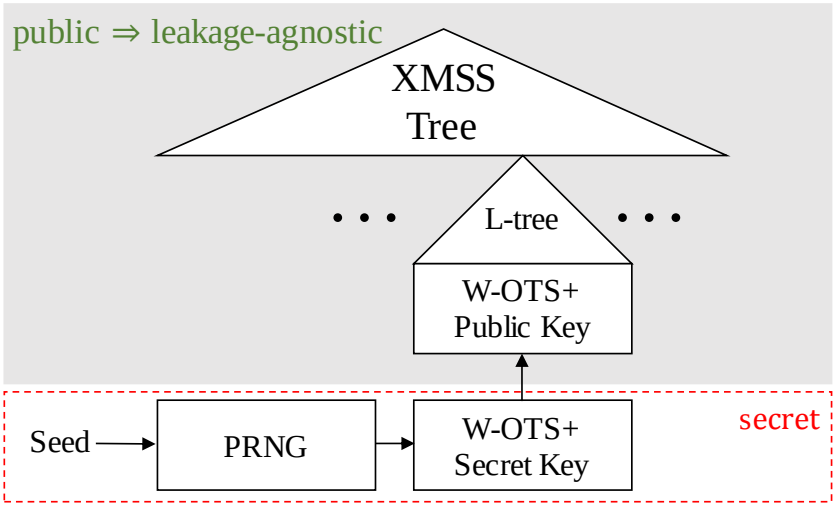

Fig. 2. Parts of XMSS relevant for SCA

Fig. 2 summarises the parts of XMSS relevant for SCA. The entire XMSS tree is public and, thus, leakage-agnostic. Even if leaked entirely, the adversary does not learn anything secret. This includes the W-OTS$^+$ public keys and the intermediate values in the L-trees, which are used to compute the XMSS tree leaves. The relevant parts are shown in the lower part of the figure and include

the seed used for the pseudorandom W-OTS$^+$ secret key generation and the W-OTS$^+$ secret keys itself.

It was found above that the power leakage resistance of W-OTS$^+$ can mainly be guaranteed because both the key generation and the signature generation are only executed once. For an XMSS signature using the W-OTS$^+$ secret key at index s, the signer first computes the W-OTS$^+$ signature, using $\mathrm{sk}_{OTS,s}$, and then the authentication path for $v_0[s]$. The authentication path calculation requires that all other $v_0[i]$ are computed too. While some nodes of the authentication path can be reused, some must be recomputed. Assuming the signer does not reuse nodes at all, we know that, at the time of the signature generation for index s, the $\mathrm{sk}_{OTS,s}$ already leaked a few times before.

Assuming the most powerful side-channel adversary [31] who can choose the leakage function arbitrarily and adaptively change it for each signature generation, this leads to a leakage of $2^{H+1} \cdot \lambda$ bits, where λ is a bound for the bits leaked per W-OTS$^+$ key generation and signature generation. Even a small bound λ trivially breaks the security of XMSS for any reasonable choice of H and n. However, in practice such an adversary does not exist. When considering a real-world leakage model, the attack becomes infeasible: During each signature generation the W-OTS$^+$ chaining function is called with the exact same inputs to produce the same W-OTS$^+$ public keys. While this is useful for filtering out noise, which is inevitable in every power analysis attack, the leaked information is still meagre.

When combining this finding with the assumption that the PRNG and hash function have no leakage and the findings above, we find that XMSS has the same leakage as W-OTS$^+$, but the adversary can use the multiple computations to reduce the noise.

3.4 Applicability to XMSS^{MT}

The conclusions we drew so far can be generalised to XMSS^{MT}, since an XMSS^{MT} signature generated using a hyper-tree with T layers can be viewed as T independent XMSS signatures from a side-channel perspective. One major difference is that the W-OTS$^+$ signature generations on the upper layers are executed more than once (if no caching is implemented). Intuitively, this seems to provide more leakage than the single tree variant of XMSS. However, since no relevant leakage could be identified during W-OTS$^+$ signature generation, XMSS^{MT} provides similar side-channel resistance.

3.5 Hash Function and PRNG Side-Channel Resistance

So far, we concluded that W-OTS$^+$, XMSS, and XMSS^{MT} provide strong side-channel resistance, under the assumption that the used hash function and the PRNG are side-channel resistant. Although the actual fulfilment of this requirement is implementation-specific, we now discuss the general side-channel resistance of the used building blocks.

Hash function The used hash function is the only function within W-OTS$^+$ which processes secret data. However, a hash function per se cannot be vulnerable or resistant to side-channel attacks, since it can be used in numerous ways which do not necessarily involve a secret key. In the XMSS Internet-Draft [14], the keyed hash function f_k of W-OTS$^+$ is implemented using either a hash function of the SHA-2 or SHA-3 function family using the construction $f_k(x) = f(0^n \mid\mid k \mid\mid x)$, where f is `SHA-256`, `SHA-512`, `SHAKE-128`, or `SHAKE-256`. However, the key k is generated from a public seed, while the actual secret data is x.

Several side-channel attacks, all of which being DPA attacks, have been proposed on both SHA-2 and SHA-3 hash function in the context of HMACs [2, 23, 32, 33]. However, they are not applicable to the W-OTS$^+$ chaining function, since each secret key part is only used with constant randomisation bitmasks (k and r). Additionally, SPA attacks are unable to recover any significant amount of secret key bits due to the absence of conditional branches depending upon the secret key for both SHA-2 and SHA-3. Note that the hash function may be replaced in future XMSS standards. It is thus imperative to analyse whether the replacement still provides similar side-channel resistance.

PRNG The PRNG which may be used for generating the W-OTS$^+$ secret keys is not specified by the XMSS Internet-Draft [14]. Thus, an implementer may freely choose a secure PRNG which matches the security parameter n.

For $n = 256$ and the SHA-2 hash function family, the XMSS Internet-Draft recommends the use of the following construction to generate a pseudorandom value for the index i from a seed: `SHA-256` $(0x000..03 \mid\mid seed \mid\mid i)$. For other parameters, similar recommended constructions are given.

This construction can be analysed with respect to side-channel attacks building upon the conclusions for the W-OTS$^+$ chaining function. The non-existence of conditional branches depending upon the input of the hash function implies that no SPA can be mounted upon any of the recommended PRNG. However, all constructions are good candidates for DPA attacks, since the hash functions are evaluated for the same seed with different indices.

3.6 DPA Attack on SHA-2 PRNG

To the best of our knowledge, only hash function side-channel resistance in the context of HMAC has been addressed in the literature. We adapt the attack on a SHA-2 HMAC [2] for the recommended XMSS PRNG. We briefly recall the attack by Belaïd et al.:

An HMAC for the message m can be computed by applying a hash function H twice [18]: $\texttt{HMAC}(m, k) = H\left((k \oplus opad) \mid\mid H\left((k \oplus ipad) \mid\mid m\right)\right)$. The bitmasks $opad$ and $ipad$ denote constants $0x5c5c \ldots 5c$ and $0x3636 \ldots 36$. When using a Merkle-Damgård-based hash function, the key is padded to the block length of the hash function (e.g., 512 bits for SHA-256), such that each evaluation of H results in at least 2 evaluations of the compression function cf.

The inner hash-evaluation of the HMAC is illustrated in Fig. 3. First, the compression function cf is called with the masked key $(k \oplus ipad)$ and the fixed

Algorithm 1 SHA-256 compression function cf [26]. Relevant operations for recovering $D^{(0)}$ are highlighted in blue.

Input: IV (256 bit), m_i (512 bit)
$W_t \leftarrow m_i^{(t)} \quad 0 \leq t \leq 15$
$W_t \leftarrow \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-16}) + W_{t-15} \quad 16 \leq t \leq 63$
$A \leftarrow IV^{(0)}; B \leftarrow IV^{(1)}; C \leftarrow IV^{(2)}; D \leftarrow IV^{(3)};$
$E \leftarrow IV^{(4)}; F \leftarrow IV^{(5)}; G \leftarrow IV^{(6)}; H \leftarrow IV^{(7)};$
for $t = 0; t < 64; t++$ **do**
    $T1 \leftarrow H + \Sigma_1(E) + Ch(E,F,G) + K_t + W_t$
    $T2 \leftarrow \Sigma_0(A) + Maj(A,B,C)$
    $H \leftarrow G; \quad G \leftarrow F; \quad F \leftarrow E; \quad E \leftarrow D + T1;$
    $D \leftarrow C; \quad C \leftarrow B; \quad B \leftarrow A; \quad A \leftarrow T1 + T2$
end for
return $[IV^{(0)}+A, \quad IV^{(1)}+B, \quad IV^{(2)}+C, \quad IV^{(3)}+D,$
$IV^{(4)}+E, \quad IV^{(5)}+F, \quad IV^{(6)}+G, \quad IV^{(7)}+H]$

initialisation vector (IV). Then, for each block in the message m, an additional call to cf iteratively combines the resulting IV from the previous iteration with 512 bits of the message m. Since the first evaluation only processes the key, but no variable data, it is not possible to mount a DPA attack on the computations inside. Instead, Belaïd et al. target the second evaluation of cf, which processes the first block of m and the result of the first evaluation of cf, denoted by IV_1. The computations inside cf can be used to entirely recover IV_1 which is enough to forge the inner part of the HMAC.

The actual attack is based upon intermediate values inside the SHA-2 compression function cf shown in Alg. 1. For definitions of Ch, Maj, Σ_0, Σ_1, σ_0, and σ_1, see [26]. Let $D^{(i)}$ denote the value of D before iteration $t = i$, thus $D^{(0)} = IV_1^{(3)}$. Similarly, $T1^{(i)}$ is the value of $T1$ that was computed during iteration $t = i - 1$. Additionally, let values that are different for each HMAC generation be denoted by bold letters (e.g., $\mathbf{W_t}$), while values that are the same for all generations are in standard letters (e.g., $T1$). To attack the HMAC, the adversary now mounts several DPA attacks building upon each other to recover

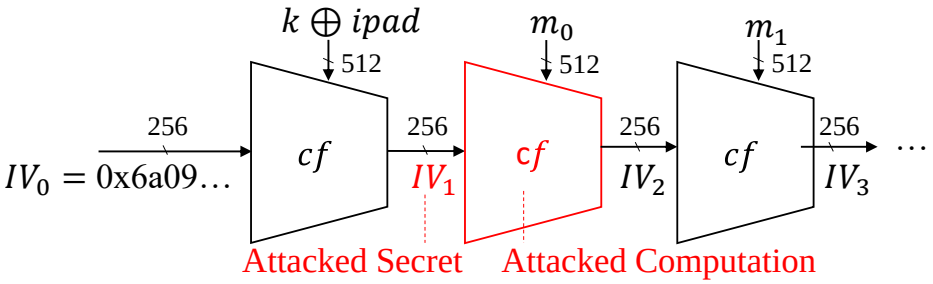

Fig. 3. DPA on SHA-256 HMAC (simplified from [2])

$A^{(0)}, \ldots, H^{(0)}$. These values are enough to forge the hash output of the inner hash function output for arbitrary message. The outer hash is attacked similarly.

As an illustration, we briefly describe how $D^{(0)}$ can be recovered and refer to [2] for the full attack. We define $\delta^{(1)} := H^{(0)} + \Sigma_1(E^{(0)}) + Ch(E^{(0)}, F^{(0)}, G^{(0)}) + K_0$, i.e., line 7 computes $\mathbf{T1^{(1)}} \leftarrow \delta^{(1)} + \mathbf{W_0}$. A DPA can easily recover $\delta^{(1)}$, since $\delta^{(1)}$ is fixed and $\mathbf{W_t}$ is known and variable. Once the adversary knows $\delta^{(1)}$, they can compute $\mathbf{T1^{(1)}}$ for each known word $\mathbf{W_0}$. The second DPA attack then recovers $D^{(0)}$ from $\mathbf{E^{(1)}} \leftarrow D^{(0)} + \mathbf{T1^{(1)}}$ using known values for $\mathbf{T1^{(1)}}$. Building upon the recovered values of $\mathbf{T1^{(1)}}$, another 7 DPAs in the first and second iteration can be used to recover the values of $A^{(0)}, B^{(0)}, C^{(0)}, E^{(0)}, F^{(0)}, G^{(0)}$ and $H^{(0)}$ (see [2]).

Application to hash-based PRNG: To the best of our knowledge, no power analysis attack on hash-based PRNG has been presented so far. However, the HMAC construction above resembles the PRNG suggested by the XMSS Internet-Draft [14] for the generation of W-OTS$^+$ secret keys. Trying to apply the attack of Belaïd et al. [2], we notice that the message words W_0 and W_1, which were used to mount the DPA attack, are always zero for any reasonable parameter choice ($h \leq 448$). If these known values are fixed, a DPA attack does not work.

The attack can be adapted to use $\mathbf{W_{14}}$ and $\mathbf{W_{15}}$ instead, assuming $i < 2^{64}$. The adversary is able to recover $A^{(14)}, \ldots, H^{(14)}$ from the computations in iteration 14 and 15. Although this does not allow to recover IV_1, it is still sufficient to recompute all pseudorandom secret keys, since $W_k = 0$ for $0 \leq k \leq 13$ which consequently means that $A^{(14)}, \ldots, H^{(14)}$ are the same for all values of i.

For our proof-of-concept, we assume that the PRNG is called for uniformly random values between 0 and $2^{64} - 1$. In XMSS it is called for subsequent values which are no bigger than 2^{20}. However, using these parameters, our attack is unable to recover all bits of $A^{(14)}, \ldots, H^{(14)}$. We leave the analysis on how much bits can be recovered for a certain parameter set to future work.

Implementation To validate that our attack indeed can be used to recover all W-OTS$^+$ secret keys, we created a proof-of-concept implementation of the attack. The source code of our implementation is available [17].

Power Simulation Since an actual hardware implementation of XMSS was not available, we implemented a power simulator which is capable of creating power traces in the Hamming weight (HW) leakage model. Since a DPA attack requires the computation of hypothetical power consumption values for each possible key hypothesis, our implementation recovers each byte of $A^{(14)}, \ldots, H^{(14)}$ separately. At first, we assume that we have a byte-wise leakage of the HW, which allows the recovery of the key with few traces. However, since this is not realistic, we also extend this to work with the leakage of the HW per 32-bit word using partial DPA.

Some of the DPA used to attack the PRNG target a 32-bit modular addition and some target bitwise `AND`. They require slightly different hypothesis calculation due to carry handling. For details we refer to our source code [17].

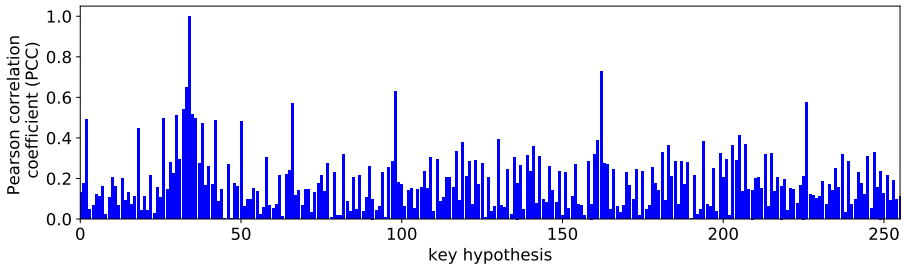

Fig. 4. Maximum PCC of all possible key hypotheses in the 8-bit HW leakage model. The correct sub-key (34) can be easily detected

Results We started evaluating our proposed attack in the 8-bit HW leakage model. Fig. 4 illustrates the maximum correlation values of each possible key hypothesis for computing the least significant byte of $\delta^{(15)}$, i.e., this is a DPA on a part of a 32-bit modular addition. We use the Pearson correlation coefficient (PCC) throughout the experiments for this paper. The correct hypothesis results in a correlation of 1.0, which is significantly higher than any other correlation, which allows the recovery of the least significant byte of $\delta^{(15)}$. Correlation values when using physically measured traces will be smaller than 1.0 due to noise. The detection of the correct sub-key will therefore be harder, and, in consequence, may require more traces. Fig. 4 also shows that correlation values are small (< 0.4) for most key candidates and only higher for 16 key candidates in this experiment. Thus, even if the noise is too high to successfully require the correct sub-key, it still allows a drastic reduction in the search space which can then be enumerated to find the correct key.

Next, we wanted to evaluate the success probability of the entire attack, which includes 9 DPA on 32-bit operation, i.e., 36 DPA when using the 8-bit HW leakage model. The success rates of the single DPA are not independent of each other due to two reasons: Firstly, when attacking addition, the higher significant bytes can only be recovered reliably if the lower significant byte key guesses are correct, since only then we can correctly calculate the carry bits. Secondly, the attacked operations depend on each other, e.g., the DPA attack on $D^{(14)}$ requires that the DPA on $\delta^{(15)}$ was successful. Thus, the success rate of the entire attack is certainly smaller than for each individual DPA.

Fig. 5 shows the success rate of the full key recovery attack using different numbers of traces, where one trace corresponds to an execution of the PRNG with an uniformly random index between 0 and $2^{64} - 1$. When using only $T = 8$ traces per experiment, the recovery failed for 100% of our trials, whereas using $T = 10$ already resulted in a success probability of almost 60%. This further increased to 93.3% for $T = 512$. However, we noticed that the DPA on `AND` operations always failed to recover the key if the sub-key is equal to zero. This is because for a zero key value the calculated value always has a HW of zero. Since this is a constant value, no correlation can be found with DPA. However, this can be detected by the adversary, and they can conjecture that the key must be zero. We did not

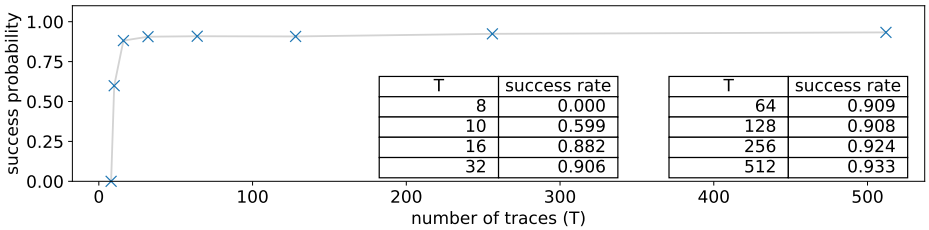

Fig. 5. Success rate of the full DPA key recovery attack on the vulnerable PRNG in the 8-bit HW leakage model

implement this optimisation in our proof-of-concept implementation, thus, since the attacked values are uniformly distributed and we have a total of 16 DPA on `AND`, the probability of having all key bytes $\neq 0$ is $\left(\frac{255}{256}\right)^{16} \approx 0.939$. Therefore, we conjecture that the best achievable success rate in this set-up is about 93.9%, no matter how many traces are used. The actual numbers from these experiments can only provide a lower bound, since our traces contain no noise. In real-world measurements, the required number of traces is larger depending on noise.

Partial DPA So far, we assumed that the implementation leaks the HW of each byte separately, such that we can mount independent DPA upon them. However, since SHA-2 only involves 32-bit arithmetic, a byte-wise implementation is unrealistic. Most implementations will use 32-bit words and, thus, only leak the HW of the entire words.

Luckily, the strategy can be adapted and still be used to recover each byte separately using partial DPA [2], although requiring a much higher number of traces. We integrated this in our proof-of-concept implementation and were able to reproduce the results of [2].

3.7 Impact

Our proof-of-concept implementation shows that if the SHA-2-based PRNG (`SHA-256` $(0x000..03 \mid\mid seed \mid\mid i)$) is called for indices i which vary in 64 bits, an adversary is able to recover an intermediate value which allows one to calculate the output of the PRNG for arbitrary indices. Applied to XMSS and XMSS^{MT}, this allows an adversary to recover all W-OTS$^+$ secret keys which trivially compromises the security of the scheme.

However, in both schemes, the PRNG is never called for indices larger than 2^{20}, which prevents the presented attack. If a different PRNG is used, it may be vulnerable to our attack with current XMSS parameters. For example, if the PRNG is modified to `SHA-256` $(0x000..03 \mid\mid seed \mid\mid$ `SHA-256`$(i))$, the inner hash evaluation results in uniformly random inputs to the outer hash evaluation which makes our attack practical. This emphasises that if a different PRNG is used, not only the black-box security needs to be considered, but also its side-channel resistance.

3.8 Recommendations

Since XMSS is currently being standardised, practical implementations which need to be protected against power analysis attacks are likely to be created soon. Our results suggest that the most critical part to protect is the PRNG. While the proposed XMSS standard leaves open the actual choice of the PRNG, we showed that the PRNG selection is critical for side-channel resistance. It is crucial to use a PRNG which is well-studied with respect to side-channel attacks, like the one recommended by the XMSS Internet-Draft.

We also found that optimised authentication path computation (e.g., using the BDS algorithm [5]) greatly decreases the side-channel leakage of XMSS because it minimises the accesses to the secret keys and, consequently, executions of the PRNG. Although this optimisation is deemed optional by the XMSS Internet-Draft, every implementation should use it.

Timing attacks were not discussed in this paper, but protecting implementations against them is also necessary. Since constant time implementations exist for all used hash functions, PRNGs, and PRFs, protecting XMSS and SPHINCS against such attacks is straightforward.

4 SPHINCS-256: A DPA on BLAKE

In the previous section, we analysed the side-channel resistance of XMSS and XMSS^{MT} and presented a DPA attack on a SHA-256-based PRNG which is used within both XMSS and XMSS^{MT}. To extend the analysis, we evaluate SPHINCS-256 in the same regard.

SPHINCS relies on XMSS^{MT}, HORST, and a stateless way of addressing hash-based instances within the scheme. Since the HORST hash tree construction does not leak anything about its secret key, we can assume this component to be side-channel resistant. Moreover, XMSS^{MT} can also be assumed secure by the previous analysis. This leaves us only with the stateless way of computing the PRNG seeds, which we now analyse. This analysis was initially studied in [12].

SPHINCS-256 PRF In SPHINCS-256, the W-OTS$^+$ and HORST secret seeds are generated with BLAKE-256($\mathrm{sk}_1 \mid\mid A$) where $\mathrm{sk}_1 \in \{0,1\}^{256}$ is the SPHINCS secret key, $A \in \{0,1\}^{64}$ the address of the instance, and BLAKE-256 the hash function [1]. Recovering sk_1 would therefore result in a total security break. We now present a 6-DPA attack on the BLAKE hash function in the context of SPHINCS-256 that recovers one 32-bit chunk of the secret key sk_1.

4.1 DPA

The BLAKE-256 compression procedure takes 12 similar rounds during which the input is mixed. Similarly as in Sec. 3, the goal is to subsequently recover intermediate values at certain points in the procedure, to eventually recover one secret chunk. As these values are mixed with variable values early in the procedure, the DPAs focus on the first two rounds. Within SPHINCS-256, the

first round is summarised in Alg. 2. Here, the values v_i for $0 \leq i < 15$ are initialised with known constant values. A general mixing subroutine `Mix` involved in these steps is shown in Alg. 3. Here, $M_i \in \{0,1\}^{32}$ for $0 \leq i < 15$ is a chunk of the input padded with a constant and known padding. The function $\sigma_z(i)$ is a permutation that depends on the round $0 \leq z < 12$. Again, the values of C_i for $0 \leq i < 15$ are given constants.

Algorithm 2 Round $z = 0$ of BLAKE-256 compression algorithm [1].

Input: $(s_0, \ldots, s_7)$ — secret key sk_1 split into 8 chunks of 32 bits each
Input: (a_0, a_1) — address A split into two chunks of 32 bits each

1: $\texttt{Mix}(v_0, v_4, v_8, v_{12};\ s_0, s_1)$
2: $\texttt{Mix}(v_1, v_5, v_9, v_{13};\ s_2, s_3)$
3: $\texttt{Mix}(v_2, v_6, v_{10}, v_{14};\ s_4, s_5)$
4: $\texttt{Mix}(v_3, v_7, v_{11}, v_{15};\ s_6, s_7)$
5: $\texttt{Mix}(v_0, v_5, v_{10}, v_{15};\ a_0, a_1)$
6: $\texttt{Mix}(v_1, v_6, v_{11}, v_{12};\ \texttt{0x80000000}, \texttt{0x00000000})$
7: $\texttt{Mix}(v_2, v_7, v_8, v_{13};\ \texttt{0x00000000}, \texttt{0x00000001})$
8: $\texttt{Mix}(v_3, v_4, v_9, v_{14};\ \texttt{0x00000000}, \texttt{0x00000140})$

Algorithm 3 `Mix` procedure involved in Alg. 2.

Input: $(v_a,\ v_b,\ v_c,\ v_d)$ — intermediate values of 32 bits each
Input: $(M_{\sigma_z(e)},\ M_{\sigma_z(e+1)})$ — hash function input chunks of 32 bits each

1: $v_a \leftarrow (v_a + v_b) + (M_{\sigma_z(e)} \oplus C_{\sigma_z(e+1)})$
2: $v_d \leftarrow (v_d \oplus v_a) \lll 16$
3: $v_c \leftarrow v_c + v_d$
4: $v_b \leftarrow (v_b \oplus v_c) \lll 12$
5: $v_a \leftarrow (v_a + v_b) + (M_{\sigma_z(e+1)} \oplus C_{\sigma_z(e)})$
6: $v_d \leftarrow (v_d \oplus v_a) \lll 8$
7: $v_c \leftarrow v_c + v_d$
8: $v_b \leftarrow (v_b \oplus v_c) \lll 7$

In Alg. 2, line 5 involves v_0, v_5, v_{10}, and v_{15}, which all respectively depend on two constant chunks of sk_1, and the address. When the `Mix` procedure is unrolled, the operation $v_0 \leftarrow (v_0 + v_5) + (a_0 \oplus C_9)$ at line 1 involves $(v_0 + v_5)$, and $(a_0 \oplus C_9)$: the first half of the address A masked with a constant. By targeting this addition, we can recover $(v_0 + v_5)$ with a first DPA. Once recovered, the following values for v_5, v_{10}, and v_{15} can be consecutively recovered with additional DPAs. Since the rest of the `Mix` procedure does not involve any other unknown value, and since these values are not mixed again during round 0, they are, therefore, all known at the beginning of round 1.
On round 1 of the BLAKE-256 compression algorithm, $\texttt{Mix}(v_1, v_5, v_9, v_{13};\ s_4, s_5)$ is called. Line 1 in Alg. 3 for this call involves v_5 which has been recovered from before, and v_1 which can be recovered with a fifth DPA. Finally, a sixth DPA on $(v_1 + v_5) + (s_4 \oplus C_5)$ can recover s_4, which consists of one chunk of 32 bits of the secret key sk_1.

Setup and implementation The SCA was performed on an Arduino Due microcontroller, based on the Atmel SAM3X8E Cortex-M3 CPU. Power consumption was collected by placing a local near-field probe on the chip at the position shown

in Fig. 6. The attack considers the BLAKE-256 reference implementation [1] with an additional assumption: the addition of $(v_a + v_b)$ at lines 1 and 4 in Alg. 3 is performed before the rest. This makes the recovery of v_a or v_b alone harder, but should not affect our results. We provide the code that was used for evaluating the attack at [17].

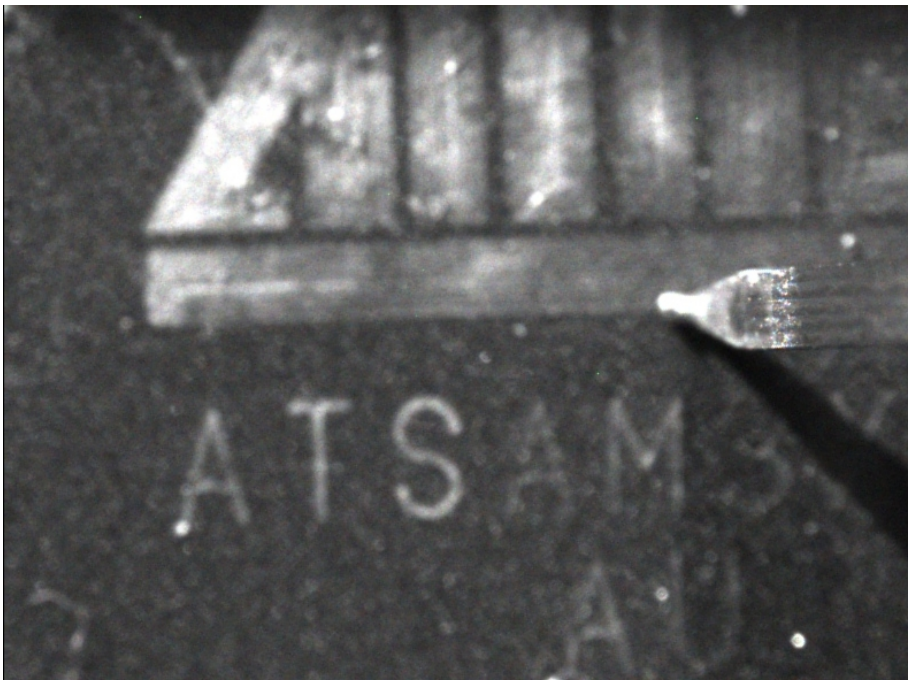

Fig. 6. Position of an EM probe on a SAM3X8E Cortex-M3 microcontroller at which strong EM radiations could be collected

4.2 Real-Device Analysis and Results

To confirm the practicality of the attack, we performed the first two DPAs of our attack on real traces. We collected $t = 10000$ different traces of the two targeted operations, where the secret key sk_1 was fixed and the addresses A were drawn uniformly at random. This number can be obtained by signing around 2000 different messages, as BLAKE-256 is called on 7 different layers with a different a_0 for a single signature. We use the HW leakage model.

We evaluated the relation between the power traces and the guesses on, first, $(v_0 + v_5)$, and, then, on v_{15}, using Pearson's correlation with a partial DPA on 16 bits, as explained in Sec. 3. The leakages of both the addition and the XOR operation are shown in Fig. 7. The upper plots show the main correlation peaks

of 1000 guesses on the most significant bits of the targeted value, while the lower plots show the power consumption average.

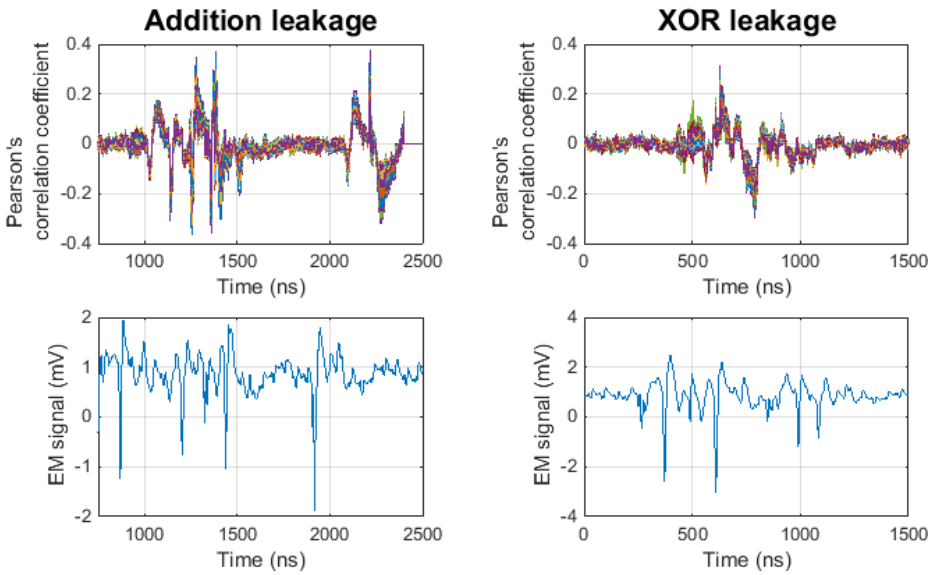

Fig. 7. Power traces average and main PCC peaks on the frist half of the targeted values for the addition and XOR operations ($t = 10000$)

By computing the maximum PCC of the 2^{16} possible values for the most significant bits of $v_0 + v_5$ (in the case of the addition) and v_{15} (in the case of the XOR operation) we obtain Figure 8. In both cases, the candidate with the bigger correlation factor in *absolute value* always happens to be the right value. Similar results were found with the least significant bits, which confirms that the overall attack can be successfully mounted, as the other DPAs target the same kind of operations.

4.3 Impact

The described attack recovers s_4, the fifth 32-bit chunk of sk_1. This makes the stateless construction of SPHINCS-256 vulnerable to DPA. Recovering this chunk potentially leads to the recovery of other chunks, but additional investigation is required.

4.4 Countermeasures

In order to mitigate the effect of this attack, we suggest *hiding* the order of the `Mix` procedures. During a BLAKE-256 round, the first four calls — as well as the

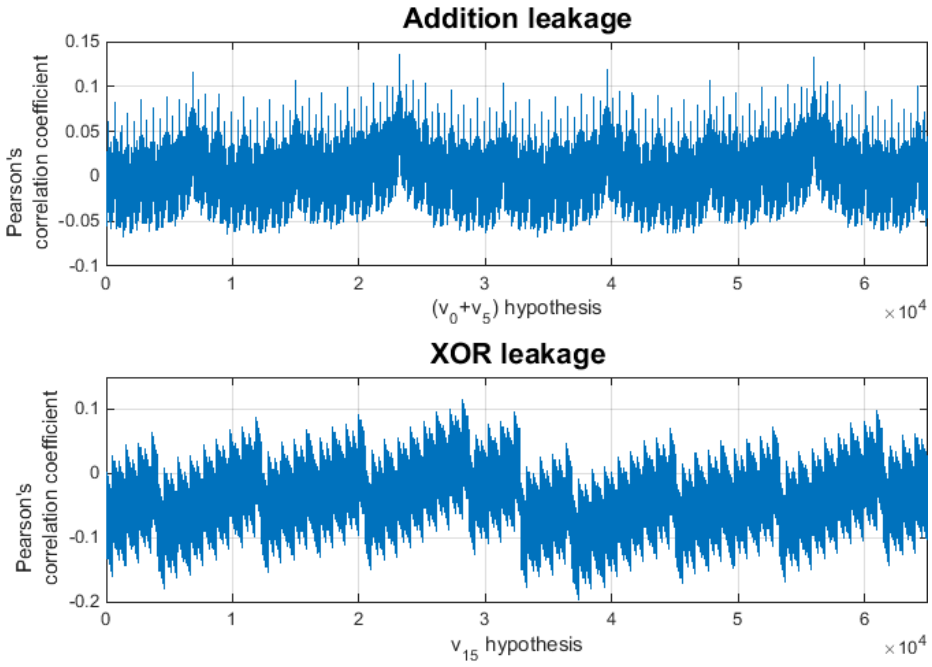

Fig. 8. Maximum PCC of all possible hypotheses on the first half of the targeted values. On the upper plot (addition), the most positively correlated value corresponds to the correct half of $(v_0 + v_5)$, while the lower plot (XOR), the most negatively correlated value corresponds to the correct half of v_{15}

next four — do not depend on each other. Their order can thus be rearranged randomly. This forces an attacker to synchronise the collected traces, making the DPAs more complex.

5 Conclusion

In this paper, we analysed the side-channel resistance of two modern HBS schemes, XMSS and SPHINCS, with a focus on DPA resistance. We presented a novel DPA vulnerability of a SHA-2-based PRNG for XMSS, as well as an attack on the BLAKE-256-based PRF used within SPHINCS-256. While the first attack is not threatening current versions of XMSS, the second one is practical for the actual parameters of SPHINCS-256.

Besides these two found vulnerabilities, we performed a thorough analysis of the building blocks of both XMSS and SPHINCS. Our results confirm the conjecture that XMSS provides strong protection against differential power analysis attacks. This further increases the confidence in the security of stateful HBS schemes, which contributes to a rigorous standardisation process.

Acknowledgments

We would like to thank Hervé Pelletier and Roman Korkikian from Kudelski Group for their help and expertise in the practical verification of the DPA on BLAKE-256. This work has been co-funded by the German Research Foundation (DFG) as part of project BU 630/28-1, and as part of projects P1 and S6 within the CRC 1119 CROSSING.

References

1. Aumasson, J., Meier, W., Phan, R.C., Henzen, L.: The Hash Function BLAKE. Information Security and Cryptography, Springer (2014)
2. Belaïd, S., Bettale, L., Dottax, E., Genelle, L., Rondepierre, F.: Differential Power Analysis of HMAC SHA-2 in the Hamming Weight Model. In: SECRYPT 2013. pp. 230–241. SciTePress (2013)
3. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: SPHINCS: Practical Stateless Hash-Based Signatures. In: EUROCRYPT 2015. LNCS, vol. 9056, pp. 368–397. Springer (2015)
4. Buchmann, J., Dahmen, E., Hülsing, A.: XMSS — A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In: PQCrypto 2011. LNCS, vol. 7071, pp. 117–129. Springer (2011)
5. Buchmann, J., Dahmen, E., Schneider, M.: Merkle Tree Traversal Revisited. In: PQCrypto 2008. LNCS, vol. 5299, pp. 63–78. Springer (2008)
6. Buchmann, J., García, L.C.C., Dahmen, E., Döring, M., Klintsevich, E.: CMSS — An Improved Merkle Signature Scheme. In: INDOCRYPT 2006. LNCS, vol. 4329, pp. 349–363. Springer (2006)
7. Buchmann, J.A., Lauter, K.E., Mosca, M.: Postquantum Cryptography — State of the Art. IEEE Security & Privacy 15(4), 12–13 (2017)
8. Butin, D.: Hash-Based Signatures: State of Play. IEEE Security & Privacy 15(4), 37–43 (2017)
9. Castelnovi, L., Martinelli, A., Prest, T.: Grafting Trees: a Fault Attack against the SPHINCS framework. Cryptology ePrint Archive, Report 2018/102 (2018), `https://eprint.iacr.org/2018/102`
10. Dods, C., Smart, N.P., Stam, M.: Hash Based Digital Signature Schemes. In: Cryptography and Coding, 10th IMA International Conference. LNCS, vol. 3796, pp. 96–115. Springer (2005)
11. Eisenbarth, T., von Maurich, I., Ye, X.: Faster Hash-Based Signatures with Bounded Leakage. In: SAC 2013. LNCS, vol. 8282, pp. 223–243. Springer (2014)
12. Genêt, A.: Hardware Attacks against Hash-based Cryptographic Algorithms. Tech. rep., École polytechnique fédérale de Lausanne (2017), Master Thesis
13. Hülsing, A.: W-OTS$^+$ — Shorter Signatures for Hash-Based Signature Schemes. In: AFRICACRYPT 2013. LNCS, vol. 7918, pp. 173–188. Springer (2013)
14. Hülsing, A., Butin, D., Gazdag, S., Rijneveld, J., Mohaisen, A.: Internet-Draft: XMSS: Extended Hash-Based Signatures. `https://datatracker.ietf.org/doc/draft-irtf-cfrg-xmss-hash-based-signatures/` (2018)
15. Hülsing, A., Rausch, L., Buchmann, J.: Optimal Parameters for XMSS^{MT}. In: MoCrySEn. LNCS, vol. 8128, pp. 194–208. Springer (2013)

16. Kannwischer, M.J.: Physical Attack Vulnerability of Hash-Based Signature Schemes. Tech. rep., Technische Universität Darmstadt (2017), Master Thesis, https://www.cdc.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_CDC/Documents/theses/Matthias_Kannwischer.master.pdf
17. Kannwischer, M.J., Genêt, A., Butin, D., Krämer, J., Buchmann, J.: GitHub repositories for DPA code of SHA-256 PRNG and BLAKE-256 PRF, https://github.com/hbs-sca
18. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (1997), http://www.ietf.org/rfc/rfc2104.txt
19. Lamport, L.: Constructing Digital Signatures from a One Way Function. Tech. rep., SRI International CSL (1979), https://www.microsoft.com/en-us/research/publication/constructing-digital-signatures-one-way-function/
20. Lee, M., Song, J.E., Choi, D., Han, D.: Countermeasures against Power Analysis Attacks for the NTRU Public Key Cryptosystem. IEICE Transactions 93-A(1), 153–163 (2010)
21. Maurand, R., Jehl, X., Kotekar-Patil, D., Corna, A., Bohuslavskyi, H., Laviéville, R., Hutin, L., Barraud, S., Vinet, M., Sanquer, M., De Franceschi, S.: A CMOS silicon spin qubit. Nature Communications 7, 13575 (2016)
22. von Maurich, I., Güneysu, T.: Towards Side-Channel Resistant Implementations of QC-MDPC McEliece Encryption on Constrained Devices. In: PQCrypto 2014. LNCS, vol. 8772, pp. 266–282. Springer (2014)
23. McEvoy, R.P., Tunstall, M., Murphy, C.C., Marnane, W.P.: Differential Power Analysis of HMAC Based on SHA-2, and Countermeasures. In: WISA 2007. LNCS, vol. 4867, pp. 317–332. Springer (2007)
24. McGrew, D.A., Kampanakis, P., Fluhrer, S.R., Gazdag, S., Butin, D., Buchmann, J.A.: State Management for Hash-Based Signatures. In: SSR 2016. LNCS, vol. 10074, pp. 244–260. Springer (2016)
25. Merkle, R.C.: A Certified Digital Signature. In: CRYPTO '89. LNCS, vol. 435, pp. 218–238. Springer (1989)
26. National Institute of Standards and Technology: FIPS PUB 180-4: Secure Hash Standard (2015), http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
27. NIST Computer Security Division: Post-Quantum Cryptography Standardization — Call for Proposals Announcement. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization (2017)
28. PQCRYPTO Project: Initial recommendations of long-term secure post-quantum systems. https://pqcrypto.eu.org/docs/initial-recommendations.pdf (2015)
29. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. on Computing 26(5), 1484–1509 (1997)
30. Silverman, J.H., Whyte, W.: Timing Attacks on NTRUEncrypt Via Variation in the Number of Hash Calls. In: CT-RSA 2007. LNCS, vol. 4377, pp. 208–224. Springer (2007)
31. Standaert, F., Pereira, O., Yu, Y., Quisquater, J., Yung, M., Oswald, E.: Leakage Resilient Cryptography in Practice. In: Towards Hardware-Intrinsic Security — Foundations and Practice, pp. 99–134. Information Security and Cryptography, Springer (2010)
32. Taha, M.M.I., Schaumont, P.: Differential Power Analysis of MAC-Keccak at Any Key-Length. In: IWSEC 2013. LNCS, vol. 8231, pp. 68–82. Springer (2013)
33. Zohner, M., Kasper, M., Stöttinger, M., Huss, S.A.: Side Channel Analysis of the SHA-3 Finalists. In: DATE 2012. pp. 1012–1017. IEEE (2012)