

Soft Analytical Side-Channel Attacks on SHA-2 and HMAC

Maxime Lecomte¹, Julien Maillard¹, Antoine Moran^{1,2}, Guénaél Renault^{2,3}
and Benjamin Smith²

¹ Univ. Grenoble Alpes, CEA, Leti, F-38000 Grenoble, France.

² LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France.

³ ANSSI, Paris, France.

Abstract. We investigate the use of bit-level Soft Analytical Side-Channel Attacks (SASCA) to recover secret inputs of the SHA-256 hash function, and secret keys of HMAC-SHA-256, using Sentential Decision Diagrams to overcome the computational challenge posed to classical Belief Propagation by multiple-word modular addition. Our attack on HMAC requires no control nor knowledge of the input, and exploits its double manipulation of the key to improve key-recovery performance. We make comprehensive simulations to evaluate the attacker’s recovery capacity for both SHA-256 and HMAC-SHA-256. Finally, we study the profiling capabilities of load instructions in a multiposition EM probe setup on a STM32F303RET6 microcontroller, and use these results to evaluate the capability of our attack against software implementations of HMAC-SHA-256 in a realistic scenario.

Keywords: Side-Channel Attack · SASCA · SHA-256 · HMAC · Sentential Decision Diagrams

1 Introduction

While cryptographic hash functions are public and have no keys as such, they are often given secret keys to process—for example, in the well-known HMAC construction [Nat08]. They are therefore interesting targets for side-channel analysis (SCA), where attackers try to recover the secret by observing side-channel leakage—such as power consumption or electromagnetic (EM) radiation—during the hashing process.

The *Secure Hash Algorithm* (SHA-2) family of hash functions comprising SHA-224, SHA-256, SHA-384, and SHA-512 was standardized by the National Institute of Standards and Technology (NIST) in 2002 [Nat02]. While it was superseded by the newer SHA-3 standard in 2015 [Nat15], SHA-2 is still commonly used in internet security protocols including TLS, IPsec, and SNMP, blockchain technologies including Bitcoin, and especially in *Message Authentication Codes* (MACs) such as HMAC-SHA-256. Indeed, at the time of writing, SHA-256 based HMAC is still enabled by default in SSH, TLS, and IPsec. SHA-256 and its HMAC counterpart are also widely used as PRFs in higher-level primitives like HKDF [KE10, BCD20] and XMSS [HBG⁺18, CAD⁺20]. This is not a security problem in itself: while its design is similar to the now-broken hash SHA-1, no algebraic or algorithmic attack is known to reduce the claimed security of SHA-2.

So far, the side-channel security of SHA-2 implementations has been studied through *Differential Power Analysis* (DPA) [MTMM07, BBD⁺13, KGB⁺18, BDT⁺23] and *Template Attacks* (TAs) [BDT⁺21]. *Soft Analytical Side-Channel Attacks* (SASCAs) [VGS14] an extension of TAs, can outperform Algebraic SCAs even in noise-free contexts, with many fewer training traces than profiled DPA attacks [GS15]. SASCAs have been shown to be

effective against SHA-3 [KPP20, YK21, MHL24], but have not yet been tried against SHA-2 or HMAC instances. This motivates our investigation of the resistance of SHA-2 and HMAC-SHA-256 against SASCA.

1.1 Contributions

In this paper, we implement SASCAs based on Belief Propagation (BP) to recover hash inputs by exploiting the leakage of inter-round states, and evaluate the robustness of SHA-256 and HMAC-SHA-256 against SASCA with a bitwise model using simulations. Our main contributions are:

- Using Sentential Decision Diagrams (SDD), we propose the first SASCA against SHA-256. By eliminating the need for variability, we tackle existing limitations of previous DPAs against SHA-256 used to attack XMSS signatures.
- We show how the structure of HMAC increases its vulnerability to SASCA, with a relaxed attacker model compared to the state-of-the-art.
- We evaluate the classifier accuracy required for successful attacks, and study the impact of the number of profiled rounds of the hash function on attack success.
- Finally, we perform multiposition profiling of load operations on an STM32F303 board, and show that this method allows key recovery for unmasked software implementations of HMAC-SHA-2.¹ We are able to recover a HMAC-SHA-256 key in 21% of the cases with as few as 1250 traces, while 2500 traces suffice to recover a SHA-256 input block in 24% of the cases.

All of our attacks can be performed in a few minutes with a parallel implementation of the BP algorithm on a regular desktop computer with less than 3GB of memory.

1.2 Related work

McEvoy, Tunstall, Murphy, and Marnane [MTMM07] mounted the first DPA on HMAC-SHA-2, with a Hamming distance leakage model on an FPGA implementation, and described masking countermeasures on SHA-2 internal routines to thwart their attack. Their attack was improved by Belaïd, Bettale, Dottax, Genelle, and Rondepierre [BBD⁺13] with an implementation-independent DPA in the Hamming Weight model (and some countermeasures). More recently, [BDT⁺21] and [BDT⁺23] proposed template and DPA attacks, respectively, on an HMAC-SHA-2 hardware implementation. In [KGB⁺18], the authors used the same DPA procedure against XMSS when SHA-256 is used as a PRNG within W-OTS⁺. However, the straight-forward PRNG construction with SHA-256 recommended by XMSS Internet Draft [HBGM16] does not provide enough variability to mount a DPA on all bits of the secret. Thus, the authors acknowledged that their attack could work if a slightly modified PRNG is used. All of these attacks assume that the messages sent to the device are variable and known to the attacker. These attacks do not recover the secret key, but rather the chaining value derived from the secret, which is enough to forge counterfeit HMAC signatures.

SASCA was first used by Veyrat-Charvillon, Gérard, and Standaert [VGS14] to attack a Furious implementation of AES. Later, it was adapted to key-recovery attacks on Kyber targeting the number theoretic transform [PPM17, PP19, HHP⁺21, HSST23]. More recently, You, Kuhn, Sarkar, and Hao [YKSH23] used SASCA to recover the secret key of ASCON-128 in a few traces, even against a first-order masked implementation.

¹As we note in Section 7, masking may be an effective countermeasure to our attacks; however, unmasked implementations of SHA-2 are predominant in real-world deployments.

Kannwischer, Pessl, and Primas mounted the first single trace SASCA on Keccak [KPP20], with simulations in a Hamming-weight leakage scenario on 8-, 16-, and 32-bit architectures. Their approach represented variables with clustered nodes (either 8- or 16-bit clusters); while this was compatible with most of Keccak’s inner routines, they had to revert to a bit-level representation around the χ function.

Later, You and Kuhn [YK21] performed the first SASCA on a software implementation of Keccak-f[1600]. Unlike [KPP20], they opted for a bit-level representation of the whole Keccak function, obtaining bit-level probabilities with a *fragment template attack*, which succeeded in a single trace despite the information loss inherent to bit-level marginalization. Building on this, Maillard, Hiscock, Lecomte, and Clavier [MHL24] studied the capabilities of SASCA against different variants of Keccak. They also showed that multiple calls to Keccak-based SHAKE-256, used as a pseudo-random function in Kyber and Dilithium, can improve an attacker’s ability to recover the secret message and key.

More recently, Wedenig, Nagpal, Cassiers, Mangard, and Peharz introduced *Exact Soft Analytical Side-Channel Attacks* [WNC⁺25], using SDDs to improve the performance and precision of SASCA for the first time. They used a PSDD to build a factor for a reduced version of AES’s MIXCOLUMN, then used its linearity to compute the marginals for the whole MIXCOLUMN operation, thus reducing the runtime of message inference by three orders of magnitude (compared to exact inference).

1.3 Outline

We recall necessary background in Section 2, and present the attack model in Section 3. In Section 4 we describe the construction of factor graphs for SHA-256, and discuss the advantages of SDD-based factors. We mount attacks exploiting single invocations of the hash function, and on HMAC-SHA-256, in Section 5. In Section 6, we profile the electromagnetic emanations during the loading of a 32-bit word on a STM32F303RET6 board: by acquiring multiple traces with the probe placed at various positions over the chip, we practically assess the effectiveness of our attacks using a multi-position approach. We discuss possible countermeasures in Section 7, and perspectives on further research in Section 8.

2 Background

In this section, we recall some general notions about SASCA, SHA-2, and HMAC.

2.1 SASCA and Belief Propagation

SASCAs are an extension of *template attacks* [CRR02] in which adversaries exploit arithmetic relationships between the profiled variables to efficiently compute a *Maximum A Posteriori* (MAP) estimate of the distribution of a secret. These estimations can be made by exploiting the locality of the target function, represented within a *factor graph*.

A factor graph is a bipartite graph containing two types of nodes:

1. *Variable nodes* store the probability distributions of their corresponding intermediate variables.
2. *Factor nodes* store constraint forms representing arithmetic links between the (variable) nodes they are connected to.

For example: a factor node representing the XOR of three variables a , b , and c holds the constraint function

$$f_{\text{XOR}}(a = x_a, b = x_b, c = x_c) = \begin{cases} 1 & \text{if } x_a = x_b \oplus x_c, \\ 0 & \text{otherwise.} \end{cases}$$

Marginals estimation is carried out by a message-passing algorithm on the factor graph called *belief propagation* (BP), in which each variable node computes a localized marginal distribution estimation with the information gathered from its neighbors. This information transits in the graph via a message-passing algorithm [KFL01], in which variable nodes send messages to their surrounding factor nodes, and vice versa, until convergence is reached.

A message from a factor node f to a variable node v is denoted $\mu_{f \rightarrow v}$, while a message from a variable node v to a factor node f is denoted $\mu_{v \rightarrow f}$. Each message represents a probability distribution: we will for instance write $\mu_{v \rightarrow f}(v = x_v)$ to express the probability that v equals x_v in the message $\mu_{v \rightarrow f}$.

For each node m , we let $n(m)$ denote the set of its neighbors. In particular, if f is a factor node, then $n(f)$ is the set of arguments of f . (For convenience, we use the same symbol for a factor node and its corresponding local function.) For a particular value x_v , a variable node v will send the following message to a function node f :

$$\mu_{v \rightarrow f}(v = x_v) = \prod_{h \in n(v) \setminus \{f\}} \mu_{h \rightarrow v}(v = x_v), \quad (1)$$

that is, the product of all the messages coming from nodes other than f . A function node f will send the following message to a variable node v :

$$\mu_{f \rightarrow v}(v = x_v) = \sum_{\mathbf{w} \in \sim\{v\}} \left(f(v = x_v, n(f) \setminus v = \mathbf{w}) \prod_{y \in n(f) \setminus \{v\}} \mu_{y \rightarrow f}(y = w_y) \right). \quad (2)$$

Here $\sum_{\mathbf{w} \in \sim\{v\}}$ denotes the “not-sum” of [KFL01] (that is, the sum over $\mathbf{w} = (w_{v_1}, \dots, w_{v_n})$ in the Cartesian product of the domains of all the variables $(v_1, \dots, v_n)$ in $n(f)$ *except* v). This means that $\mu_{f \rightarrow v}$ is the sum over the variables coming to f other than v , of the product of the messages from these variables to f , for all the instantiations that comply with the constraint imposed by f . After the message-passing algorithm terminates, the marginal distribution of a variable is obtained by multiplying all the messages arriving at its corresponding variable node.

The BP algorithm has been proven to compute exact marginals on acyclic graphs. However, factor graphs used for cryptographic applications often contain cycles, in which case *loopy-BP* can only provide an estimation of the exact marginals. The quality of this estimation is close to exact inference mainly in high- and low-SNR scenarios, with suboptimal results in between [NCK+24].

2.2 Sentential Decision Diagrams

Sentential Decision Diagrams (SDDs) are a category of tractable circuits used for knowledge compilation: they represent boolean functions as circuits which can answer certain queries in polynomial time [Dar11]. We recall some useful definitions below.

(X, Y)-decompositions. Let $f(\mathbf{X}, \mathbf{Y})$ be a boolean function, where $\mathbf{X}$ and $\mathbf{Y}$ are disjoint sets of boolean variables. If we can write f as $((p_1(\mathbf{X}) \wedge s_1(\mathbf{Y})) \vee \dots \vee (p_n(\mathbf{X}) \wedge s_n(\mathbf{Y})))$, then we call $\{(p_1, s_1), \dots, (p_n, s_n)\}$ an **(X, Y)-decomposition** of f . The tuples (p_i, s_i) are called *elements* of the decomposition. The decomposition is *strongly deterministic* if $p_i \wedge p_j = \text{False}$ for $i \neq j$; in that case the p_i are called *primes*, while s_i are called *subs*.

X-partitions. A strongly deterministic **(X, Y)-decomposition** $\alpha = \{(p_1, s_1), \dots, (p_n, s_n)\}$ of f is called an **X-partition** of f if the primes are consistent (i.e. $p_i \neq \text{False}$ for all i) and their disjunction is valid (i.e. $p_1 \vee \dots \vee p_n = \text{True}$).

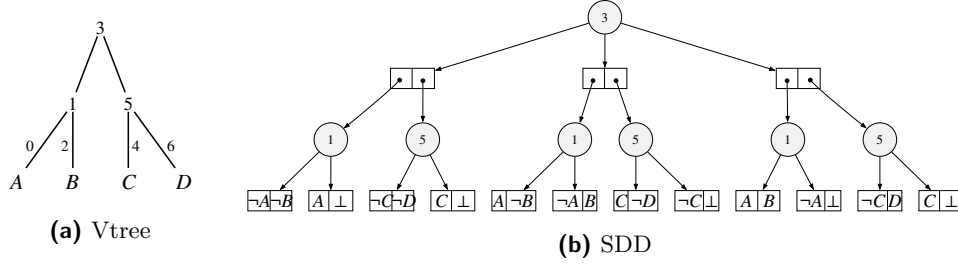


Figure 1: A Vtree and an SDD for the half-adder circuit.

Vtrees. A *vtree* is a rooted full binary tree whose leaves are in one-to-one correspondence with the elements of a set of variables $\mathbf{X}$.

SDD structure. An SDD α respects a vtree v if one of the following holds:

- α is a terminal node, that is, v contains a single variable X and either $\alpha = X$, or $\alpha = \neg X$, or $\alpha = \text{True}$, or $\alpha = \text{False}$; or
- there is an $\mathbf{X}$ -partition $(p_1 \wedge s_1) \vee \dots \vee (p_n \wedge s_n)$ of the boolean circuit represented by α , where the p_i resp. s_i are SDDs built over the left resp. right subtree of v . In this case, we call $\alpha = \{(p_1, s_1), \dots, (p_n, s_n)\}$ a *decision node*. Moreover, if they respect the whole child trees instead of subtrees, then we say α is *normalized* for v .

With this structural definition, we can build an SDD for a boolean function f on top of a vtree by finding an $\mathbf{X}$ -partition of f and recursively decomposing it until we build the leaf literal nodes of the vtree.

PSDDs. Probabilistic SDDs (PSDDs) extend normalized SDDs with probability distributions [KVdBCD14]. For every node of an SDD normalized for a vtree v , we define an additional parameter θ which allows us to recursively define the distribution P_α of α :

- For a terminal node α and a vtree $v = X$, we have that:
 - if $\alpha = X$, then $P_\alpha(X) = 1$ and $P_\alpha(\neg X) = 0$.
 - if $\alpha = \neg X$, then $P_\alpha(X) = 0$ and $P_\alpha(\neg X) = 1$.
 - if $\alpha = \text{True}$, then $P_\alpha(X) = \theta$ and $P_\alpha(\neg X) = 1 - \theta$.
 - if $\alpha = \text{False}$, then $P_\alpha(X) = 0$ and $P_\alpha(\neg X) = 0$.
- For a decision node $\alpha = \{(p_1, s_1), \dots, (p_n, s_n)\}$, $\theta = (\theta_1, \dots, \theta_n)$ with $\sum_i \theta_i = 1$ and $\theta_i = 0 \iff s_i = \text{False}$. Then, for an instantiation $\mathbf{x}$ of the primes and $\mathbf{y}$ of the subs, only one prime p_i is satisfied by $\mathbf{x}$ (by strong determinism), so we define

$$P_\alpha(\mathbf{xy}) = P_{p_i}(\mathbf{x}) \times P_{s_i}(\mathbf{y}) \times \theta_i.$$

By parametrizing an existing SDD, PSDDs can represent an arbitrary number of probability distributions. For SASCA applications, we can compute all the summations and products needed for message passing as in eqs. (1) and (2) in time polynomial in the size of the PSDD. As such, they act like any regular factor in the factor graph.

Illustrative example. We illustrate the PSSD formalism on a half-adder, a boolean circuit over inputs $A, B \in \{0, 1\}$ computing a sum bit $C = A \oplus B$ and a carry bit $D = A \wedge B$.

Figure 1 shows a vtree v for the half-adder, and the corresponding SDD normalized for v . The root of v is Node 3, whose left subtree (Node 1) holds $\{A, B\}$ and right subtree (Node 5) holds $\{C, D\}$. The SDD is rooted at a decision node α at vtree node 3 with three elements $\{(p_1, s_1), (p_2, s_2), (p_3, s_3)\}$, where the primes p_i are normalized SDDs over $\{A, B\}$ and the subs s_i are normalized SDDs over $\{C, D\}$:

$$\begin{aligned} p_1 &= \neg A \wedge \neg B, & s_1 &= \neg C \wedge \neg D, \\ p_2 &= (A \wedge \neg B) \vee (\neg A \wedge B), & s_2 &= C \wedge \neg D, \\ p_3 &= A \wedge B, & s_3 &= \neg C \wedge D. \end{aligned}$$

We parametrize α with $\theta = (\theta_1, \theta_2, \theta_3)$, with each θ_i corresponding to p_i , and $\sum_i \theta_i = 1$. The θ_i are set to reflect the marginal distribution P_{AB} of P_α over the left variables $\{A, B\}$:

$$\theta_i = \sum_{\mathbf{x} \models p_i} P_{AB}(\mathbf{x}),$$

For this example, we take $P_{AB}(A=1) = P_{AB}(B=1) = 0.75$. Summing the mass placed on each prime's support gives

$$\begin{aligned} \theta_1 &= P_{AB}(A=0, B=0) = 0.25 \times 0.25 = 0.0625, \\ \theta_2 &= P_{AB}(A=1, B=0) + P_{AB}(A=0, B=1) = 0.75 \times 0.25 + 0.25 \times 0.75 = 0.375, \\ \theta_3 &= P_{AB}(A=1, B=1) = 0.75 \times 0.75 = 0.5625. \end{aligned}$$

The joint distribution P_α is then defined recursively by

$$P_\alpha(\mathbf{xy}) = P_{p_i}(\mathbf{x}) \times P_{s_i}(\mathbf{y}) \times \theta_i \quad \text{where } p_i \text{ is the unique prime satisfied by } \mathbf{x}.$$

The marginal of any variable is obtained by summing P_α over all remaining variables. For instance, for $C = 1$ we have

$$\Pr(C = 1) = \sum_{A, B, D} P_\alpha(A, B, C = 1, D).$$

By strong determinism, exactly one prime p_i is satisfied for each instantiation of the variables (A, B) , so the sum splits over elements:

$$\Pr(C = 1) = \sum_i \theta_i \left(\sum_{\mathbf{x} \models p_i} P_{p_i}(\mathbf{x}) \right) \left(\sum_D P_{s_i}(C = 1, D) \right).$$

The first inner sum is 1 because P_{p_i} is a distribution over the satisfying assignments of p_i , so writing $P_{s_i}(C = 1) := \sum_D P_{s_i}(C = 1, D)$, we have

$$\Pr(C = 1) = \sum_i \theta_i \cdot P_{s_i}(C = 1).$$

Evaluating $P_{s_i}(C = 1)$ for each sub gives $P_{s_1}(C = 1) = 0$, $P_{s_2}(C = 1) = 1$, $P_{s_3}(C = 1) = 0$, and therefore

$$\Pr(C = 1) = \theta_2 = 0.375.$$

Similarly, we get $\Pr(D = 1) = \theta_3 = 0.5625$.

This is the key computational advantage of the SDD structure: because $\{p_1, p_2, p_3\}$ is an $\{A, B\}$ -partition, marginalizing an output variable only requires summing the θ_i of the elements whose sub satisfies that variable, with no enumeration over the full joint space.

2.3 SHA-256 specification

Figure 3 illustrates the high-level structure of SHA-256 [Nat15], built using the Merkle–Damgård architecture on a compression function **Compress** (Algorithm 1) mapping 512-bit message blocks and 256-bit chaining values (initialized with the public *initialization vector* IV). The input P is padded and divided into 512-bit chunks $P_0, \dots, P_{n-1}$, each treated by one **Compress** call. The hash output Z is the output of the last **Compress** in the chain.

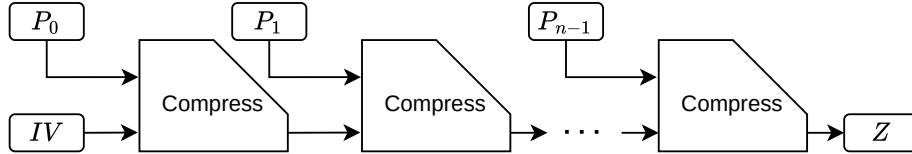


Figure 3: The Merkle–Damgård construction in SHA-2.

Algorithm 1 (**Compress**) is the SHA-256 compression function. We view the 512-bit input message block M as sixteen 32-bit words $(M_0, \dots, M_{15})$. First, 48 rounds of *message expansion* compute the extended message $(W_0, \dots, W_{63})$ using the operations

$$\sigma_0(a) := (a \ggg 7) \oplus (a \ggg 18) \oplus (a \gg 3), \quad (3)$$

$$\sigma_1(a) := (a \ggg 17) \oplus (a \ggg 19) \oplus (a \gg 10), \quad (4)$$

where $a \ggg i$ denotes a bitwise right rotation of a by i positions, $a \gg i$ denotes a right-shift by i positions, and $\oplus$ denotes bitwise XOR. Then, the chaining value is updated with 64 *compression* rounds composed of the 32-bit boolean operations

$$\Sigma_0(a) := (a \ggg 2) \oplus (a \ggg 13) \oplus (a \ggg 22), \quad (5)$$

$$\Sigma_1(a) := (a \ggg 6) \oplus (a \ggg 11) \oplus (a \ggg 25), \quad (6)$$

$$\text{Ch}(a, b, c) := (a \& b) \oplus (\neg a \& c), \quad (7)$$

$$\text{Maj}(a, b, c) := (a \& b) \oplus (a \& c) \oplus (b \& c) \quad (8)$$

where $\&$ denotes bitwise logical-and, and $\neg$ denotes bitwise complement.

Algorithm 1: The SHA-256 compression function **Compress**. All values are 32-bit words. Here $\boxplus$ denotes 32-bit modular addition; σ_0 , σ_1 , Σ_0 , Σ_1 , **Ch** and **Maj** are defined in (3), (4), (5), (6), (7), and (8), respectively.

Data: Data block $M = (M_0, \dots, M_{15})$ and chaining value $V = (V_0, \dots, V_7)$

Result: The chaining value **Compress**(V, M)

```

1  $(W_0, \dots, W_{15}) \leftarrow (M_0, \dots, M_{15})$  ;
2 for  $t = 16$  to  $63$  do                                     // 48 message expansion rounds
3    $W_t \leftarrow \sigma_1(W_{t-2}) \boxplus W_{t-7} \boxplus \sigma_0(W_{t-15}) \boxplus W_{t-16}$ 
4  $(A, B, C, D, E, F, G, H) \leftarrow (V_0, \dots, V_7)$  ;
5 for  $t = 1$  to  $64$  do                                         // 64 compression rounds
6    $T_1 \leftarrow H \boxplus \Sigma_1(E) \boxplus \text{Ch}(E, F, G) \boxplus K_t \boxplus W_t$  ;
7    $T_2 \leftarrow \Sigma_0(A) \boxplus \text{Maj}(A, B, C)$  ;
8    $(H, G, F, E, D, C, B, A) \leftarrow (G, F, E, D \boxplus T_1, C, B, A, T_1 \boxplus T_2)$ 
9 return  $(V_0 \boxplus A, \dots, V_7 \boxplus H)$ 
  
```

3 Attacker model

As in all SASCAs, we consider a profiled attack scenario: we assume the adversary can train an arbitrary number of classifiers either on the target board, or on a clone, with controlled data. As in every profiled attack, the work presented in this paper is affected by problems such as clone availability and template portability. We also assume the adversary can perform leakage assessment on an arbitrary number of intermediate variables of the SHA-2 computation.

By default, we restrict the attacking phase to the observation of a single side-channel trace. The goal is to provide a ground estimation of the noise requirement for a successful attack. This model can be relaxed for some use-cases: for example, the attacks in [Subsection 5.4](#) can be conducted as long as the same key is used for every HMAC computation, allowing the attacker to reproduce the measurements regardless of the input message. In this situation, the attacker could use these multiple measurements to obtain more accurate predictions on intermediate variables.

3.1 Variable representation

We need to choose a size for representing the internal variables of SHA-256. We saw in [Section 2](#) that SHA-256 operates on 32-bit variables. While 32-bit variables can be templated [\[CDSU23\]](#), manipulating probability distributions on 2^{32} possible values requires a high amount of storage. We also need to consider the cost of the message-passing algorithm. Indeed, resorting to naive factors, with an enumeration cost of 2^{32} , would give very long propagation times. Additionally, we did not opt for a cluster-based graph (see [Subsection 1.2](#)), because the rotations in σ_0 , σ_1 , Σ_0 , and Σ_1 are not well-suited to chunk-optimized BP factors (unlike the Keccak subroutines in the attack of [\[KPP20\]](#)). Similarly, for modular addition, it is possible to build convolution-based factors to add two words represented on fragments. However, adding 4 or 5 words (as required by SHA-2 subroutines) would take several iterations and decrease inference performance.

We therefore chose a bit-level approach. Despite the information loss inherent in bit-level marginalization, this allows direct propagation between the input and output of each of SHA-2's internal operations, as well as modular additions, thanks to PSDD-based factors.

Remark 1. Another profiling strategy is the fragment template approach of [\[YK21\]](#), which applies LDA-based templates to fragments of the 32-bit words, before marginalizing to bit-level probability tables. In their particular case-study of Keccak, they observed that fragment size had little impact on the final accuracy of bit-level predictions. Thus, they viewed the direct training of bit-level templates as relevant as targeting higher-size chunks before marginalizing to bits, at the cost of higher training times.

4 SHA-256 Factor Graph

We now describe our construction of the factor graph of SHA-256 for our SASCAs.

4.1 Variables

We use the following naming conventions for the variable nodes in the factor graph:

- The expanded message consists of 64×32 variables $\{w_{i,j} : 0 \leq i < 64, 0 \leq j < 32\}$, where $w_{i,j}$ represents the j -th bit of the i -th 32-bit word in big-endian encoding (so, for example, $w_{1,0}$ is the most significant bit of the second 32-bit word).

- We write $V^t = (A^t, B^t, C^t, D^t, E^t, F^t, G^t, H^t)$ for the intermediate state at compression round t of **Compress**. Each component of V^t represents a set of 32 single-bit variables. These variables are denoted by $v_{i,j}^t$ with $0 \leq i < 8$ (so, for example, $v_{3,4}^6$ denotes the fifth most significant bit of D^6).
- The input vector IV and the constant vectors K are defined as constants in the graph. Unlike variables, constants never update their marginal distributions.

4.2 Bit-level factors

We need to build a bit-level factor graph representing the **Maj**, **Ch**, σ_0 , σ_1 , Σ_0 and Σ_1 routines, as well as 4-word and 5-word adders, on our bit-level variable representation.

Inner routines. Following You and Kuhn [YK21], we consider each routine as a subgraph composed of elementary bit-level factors, chained to represent the whole SHA-2 computation. These subgraphs represent the inner logic of the message expansion and the compression rounds. For example, the **Ch** and **Maj** operations on 32-bit words are divided into 32 bit-level transforms (see Figure 4 and Figure 5). Similarly, we build the subgraphs for σ_0 , σ_1 , Σ_0 , and Σ_1 by connecting matching bits after shifting or rotation.

Modular addition. The last operation we must represent is modular addition of four and five 32-bit words. We used a *carry-select adder* as our baseline adder. *Carry-select adders* use multiplexers to select carry outputs from different levels of the adder. This design has fewer levels of carry propagation compared to ripple-carry adders, reducing the circuit depth and improving performance in SASCA applications. However, its gate-level depth is still $\sqrt{n}$, where n is the number of bits of the added word. They also introduce loops inside the subgraph, which degrade the performance of the message-passing algorithm. The alternative is a simple ripple-carry adder—but while loop-free, their slow propagation times make them impractical for SASCA applications.

4.3 PSDD-based bit-level factors

PSDDs allow us to build factors for SASCA applications with two significant advantages over the naive bit-level factors of Subsection 4.2: they guarantee exact loop-free inference, and let us perform the inference for the whole routine in one iteration. This has a concrete impact on inference quality:

- For modular addition, we can perform exact inference for the addition of 4 or 5 words in one iteration, instead of several iterations for loopy bit-level carry-select adders.
- For SHA-2 subroutines, we can perform exact loop-free inference for **Ch**, **Maj**, σ_0 , σ_1 , Σ_0 , and Σ_1 . This is also achievable with baseline bit-level factors by merging them together, similar to what was done for Kyber’s NTT butterfly factor in [PP19].

PSDD-based factors are constructed as follows:

- For each target operation (4/5-word modular addition, σ_0 , σ_1 , Σ_0 , Σ_1 , **Maj**, and **Ch**) we built the CNF of the corresponding boolean function using the Z3 solver [DMB08] (via the Z3Py interface in Python), which uses the Tseytin transformation to produce a CNF whose size grows linearly with the initial circuit size.
- We used the CNF-to-SDD compiler from [CD13] to compile our CNF into an SDD with dynamic minimization. We merged **Maj** and Σ_0 into a single SDD, and similarly for **Ch** and Σ_1 . We did not succeed in compiling a single SDD for the whole computation of T_1 or T_2 .

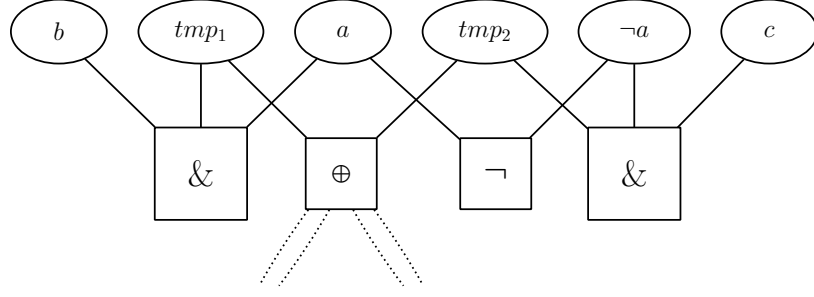


Figure 4: Baseline bit-level subgraph for the Ch operation.

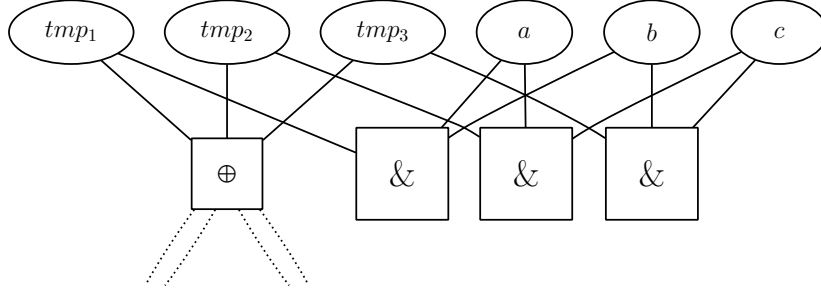


Figure 5: Baseline bit-level subgraph for the Maj transform.

- Finally, we integrated this SDD as a regular factor of our factor graph.

4.4 Factor Graph

The last step is to build the factor graph itself.

Factor graph construction. With all the PSDD factors compiled, we build the factor graph by wiring the variable and factor nodes based on the structure of SHA2 functions. Figures 6 and 7 present the subgraphs for a single round of expansion and compression, respectively. All variable nodes in these figures actually represent 32 single-bit variable nodes.

Adding leakage. To each variable node of every inter-round variable $v_{i,j}^t$ and every expanded message variable $w_{i,j}$, we connect an additional prior factor which transmits the outcome of the template matching at each round.

In all our experiments, we assume that all expanded message and intermediate state words leak during the hash computation, typically when loaded/stored. To keep the assumptions as implementation-agnostic as possible, we do not add intermediates leaking values.

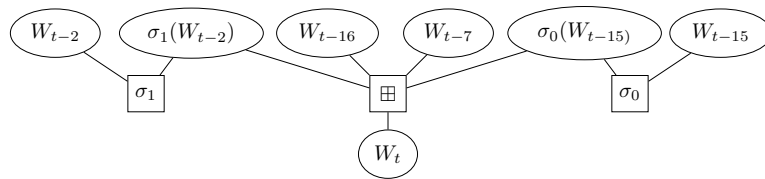


Figure 6: PSDD-based bit-level subgraph for round t of the expansion function.

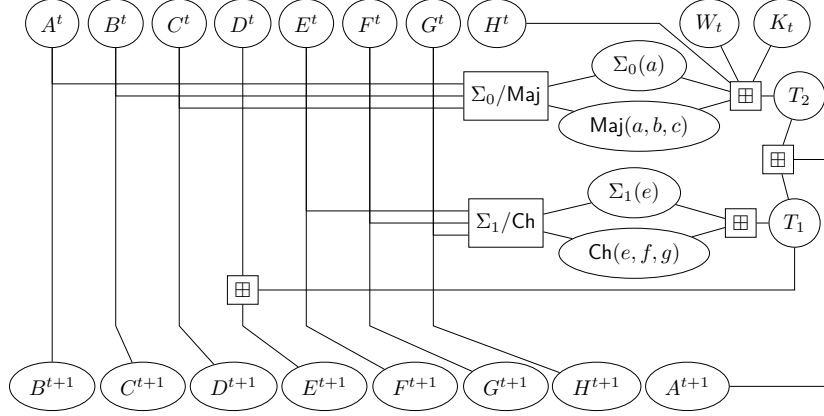


Figure 7: PSDD-based bit-level subgraph for round t of the compression function.

In practice, this model can be refined: we expect intermediate computations inside the expansion and compression phases to provide further leakage on internal variables. These could be integrated in the factor graph by creating additional prior factors. Alternatively, they could be included during the training of neural networks, thus improving the template accuracy of variables already represented in the graph.

5 SASCA simulations

We now explore a general attack scenario where an adversary can apply previously-crafted templates to a measurement of a single hash-function call. These simulations show the impact of noise and of the number of targeted SHA-2 rounds on the success of the attack.

5.1 Leakage Simulation

As stated in the previous section, we consider a bit-level leakage model for our experiments. To simulate side-channel attacks with varying noise levels, we used the notion of *virtual leakage* introduced in [MHL^C24]: for each bit b of our leaking variables, instead of considering the multivariate leakage, we consider its projection L_V in the latent space of our virtual classifier:

$$L_V = \alpha \cdot b + \beta, \quad \beta \sim \mathcal{N}(\mu, \sigma^2) \quad (9)$$

where α is a multiplicative coefficient and β is a Gaussian noise parameter. Different levels of noise can be simulated by modifying σ , the standard derivation of the noise distribution.

Setting α to 1 and μ to 0, we simulate the output of a bit-level classifier by performing random sampling of predictions following Equation 9 with an increasing noise level. Note that this model is only used in this section and will be dropped in Section 6.

In the sequel, we use bit-level accuracy instead of σ to characterize the template accuracy required for attack success. This accuracy can be computed from the CDF of the normal distribution as

$$\text{Acc} = P(L_V < 0.5 \mid b = 0).$$

Figure 8 illustrates the evolution of bit-level template accuracy as noise increases with σ .

Limitations. Bit-level leakage modeling allows us to lead attack simulations with varying noise levels to evaluate the robustness of targeted algorithms against SASCA. This model is a much simpler version of the classifier’s predictions than what may be observed in the

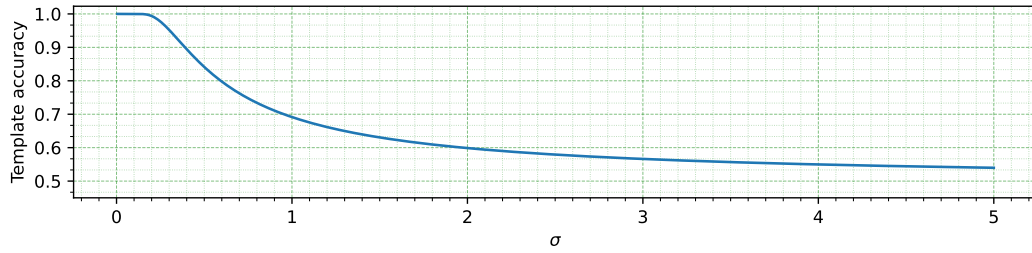


Figure 8: Accuracy of a bit-level classifier as a function of the standard deviation *sigma* of the noise.

real world: it supposes all bits are predicted with the same accuracy, which is often not the case in real scenarios [CLH19].

5.2 General attack considerations

We will now give some details on the parameters used for our simulations.

Simulation parameters. For every scenario presented in this section, we varied the parameter σ (as introduced in Equation 9) between 0 and 1.4 with a step of 0.1. This corresponds to an average bit-level classifier accuracy of between 0.64 and 1, which we also include in the graph for clarity. For every value of σ , we performed 30 new independent experiments, with a new random input and new random initialization for leakage generation, and then computed the mean rank and mean success rate.

For each scenario, we targeted between 16 rounds (the minimum to template all key bits) and 50 rounds. Each run of the BP algorithm was performed for a maximum of 300 iterations, stopping earlier if the messages arriving to the hash function input block changed by less than 0.01 for at least 5 consecutive iterations.

Message scheduling. Early works on SASCA suggested custom message-scheduling between the factors of the graph improved performance and inference stability [KPP20, PP19]. However, more recent studies [NCK⁺24] concluded that structured scheduling such as *in-order scheduling* (propagating messages from the input of the algorithm to the output and back) has at best no impact on inference performance, and in some cases degrades it. Based on these observations and the irregularity of the SHA-2 factor graph, we did not use any custom scheduling in our experiments.

Baseline attack. As a basis for comparison in the rest of this section, and to highlight the correction capacity of SASCA, we include a reference baseline attack score for each of our simulations, corresponding to a naive template attack on the hash function input.

Rank estimation. In addition to the success rate, we provide estimations of the number of key enumerations needed to recover the key after propagation. We used Poussier, Standaert, and Grosso’s key rank estimation algorithm [PSG16], which convolves histograms containing the log-probabilities of the key. A trade-off must be found for the number of bins: more bins yields a more accurate estimation at the cost of higher time and space complexity. We used 50 bins in all our experiments.

5.3 SHA-256

Figure 9 depicts our simulation results for a single call to SHA-256. The chaining value is the default SHA-256 IV , which is of course known to the attacker. For this simulation, the attacker can gain probabilistic information (*i.e.*, likelihoods from bit-level template attacks) on the expanded message W and the intermediate variables of a certain number of compression rounds.

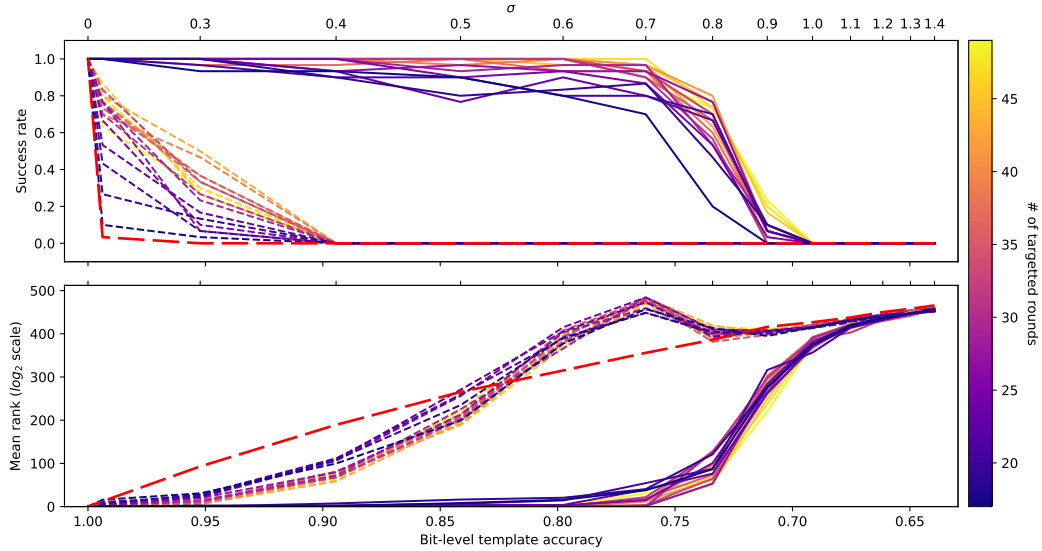


Figure 9: Recovery success rate (top) and mean rank of the correct input (bottom) after BP for a fully unknown SHA-256 input. Results are shown for every two rounds. Dashed lines correspond to experiments ran with baseline bit-level factors; solid lines correspond to PSDD-based factors. Red dotted lines report the baseline attack success (corresponding to a naive TA on the target variable, without SASCA), as explained in Subsection 5.1.

Resistance to noise. The first interesting observation is the difference in attack performance between baseline factors and PSDD-based factors. In the best scenario, the former does not converge—or converges to the wrong input—when bit-level template accuracy drops below around 88%, thus making it impractical; whereas the latter achieves substantially better performance, with a success rate above 50% when bit-level template accuracy exceeds 73%. This performance difference shows the impact of the PSDD-based factors described in Subsection 4.3: exact propagation between the inputs and outputs of the modular addition factors is crucial for the quality of the marginals’ estimation.

Impact of the number of targeted rounds. For both approaches, templating and including more rounds of compression and expansion leads to better performance, illustrating SASCA’s ability to gather information from additional rounds. For the PSDD-based approach, the impact of the number of targeted rounds reduces after 32 rounds, illustrating the difficulty of BP in gathering information on the hash input from expansion rounds in which no input word is involved. Targeting 32 rounds or more allows success rates above 90% for bit-level accuracies above 76%.

Interest of static input recovery. Our attack does not require variability of the input, which makes possible new attack paths on SHA-2-based cryptosystems. For instance, we eliminate the limitations of the previous DPA on XMSS [KGB⁺18]. Indeed, in the

construction recommended by the XMSS Internet Draft [HBGM16], $\text{SHA-256}(0x00 \dots 03 \parallel \text{seed} \parallel i)$, the counter i does not provide the variability on its MSBs required for DPAs.

5.4 Targeting SHA-256 HMAC

HMAC is a data authentication scheme built from a hash function h : in our case, SHA-256. SHA-256 based HMAC is still widely used by default in many applications including IPsec, SNMPv3, EIGRP, and SSH. Formally, HMAC is defined by

$$\text{HMAC}_K(P) := h((K \oplus \text{opad}) \parallel h((K \oplus \text{ipad}) \parallel P)), \quad (10)$$

where P is the data to be authenticated, K is a 256-bit shared secret key, and ipad and opad are public parameters of the scheme. The data flow is illustrated in Figure 10: we observe that the SHA-256 compression function processes two derivations of K .

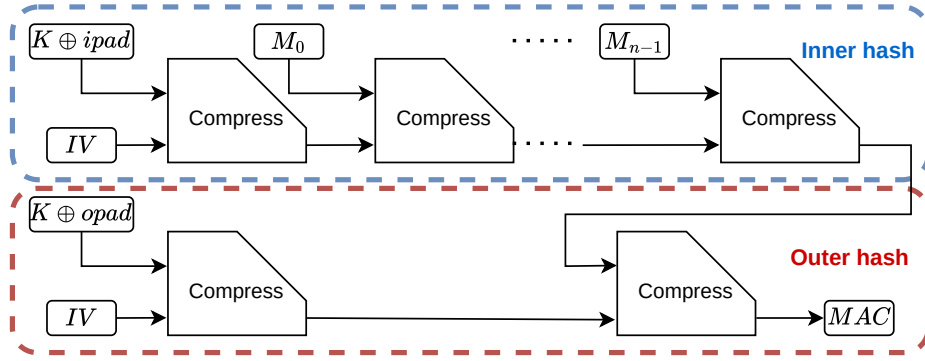


Figure 10: HMAC construction for SHA-256.

We propose an attack exploiting the structure of HMAC by combining leakages coming from both manipulations of the padded key to improve the chances of key recovery. Attackers can perform SASAs on the two SHA-256 calls $h(K \oplus \text{ipad})$ and $h(K \oplus \text{opad})$ simultaneously—specifically, on the first call to **Compress** in the inner and outer hashes, which only treat K with fixed padding and IV—by joining the two corresponding factor graphs with variables representing the bits of K . Thus, the attacker does not need variability, control, or knowledge of the message M .

Previous DPAs on HMAC-SHA-2 [BBD⁺13, BDT⁺23] begin by recovering the output of the first **Compress** of the inner hash, which digests $K \oplus \text{ipad}$. Then, the attacker can proceed with the recovery of the output of the first **Compress** of the outer hash, which digests $K \oplus \text{opad}$. Given these two outputs, we can forge HMAC tags for any message. In these attack paths, a successful attack on the first SHA-2 call is a prerequisite to the attack on the second call. On the other hand, by targeting the preceding call to the compression function in both invocations to the hash function, we can directly combine the information they convey. Also, for all previous DPAs [BBD⁺13, KGB⁺18, BDT⁺23], each attack actually consists of several DPAs. For instance, the attack of [BBD⁺13] requires 36 DPAs per SHA-2 call.

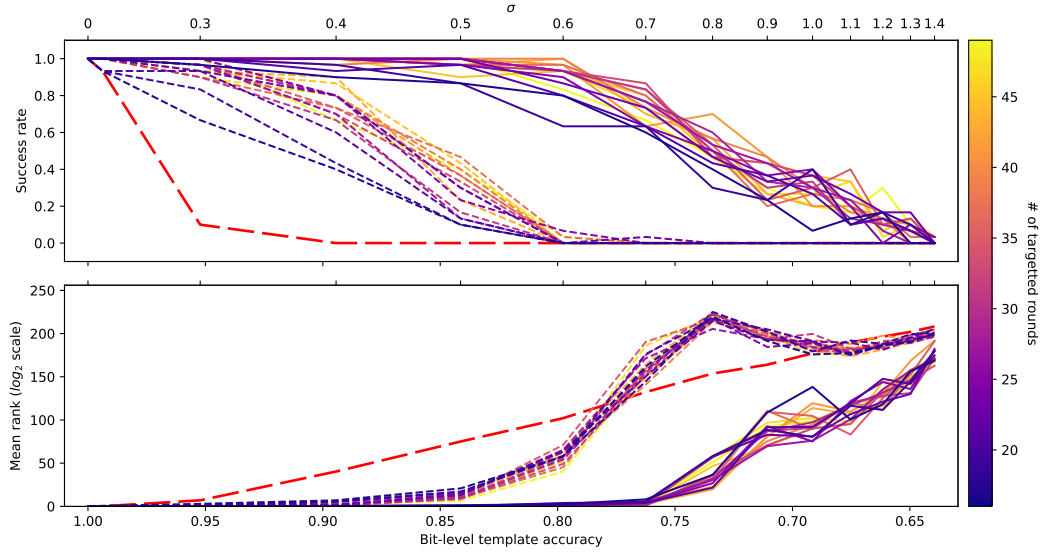


Figure 11: Success rate of the recovery (top) and mean rank of the correct input (bottom) after BP for a 256-bit HMAC key. Results are shown for every two rounds. Dashed lines correspond to experiments ran with baseline bit-level factors; solid lines correspond to PSDD-based factors. Red dotted lines report the baseline attack success (corresponding to a naive TA on the target variable, without SASCA), as explained in [Subsection 5.1](#).

Resistance to noise. [Figure 11](#) presents our simulation results. We see that combining the two factor graphs benefits the attacker in low-SNR experiments: the recovery ratio is higher for the double graph attack than for the simple graph one. In the HMAC scenario, we see that an attacker is able to recover the 256-bit key with a non-zero probability when bit-level predictions have an accuracy higher than 0.65.

Impact of the number of targeted rounds. Merging both SHA-2 calls in a single graph can have a negative impact on BP convergence: concatenating both hash function calls increases the total graph diameter, which increases the number of cycles in the graph as well as the propagation times. Unlike the single-graph scenario of [Subsection 5.3](#), increasing the number of targeted rounds here does not necessarily improve performance: in our simulations, performance starts to decrease after 40 rounds. The advantages of targeting more rounds becomes less clear for bit-level accuracies under 0.8.

Discussion. Merging the graphs corresponding to the processing of both padded derivations of the key benefits the attacker in lower-SNR cases. However, despite the gain of information and the increased noise resistance, we observe that the key-recovery success-rate drops below 1 at a lower noise level than in our SHA-256 attack. This suggests the BP still has room for improvement, since it does not optimally exploit all available information.

Our attack does not rely on the data M . Consequently, if the same key K is used to compute HMAC for several messages, an attacker can extract the compression function calls taking $K \oplus \text{ipad}$ and $K \oplus \text{opad}$ as input from each HMAC invocation to enhance template accuracy (*e.g.*, by averaging the traces), or to link them in a wider graph.

6 Attacking a device

The simulations of [Section 5](#) gave us the minimal accuracy necessary for a successful attack. In this section, we evaluate the bit-level profiling capabilities of an attacker. We

assess EM leakage of 32-bit loading operations on a microcontroller unit. To reach the template accuracy required by our attack, we use a multi-position approach which consists in performing acquisitions with different EM probe locations to better target every bit.

6.1 Experimental setup

Board setup. We performed our experiments on a STM32F303RET6 board, which packs a Cortex-M4 core. Electromagnetic emanations were measured over the package with a Langer RF-B 0.3-3 near-field probe, mounted on a motorized 3-axis stage. The signal was first amplified with a 50db low-noise amplifier and then acquired with a Tektronix MSO64 oscilloscope, at a sampling rate of 2.5GS/s.

Targeted operation. As stated in Section 4.4, we include side-channel leakage for all the variables of the expanded word and of the internal state of the compression function, which are expected to be loaded and/or stored during the execution. To compare the quality of this leakage on a real board with the requirements from our simulations, we recorded a sequence of 2 consecutive loading instructions of random 32-bit words from memory, and trained templates to predict the values of the first loaded word.

6.2 Template construction

Acquiring traces. As our simulation in Subsection 5.4 suggests, side-channel attacks on HMAC-SHA-2 require high signal-to-noise leakage. If we consider that a 32-bit loading operation leaks with a noise-free HW model, then a bit-level classifier would perform with an average accuracy of 0.57 and the attack would fail.

To improve the accuracy of our templates, we acquired EM emanations at multiple positions over the board. This approach has already been used in the context of side-channel-based disassembly: in [CLH19], Cristiani, Lecomte, and Hiscock [CLH19] achieved bit-level recovery of 14-bit instructions fetched on an 8-bit PIC16F microcontroller, with an accuracy of more than 98% for every bit, using a greedy search algorithm to find the best positions for every bit.

We followed a simpler approach: we recorded traces for 10 000 random 32-bit words at each position on a 5×5 grid over the chip package, with a step of 0.46mm in both directions. As in [SBO⁺15], we then concatenated traces corresponding to the same input and trained 32-bit-level classifiers with those traces. To improve the accuracy of our predictions, we recorded up to 1000 acquisitions per input before averaging them, for a total of 250M acquisitions.

Template training. We first performed a dimensionality reduction with Principal Component Analysis to the concatenated traces, keeping 500 components. Then, we trained Linear Discriminant Analysis classifiers on the reduced traces, for every bit of the word. Of the 10 000 averaged traces, 9000 were used for training and 1000 for testing the attack. Table 1 shows the accuracies of all 32 templates for every bit, with an increasing number of total traces. As Figure 12 shows, the best positions for each bit are well-distributed over the grid, validating the multi-position approach.

6.3 Attack evaluation

We evaluated the correction capabilities of our SASCA on both SHA-2 and HMAC-SHA-2 as follows:

- We performed 300 experiments for each scenario (between 10 and 1000 averaged traces per position, for a total of between 250 and 25 000 traces per input).

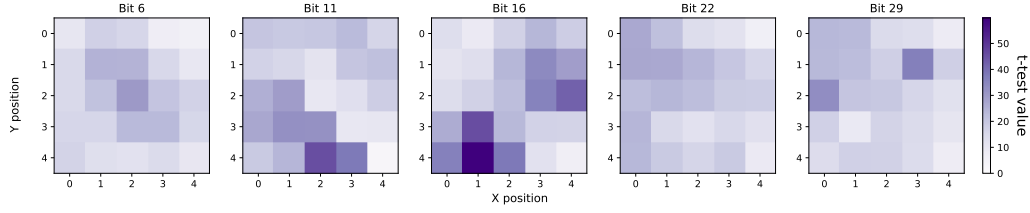


Figure 12: t -test maps for bits 6, 11, 16, 22, and 29 of the loaded 32-bit word. The variability of the position of the best leakage point, as well as the differences in magnitude between bits, validates the probe multi-position approach.

- For each experiment, we generated the template output for every bit of every leaking variable in our graph by randomly choosing an element of the set of predictions for the bit value. Then, we ran our BP algorithm for a maximum of 300 iterations, stopping when messages arriving at the target variables changed by less than 0.01 for at least 5 consecutive iterations.

Figure 13 shows the success rate of both scenarios. As the simulations suggest, merging graphs in the HMAC scenario yields a higher success rate in the lower trace count (under 5000 traces), making recovery possible with as little as 1250 attack traces. Above 5000 traces, the increased graph complexity reduces attack performance: the recovery of a full SHA-256 input reaches 100% success rate with fewer traces than HMAC-SHA-2. We note that in nearly every experiment where BP reached the maximum number of iterations, the resulting guess was incorrect. This link between non-convergence and incorrect guesses has already been observed in previous attacks using BP against a variety of targets [PP19, MHL24]. This information can be used by attackers to filter unsuccessful attempts and repeat the attack.

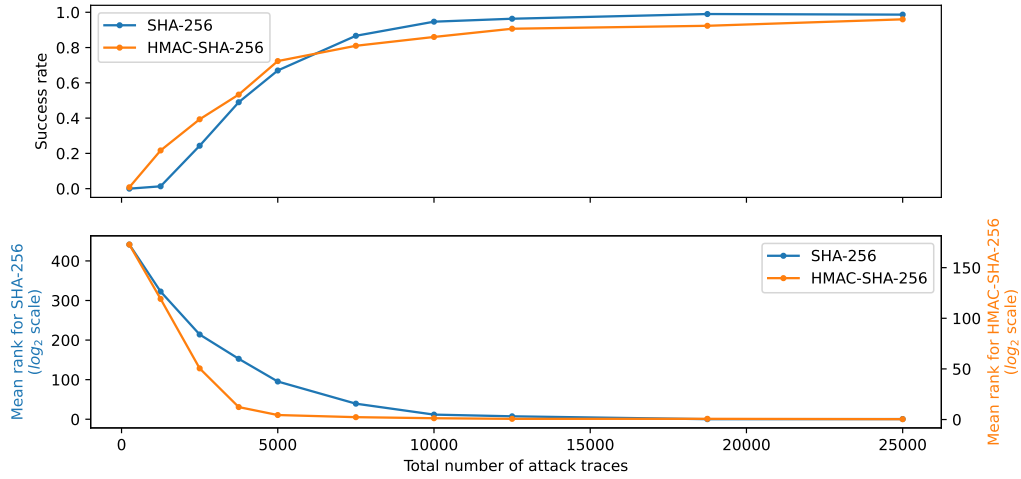


Figure 13: Success rates (top) and mean ranks (bottom) of the recovery of SHA-256 input and HMAC-SHA-256 key, with fixed input.

7 Countermeasures

We now consider some countermeasures against our attacks.

Table 1: Bit-level template accuracy for increasing numbers of attack traces.

Bit	Total attack traces				
	250	1250	5000	12 500	25 000
0	0.661	0.745	0.841	0.861	0.870
1	0.593	0.660	0.691	0.725	0.726
2	0.634	0.685	0.719	0.757	0.760
3	0.657	0.731	0.788	0.828	0.860
4	0.685	0.773	0.816	0.842	0.860
5	0.672	0.739	0.796	0.825	0.852
6	0.687	0.751	0.818	0.870	0.892
7	0.682	0.736	0.795	0.812	0.830
8	0.708	0.753	0.823	0.865	0.887
9	0.652	0.696	0.762	0.802	0.801
10	0.621	0.679	0.728	0.761	0.777
11	0.762	0.815	0.858	0.891	0.900
12	0.677	0.727	0.779	0.834	0.855
13	0.632	0.692	0.709	0.770	0.787
14	0.673	0.739	0.771	0.804	0.830
15	0.692	0.752	0.808	0.833	0.859
16	0.781	0.875	0.947	0.958	0.973
17	0.602	0.660	0.676	0.707	0.726
18	0.613	0.659	0.688	0.705	0.727
19	0.629	0.670	0.749	0.768	0.787
20	0.636	0.722	0.773	0.810	0.823
21	0.644	0.735	0.804	0.818	0.846
22	0.668	0.728	0.778	0.813	0.810
23	0.597	0.646	0.688	0.720	0.758
24	0.627	0.671	0.728	0.761	0.803
25	0.613	0.633	0.660	0.713	0.734
26	0.605	0.676	0.698	0.719	0.753
27	0.649	0.661	0.718	0.765	0.764
28	0.713	0.750	0.801	0.838	0.844
29	0.694	0.790	0.853	0.885	0.908
30	0.635	0.731	0.918	0.988	0.999
31	0.583	0.616	0.635	0.656	0.691

Masking. Some secure implementations of HMAC-SHA-2 have been proposed [MTMM07, HWZ18], but masking SHA-2 is usually costly since it requires switching between arithmetic and boolean masking. To the best of our knowledge, no successful full-fledged attack has ever been conducted against a masked implementation of SHA-2.

Masking as a SASCA countermeasure has been studied in the context of Kyber [PPM17, PP19, HHP⁺21]. In these cases, the limited range of the target secret enabled attacks even on first-order masking, though with lower noise requirements. Similarly, chosen-ciphertext strategies [HHP⁺21] allowed for successful secret recovery at noise levels similar to those in unmasked scenarios. These mitigations against masking do not apply here, so we expect masking to be a successful countermeasure to our proposed attacks.

Still, the predominance of unmasked SHA-2 and HMAC-SHA-2 implementations in real-world deployments justifies our study of attacks on unmasked implementations.

Shuffling. Shuffling (randomizing the order of operations) has been extensively proposed as a side-channel countermeasure. In the context of SASCA, shuffling breaks the determin-

ism of the construction of the factor graph, making inference much less informative. Some adaptations to the factor graph were proposed when shuffling is used against attacks with higher noise tolerance [HSST23]. However, the impact of shuffling and these adaptations in scenarios with high signal-to-noise requirements demands further investigation. In any case, randomizing the processing order in SHA-256 is no easy task: data dependencies (in SHA-256 message expansion, for example) limit opportunities to shuffle operations.

Hiding. The attacks presented in this paper are valid up to a certain level of noise, so artificial extra noise and jitter are a potential countermeasure. Hence, classical measures that limit power consumption variations, or that insert dummy cycles during sensitive code execution, may be beneficial. Indeed, these measures would complicate the template construction phase of the attack, and thus lower the SASCA accuracy.

8 Conclusion and Further Work

In this paper, we have presented new SASCAs against SHA-2 and HMAC-SHA-2, using Probabilistic Sentential Decision Diagrams to build efficient factor graphs for marginals inference. Through simulations, we showed the performance gain of these factors compared to their baseline counterparts. Compared to DPAs, our attacks do not require variability of the input, allowing us to consider attack paths on SHA-2-based cryptosystems. Using this framework, we showed how attackers can exploit the two invocations of the hash function in HMAC to improve their recovery capabilities.

Finally, we evaluated the practicality of our approach against a software implementation of SHA-2 on an STM32F303 board. To reach the necessary classifier accuracy, we performed multiple acquisitions of load operations over our chip. This resulted in 21% of successful attacks with as few as 1250 traces for HMAC-SHA-256, and 24% of successful attacks with as few as 2500 traces for SHA-256.

SASCA with variable input. Unlike previous side-channel attacks against SHA-2, the attacks presented in this paper do not need nor benefit from known variable input. However, SASCAs have already been used in scenarios with input variability [VGS14], so SASCA should be compared further with current template attacks against HMAC-SHA-2 [BDT⁺21].

Higher-dimension marginals representation. In this work, all variable marginals were represented at bit level. Compiling SASCA factors that allow for higher-dimension variables is expected to reduce the cost of marginalization from high-dimension templates. However, the cost of compilation of SDD-based factors is exponentially large as higher-dimension variables need to be represented. Further work is therefore needed to study the cost and benefit of such upgrades. Top-down compilers [OD15], which compile SDDs orders of magnitude faster than traditional bottom-up compilers, could be an interesting approach. For example, matching current template attacks on HMAC-SHA-2 would require reaching marginals representation on two-bit fragments.

The combination of a bit-level representation of SHA-2 in the factor graph with a multi-position acquisition approach allowed us to circumvent the computational cost of performing SASCAs on SHA-2. This approach could be generalized to other algorithms using large word sizes.

References

- [BBD⁺13] Sonia Belaïd, Luk Bettale, Emmanuelle Dottax, Laurie Genelle, and Franck Rondepierre. Differential power analysis of HMAC SHA-2 in the Hamming

- weight model. In Pierangela Samarati, editor, *SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography, Reykjavik, Iceland, 29-31 July, 2013*, pages 230–241. SciTePress, July 2013.
- [BCD20] Elaine Barker, Lily Chen, and Richard Davis. Recommendation for key-derivation methods in key-establishment schemes. Technical Report Special Publication 800-56C Revision 2, National Institute of Standards and Technology, 2020.
- [BDT⁺21] Yaacov Belenky, Ira Dushar, Valery Teper, Hennadii Chernyshchyk, Leonid Azriel, and Yury Kreimer. First full-fledged side channel attack on HMAC-SHA-2. In Shivam Bhasin and Fabrizio De Santis, editors, *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*, volume 12910 of *Lecture Notes in Computer Science*, pages 31–52. Springer, 2021.
- [BDT⁺23] Yaacov Belenky, Ira Dushar, Valery Teper, Vadim Bugaenko, Oleg Karavaev, Leonid Azriel, and Yury Kreimer. Carry-based differential power analysis (CDPA) and its application to attacking HMAC-SHA-2. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):1–29, 2023.
- [CAD⁺20] David A Cooper, Daniel C Apon, Quynh H Dang, Michael S Davidson, Morris J Dworkin, and Carl A Miller. Recommendation for stateful hash-based signature schemes. *NIST Special Publication*, 800(208):800–208, 2020.
- [CD13] Arthur Choi and Adnan Darwiche. Dynamic minimization of sentential decision diagrams. In Marie desJardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*, pages 187–194. AAAI Press, 2013.
- [CDSU23] Gaëtan Cassiers, Henri Devillez, François-Xavier Standaert, and Balazs Udvarhelyi. Efficient Regression-Based Linear Discriminant Analysis for Side-Channel Security Evaluations: Towards Analytical Attacks against 32-bit Implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):270–293, 2023.
- [CLH19] Valence Cristiani, Maxime Lecomte, and Thomas Hiscock. A bit-level approach to side channel based disassembling. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, volume 11833 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2019.
- [CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
- [Dar11] Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In Toby Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 819–826. IJCAI/AAAI, 2011.

- [DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'08/ETAPS'08*, page 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.
- [GS15] Vincent Grosso and François-Xavier Standaert. ASCA, SASCA and DPA with enumeration: which one beats the other and when? In *Advances in Cryptology–ASIACRYPT 2015: 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29–December 3, 2015, Proceedings, Part II 21*, pages 291–312. Springer, 2015.
- [HBG⁺18] Andreas Huelpling, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, May 2018.
- [HBGM16] Andreas Huelpling, Denis Butin, Stefan-Lukas Gazdag, and Aziz Mohaisen. XMSS: Extended Hash-Based Signatures. Internet-Draft draft-irtf-cfrg-xmss-hash-based-signatures-03, Internet Engineering Task Force, February 2016. Work in Progress.
- [HHP⁺21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen ciphertext k -trace attacks on masked CCA2 secure Kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):88–113, 2021.
- [HSST23] Julius Hermelink, Silvan Streit, Emanuele Strieder, and Katharina Thieme. Adapting belief propagation to counter shuffling of NTTs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(1):60–88, 2023.
- [HWZ18] Zhenhao He, Liji Wu, and Xiangmin Zhang. High-speed pipeline design for hmac of sha-256 with masking scheme. In *2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, pages 174–178. IEEE, 2018.
- [KE10] Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, May 2010.
- [KFL01] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.
- [KGB⁺18] Matthias J Kannwischer, Aymeric Genêt, Denis Butin, Juliane Krämer, and Johannes Buchmann. Differential power analysis of XMSS and SPHINCS. In *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings*, volume 10815 of *Lecture Notes in Computer Science*, pages 168–188. Springer, 2018.
- [KPP20] Matthias J Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on Keccak. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):243–268, 2020.

- [KVdBCD14] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press, 2014.
- [MHLC24] Julien Maillard, Thomas Hiscock, Maxime Lecomte, and Christophe Clavier. Simulating SASCA on Keccak: Security implications for post-quantum cryptographic schemes. In *21st International Conference on Security and Cryptography*, pages 518 – 527, Dijon, France, July 2024. SCITEPRESS - Science and Technology Publications.
- [MTMM07] Robert McEvoy, Michael Tunstall, Colin C Murphy, and William P Marnane. Differential power analysis of HMAC based on SHA-2, and countermeasures. In Sehun Kim, Moti Yung, and Hyung-Woo Lee, editors, *Information Security Applications, 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers*, volume 4867 of *Lecture Notes in Computer Science*, pages 317–332. Springer, 2007.
- [Nat02] National Institute of Standards and Technology. Secure hash standard (SHS). Technical Report Federal Information Processing Standards Publications (FIPS) 180-2, U.S. Department of Commerce, Washington, D.C., 2002.
- [Nat08] National Institute of Standards and Technology. The keyed-hash message authentication code (HMAC). Technical Report Federal Information Processing Standards Publications (FIPS) 198-1, U.S. Department of Commerce, Washington, D.C., 2008.
- [Nat15] National Institute of Standards and Technology. Security requirements for cryptographic modules. Technical Report Federal Information Processing Standards Publications (FIPS) 180-4, U.S. Department of Commerce, Washington, D.C., 2015.
- [NCK⁺24] Rishub Nagpal, Gaëtan Cassiers, Christian Knoll, Franz Pernkopf, Robert Primas, and Stefan Mangard. On loopy belief propagation for SASCAs: An analysis and empirical study of the inference problem. *IACR Communications in Cryptology*, 1(4):15, 2024.
- [OD15] Umut Oztok and Adnan Darwiche. A top-down compiler for sentential decision diagrams. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3141–3148. AAAI Press, 2015.
- [PP19] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, volume 11774 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2019.
- [PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, 2017.

- [PSG16] Romain Poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 61–81. Springer, 2016.
- [SBO⁺15] Daehyun Strobel, Florian Bache, David Oswald, Falk Schellenberg, and Christof Paar. SCANDALee: A Side-ChANnel-based DisAssembLer using local Electromagnetic Emanations. In Wolfgang Nebel and David Atienza, editors, *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*, pages 139–144. ACM, 2015.
- [VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.
- [WNC⁺25] Thomas Wedenig, Rishub Nagpal, Gaëtan Cassiers, Stefan Mangard, and Robert Peharz. Exact soft analytical side-channel attacks using tractable circuits. *arXiv preprint arXiv:2501.13748*, 2025.
- [YK21] Shih-Chun You and Markus G. Kuhn. Single-trace fragment template attack on a 32-bit implementation of Keccak. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*, volume 13173 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2021.
- [YKSH23] Shih-Chun You, Markus G. Kuhn, Sumanta Sarkar, and Feng Hao. Low trace-count template attacks on 32-bit implementations of ASCON AEAD. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(4):344–366, 2023.