MALEAK: Blind Side-Channel Key Recovery Exploiting Modular Addition Leakage in ARX-based Block Ciphers

Inhun Lee, GyuSang Kim, Seokhie Hong, and HeeSeok Kim

***Abstract*—Side-channel analysis (SCA) can recover secret keys by exploiting physical leakages emitted during cryptographic computations. Most SCA techniques, however, require knowledge of the plaintext or ciphertext corresponding to each measured trace, which may be unavailable in realistic adversarial settings. Blind side-channel analysis (Blind SCA), first introduced in 2014, relaxes this requirement, but existing work has mainly targeted S-box nonlinearities. We present a systematic study of blind SCA targeting modular addition, the core nonlinear operation in ARX-based block ciphers, by leveraging key-dependent statistical characteristics arising from carry propagation. We introduce MALEAK, a framework that models analysis targets as generalized key-dependent functions involving modular addition. We validate key recovery in simulation under varying noise levels and instantiate MALEAK for HIGHT and SPECK by deriving cipher-specific attack procedures. We further evaluate the MALEAK instantiations for HIGHT and SPECK on real power traces collected from an STM32F415 microcontroller (ARM Cortex-M4). Our results show that, given suitable points of interest (PoIs), MALEAK can recover secret keys of ARX-based block ciphers without access to the corresponding plaintext or ciphertext.**

***Index Terms*—Blind side-channel analysis, ARX-based block cipher, HIGHT, SPECK, modular addition**

I. INTRODUCTION

SIDE-CHANNEL Analysis (SCA) is an attack technique that infers secret information by exploiting physical leakages observable during cryptographic computations, such as power consumption, electromagnetic (EM) emanations, and execution time. Since Kocher introduced Differential Power Analysis (DPA) [1], a wide range of SCA methods have been proposed, including Correlation Power Analysis (CPA) [2], Mutual Information Analysis (MIA) [3]–[5], and Template Attacks (TA) [6]–[8]. These attacks pose a practical threat to embedded devices, as an adversary can infer secret internal states from physical leakages.

Side-channel attacks are commonly categorized into profiled and non-profiled attacks. Profiled attacks [9]–[23] assume that the adversary has access to a fully controllable clone device identical to the target. The adversary first builds a leakage model using side-channel traces collected from the clone device, and then applies this model to traces acquired from the real target device in order to recover the unknown secret key. In contrast, non-profiled attacks [24]–[36] use only traces from the target and exploit statistical relationships between predicted intermediate values for each key guess and measured leakages.

In general, both profiled and non-profiled techniques rely on the assumption that the adversary knows the plaintext or ciphertext corresponding to the collected side-channel traces. In realistic settings, however, an adversary may fail to correctly identify the inputs and outputs corresponding to each processed block. For example, in Offset Codebook (OCB) mode [37], each block is processed with an offset derived from the nonce and the secret key. Therefore, an adversary who does not know the secret key cannot directly observe or reconstruct the per-block inputs and outputs [38]. Under such constraints, conventional SCA methods are not directly applicable, motivating techniques that recover keys directly from side-channel traces.

To address this problem, blind SCA was introduced in 2014 [39] to recover secret keys from side-channel traces alone, without requiring plaintext or ciphertext. Blind SCA leverages the property that, for a key-dependent nonlinear function inside a cryptographic algorithm, the joint probability distribution between its inputs and outputs differs depending on the value of the secret key. Following this seminal work, a number of studies have investigated blind SCA for various cryptographic algorithms [38], [40]–[43].

Most prior blind SCA research has focused on S-box-based nonlinear structures as the analysis target [38]–[41]. However, nonlinear operations used in cipher design are not limited to S-boxes. In ARX-based block ciphers, for instance, modular addition is a fundamental nonlinear operation. Due to carry propagation, it induces inter-bit dependencies in the output, yielding a form of nonlinearity that differs from S-boxes. Nevertheless, to the best of our knowledge, a systematic treatment of blind SCA targeting modular addition has not yet been reported.

To fill this gap, we propose MALEAK, the first blind SCA framework that targets modular addition by exploiting leakages induced by carry propagation. To assess the feasibility of MALEAK, we define two generalized nonlinear functions that incorporate modular addition and perform experiments in a simulation environment. We further instantiate MALEAK for ARX-based block ciphers, specifically HIGHT [44] and SPECK [45]. By conducting blind SCA experiments in both simulation and real implementation settings, we demonstrate that secret key recovery is possible for ARX-based block ciphers using only side-channel traces, even without any plaintext or ciphertext information.

Inhun Lee and GyuSang Kim are with the School of Cybersecurity, Korea University, Seoul 02841, South Korea (e-mail: dldlsgns99, ks9509@korea.ac.kr).

Seokhie Hong is with the SmartM2M, Busan 48058, South Korea (e-mail: shhong@smartm2m.co.kr).

HeeSeok Kim is with the Department of AI Cyber Security, College of Science and Technology, Korea University, Sejong-si 30019, South Korea (e-mail: 80khs@korea.ac.kr).

Corresponding author: HeeSeok Kim.

A. Related Work

The concept of blind SCA was first introduced by Linge *et al.* [39]. Their key observation is that, for a key-dependent nonlinear function g in a cryptographic algorithm, the joint probability distribution of its input a and output $b = g(a, k)$ becomes distinguishable for different values of the secret key k. Building on this observation, they proposed a key recovery method that compares a theoretically derived distribution for each key hypothesis and an empirical distribution constructed from actual side-channel leakages. In their workflow, assuming that the leakage is correlated with the Hamming weight of the intermediate value, they first aligned traces at a PoI and then introduced a slicing technique that partitions the leakage range to estimate Hamming weights. They further combined this with a metric-based analysis to quantitatively measure the discrepancy between distributions.

Subsequently, Clavier *et al.* [40] proposed an improved analysis method based on a maximum likelihood criterion, achieving high key recovery success rates with fewer traces than the original metric-based approach. In [41], Clavier *et al.* further extended the methodology to quadrivariate blind SCA targeting masked AES implementations employing Boolean masking. A common limitation of these works, however, is that their validation was restricted to simulation settings.

Le Bouder *et al.* [43] redefined blind SCA through the lens of probabilistic graphical models and proposed a new analysis framework that uses Belief Propagation (BP) to infer probabilistic dependencies among side-channel leakages. They demonstrated attacks on lightweight AEAD schemes such as Sparkle [46] and Elephant [47]. Nevertheless, their evaluation was also limited to simulation environments.

Ravi *et al.* [42] extended the scope of blind SCA to post-quantum cryptography (PQC) by targeting Kyber [48]. By analyzing leakages stemming from pointwise multiplication operations during decapsulation, they showed that blind SCA can recover coefficients of the secret polynomial. While they performed experimental validation on an ARM Cortex-M4 platform, the approach relies on a trained leakage classifier, which makes the attack not fully blind.

More recently, Rezaeezade *et al.* [38] proposed a deep-learning-based blind SCA framework. They introduced a multi-point cluster labeling technique that simultaneously leverages multiple leakage points, replacing the manual assumption of Hamming weight labels for intermediate values. Moreover, they exploited the empirical observation that, even when training deep neural networks with noisy labels, salient patterns from correctly labeled samples tend to be learned first, thereby improving the success rate of Hamming weight estimation for the target intermediate values. Based on these ideas, they achieved accurate secret key recovery using only real measured side-channel traces across a range of algorithms and platforms, including AES [49], ASCON [50], and Kyber [48].

While modular addition and carry-related leakages have been exploited in conventional SCA settings [51]–[54], such approaches typically assume known inputs. In contrast, our work targets modular addition in a blind SCA setting where plaintext/ciphertext are unavailable.

B. Contributions

Our main contributions are summarized as follows.

- We propose MALEAK, the first systematic blind SCA study that exploits modular-addition leakage. We show that carry propagation induces key-dependent statistical characteristics that can be leveraged without any plaintext or ciphertext information.
- In MALEAK, we define two generalized nonlinear functions that incorporate modular addition and a secret key as analysis targets for blind SCA. By conducting experiments in a simulation environment, we show that secret key recovery is feasible for these analysis targets.
- As a core component of MALEAK, we propose an efficient carry-aware method to generate theoretical joint Hamming weight distributions for n-bit modular addition without enumerating all 2^{2n} input pairs.
- We instantiate MALEAK for HIGHT and SPECK by leveraging their round-function structures. We then validate the resulting attacks on both simulated traces and real power traces collected from an ARM Cortex-M4 platform, confirming practical key recovery using only side-channel information.

C. Organization

The remainder of this paper is organized as follows. Section II reviews the blind SCA framework and summarizes structural characteristics of ARX-based block ciphers. Section III introduces MALEAK by formulating blind SCA targets induced by modular addition and demonstrates the feasibility of key recovery in a simulation setting. Section IV instantiates MALEAK for HIGHT and SPECK, and presents practical attack points and step-by-step key recovery procedures. Section V evaluates MALEAK using both simulated traces and real power measurements, and reports the experimental results. Finally, Section VI concludes the paper.

II. Background

This section provides background on blind SCA and ARX-based block ciphers to facilitate understanding of the paper.

A. Blind Side-Channel Analysis

Blind SCA is a side-channel technique that recovers a secret key by exploiting the fact that, for a key-dependent nonlinear function inside a cryptographic algorithm, the joint distribution of its input and output varies with the value of the secret key. Blind SCA was first proposed by Linge *et al.* in 2014 [39] and differs from traditional techniques such as DPA [1] and CPA [2] in that it can estimate the secret key without using any plaintext or ciphertext information, relying solely on side-channel traces. The basic procedure of Blind SCA is to (i) generate, for each key hypothesis, the theoretical joint distribution of the nonlinear function, (ii) estimate an empirical joint distribution from measured side-channel traces, and (iii) identify the actual key by comparing the two distributions. Later, at CHES 2017, Clavier *et al.* [40] improved the construction of the empirical distribution and

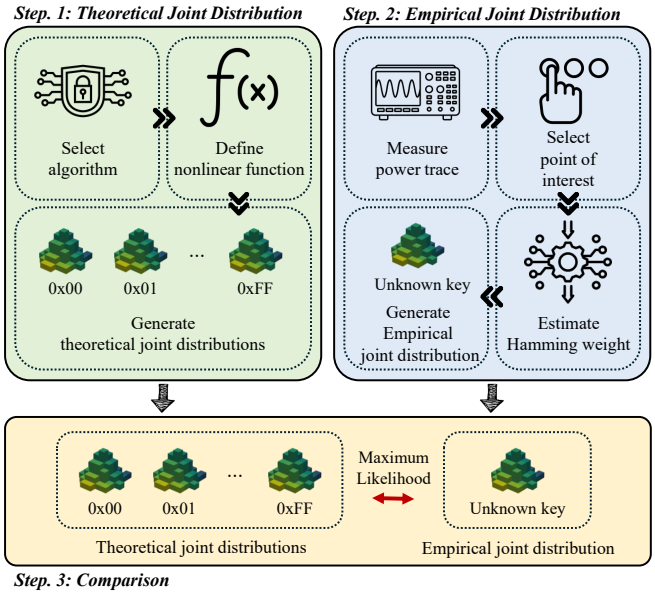

Fig. 1. Overview of the blind SCA framework. For each key hypothesis, a theoretical joint distribution is precomputed and compared with an empirical distribution built from PoI leakages via Hamming weight estimation, and the key is ranked using a Maximum Likelihood criterion.

the distribution-comparison method, thereby enhancing the performance of the original approach.

In this work, we follow the analysis scenario of [40] and perform blind SCA under the same assumptions. Specifically, we assume that the plaintext input to the cipher follows a uniform distribution, and that the side-channel leakage follows a Hamming weight leakage model. In what follows, we describe the overall procedure step by step based on the blind SCA framework proposed by Clavier *et al.* [40]. Figure 1 illustrates the blind SCA framework.

Step 1: Constructing the Theoretical Joint Distribution: The first step constructs, for each key hypothesis, the theoretical joint distribution for a key-dependent nonlinear function in the cryptographic algorithm. Let the input to the nonlinear function be m and the output be y. For example, for AES [49], this nonlinear function can be written as

$$y = S\text{-box}(m \oplus k). \tag{1}$$

Let $HW(x)$ denote the Hamming weight of input x. For each key hypothesis k, the goal is to compute the joint distribution of $(HW(m), HW(y))$. Since the plaintext input is assumed to be uniformly distributed, we iterate over all possible inputs $m \in M$, compute the corresponding output y, and accumulate the occurrence counts of each $(HW(m), HW(y))$ pair. The distribution is then normalized by the size of the input space $|M|$. Because this theoretical joint distribution is independent of the measurement setup and the physical characteristics of the device, it can be precomputed and reused during the attack.

Step 2: Constructing the Empirical Joint Distribution: The second step uses the measured side-channel traces to estimate the Hamming weights of the target intermediate values and construct the empirical joint distribution accordingly. The accuracy of this step strongly depends on PoI selection. Since PoI selection remains an open challenge in fully blind scenarios and is outside the scope of this paper, we follow prior work [38] and adopt the oracle-assisted PoI selection assumption defined in Section III-A.

After the PoI is selected, the leakage samples observed at that point across many traces are used to estimate the Hamming weight of the target intermediate value for each trace. Clavier *et al.* [40] proposed a Variance Analysis method based on linear regression, assuming the following Hamming weight leakage model:

$$P_v = \alpha HW(v) + \beta + N(0, \sigma^2), \tag{2}$$

where α and β are constants, and $N(0, \sigma^2)$ denotes Gaussian noise with standard deviation σ.

Under the assumption of uniformly distributed plaintext inputs, the Hamming weight of an n-bit intermediate value v follows the binomial distribution $B(n, \frac{1}{2})$. Accordingly, the mean and variance of $HW(v)$ are given by $E[HW(v)] = \frac{n}{2}$ and $\mathrm{Var}[HW(v)] = \frac{n}{4}$, respectively. Using these, the variance and expectation of (2) can be written as

$$\mathrm{Var}(P_v) = \alpha^2 \mathrm{Var}(HW(v)) + \mathrm{Var}(N(0, \sigma^2)), \tag{3}$$

$$E(P_v) = \alpha E(HW(v)) + \beta. \tag{4}$$

Solving for α and β yields

$$\alpha = \pm \sqrt{\frac{\mathrm{Var}(P_v) - \mathrm{Var}(N(0, \sigma^2))}{\mathrm{Var}(HW(v))}}, \tag{5}$$

$$\beta = E(P_v) - \alpha E(HW(v)). \tag{6}$$

Clavier *et al.* [40] also presented a method for determining the sign of α. Finally, given an observed leakage value and the estimated α and β, the estimated Hamming weight is computed as

$$\widehat{HW}(v) = \frac{P_v - \beta}{\alpha}. \tag{7}$$

This provides the estimated Hamming weight pair $(\widehat{HW}(m), \widehat{HW}(y))$ for the target intermediate values.

Step 3: Comparison: In the final step, the theoretical joint distribution constructed for each key hypothesis is compared with the empirical joint distribution estimated from side-channel traces. As in [40], we use the Maximum Likelihood criterion. Let the true Hamming weight pair be (h_m^*, h_y^*) and the estimated pair be (h_m, h_y). By Bayes' rule,

$$\Pr(k \mid (h_m, h_y)) = \frac{\Pr((h_m, h_y) \mid k) \Pr(k)}{\Pr((h_m, h_y))}. \tag{8}$$

Since $\Pr((h_m, h_y))$ is independent of the key, it can be omitted, and we assume $\Pr(k)$ is uniform. Therefore, for a sequence of Hamming weight pairs obtained from side-channel traces $\left((h_m, h_y)_i\right)_{i=1,\ldots,N}$, the posterior can be updated sequentially as

$$\Pr\Big(k \,\Big|\, \left((h_m, h_y)_i\right)_{i=1,\ldots,N}\Big) \propto \Pr((h_m, h_y)_N \mid k) \cdot \Pr\Big(k \,\Big|\, \left((h_m, h_y)_i\right)_{i=1,\ldots,N-1}\Big). \tag{9}$$

By the law of total probability, $\Pr((h_m, h_y)_i \mid k)$ can be expressed as

$$\Pr((h_m, h_y)_i \mid k) = \sum_{(h_m^*, h_y^*)} \Pr\big((h_m, h_y)_i \mid (h_m^*, h_y^*)\big) \cdot \Pr\big((h_m^*, h_y^*) \mid k\big), \tag{10}$$

where $\Pr((h_m^*, h_y^*) \mid k)$ is obtained from the theoretical joint distribution constructed in Step 1.

Finally, we model $\Pr((h_m, h_y)_i \mid (h_m^*, h_y^*))$ with a Gaussian distribution:

$$\Pr\big((h_m, h_y)_i \mid (h_m^*, h_y^*)\big) = \frac{1}{\sigma_m\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{h_{m,i}-h_m^*}{\sigma_m}\right)^2} \cdot \frac{1}{\sigma_y\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{h_{y,i}-h_y^*}{\sigma_y}\right)^2}. \tag{11}$$

Clavier *et al.* [40] estimated the noise standard deviations σ_m and σ_y from low-activity regions of the standard deviation trace by selecting representative mid-range values.

By iteratively evaluating these likelihoods, we can compute a posterior probability for each key hypothesis.

B. ARX-based Block Cipher

In resource-constrained environments such as IoT devices, lightweight block ciphers are required to provide confidentiality while maintaining low computational complexity and power consumption [55]. To meet these demands, numerous ARX-based block ciphers have been proposed because they can be implemented efficiently using only simple operations [44], [45], [56], [57].

Unlike substitution–permutation network (SPN) designs, ARX-based block ciphers are constructed solely from modular addition, bit rotations, and XOR operations. As a result, they avoid complex S-boxes or table lookups, enabling simple and efficient implementations in both software and hardware.

Now, we describe two representative ARX-based block ciphers, HIGHT and SPECK.

1) Block Cipher HIGHT: HIGHT [44] is a block cipher designed to provide data confidentiality in environments where low-power and lightweight implementations are required. It takes a 64-bit plaintext and a 128-bit master key as input and outputs a 64-bit ciphertext. HIGHT is built around 8-bit word operations and consists only of modular addition, rotations, and XOR, which makes it amenable to both software and hardware implementations. The notations used in HIGHT are as follows:

- P_j: the j-th plaintext byte
- C_j: the j-th ciphertext byte
- $X_{i,j}$: the j-th byte of the i-th round internal state
- MK_j: the j-th master key byte
- SK_j: the j-th subkey byte
- $\boxplus$: addition modulo 2^8
- $\oplus$: XOR
- $A^{\lll s}$: s-bit left rotation of an 8-bit value A

Subkeys are generated according to Algorithm 1, where δ_i denotes constants defined by an LFSR (Linear Feedback Shift Register).

Algorithm 1 HIGHT subkey generation

Input: $MK_0, \ldots, MK_{15}$
Output: $SK_0, \ldots, SK_{127}$

```
for i = 0 to 7 do
    for j = 0 to 7 do
        SK_{16·i+j} ← MK_{(j−i) mod 8} ⊞ δ_{16·i+j}
    end for
    for j = 0 to 7 do
        SK_{16·i+j+8} ← MK_{((j−i) mod 8)+8} ⊞ δ_{16·i+j+8}
    end for
end for
return SK_0, ..., SK_127
```

The round function is defined in (13), and the encryption procedure consists of 32 rounds in total. Each round uses two auxiliary functions F_0 and F_1 defined in (12). In the last round, the byte-shift operation is not performed.

$$\begin{aligned} F_0(X) &= (X^{\lll 1}) \oplus (X^{\lll 2}) \oplus (X^{\lll 7}), \\ F_1(X) &= (X^{\lll 3}) \oplus (X^{\lll 4}) \oplus (X^{\lll 6}). \end{aligned} \tag{12}$$

$$\begin{aligned} X_{i,j} &= X_{i-1,j-1}, \quad j = 1, 3, 5, 7, \\ X_{i,0} &= X_{i-1,7} \oplus \big(F_0(X_{i-1,6}) \boxplus SK_{4i-1}\big), \\ X_{i,2} &= X_{i-1,1} \boxplus \big(F_1(X_{i-1,0}) \oplus SK_{4i-4}\big), \\ X_{i,4} &= X_{i-1,3} \oplus \big(F_0(X_{i-1,2}) \boxplus SK_{4i-3}\big), \\ X_{i,6} &= X_{i-1,5} \boxplus \big(F_1(X_{i-1,4}) \oplus SK_{4i-2}\big). \end{aligned} \tag{13}$$

2) Block Cipher SPECK: SPECK [45] is a lightweight block cipher proposed by the U.S. National Security Agency (NSA) in 2013 in response to the need for secure and fast ciphers that can operate even in extremely constrained environments such as IoT and embedded systems. SPECK was designed to be highly flexible so that it can be deployed across diverse platforms and applications with different constraints. To this end, it allows the block size and key size to be adjusted to trade off between security and performance. SPECK supports five block sizes $2n \in \{32, 48, 64, 96, 128\}$, and for each block size it uses a key consisting of two to four n-bit words. In this paper, we focus on SPECK32/64. For SPECK32/64, $2n = 32$, and the key consists of four words. The operations used in SPECK32/64 are denoted as follows:

- S^{-7}: right circular rotation by 7 bits
- S^2: left circular rotation by 2 bits
- k_i: the i-th 16-bit round key
- $\boxplus$: addition modulo 2^{16}
- $\oplus$: XOR

The SPECK32/64 encryption updates a state consisting of two 16-bit words (x, y) iteratively. In each round, it uses the round key k_i and performs the operations defined in (14). Depending on the parameter set, the SPECK32/64 encryption consists of 22 rounds in total.

$$R_{k_i}(x, y) = \big((S^{-7}x \boxplus y) \oplus k_i,\ S^2 y \oplus (S^{-7}x \boxplus y) \oplus k_i\big) \tag{14}$$

SPECK's key schedule is designed to generate round keys sequentially by reusing the same structure as the encryption

Algorithm 2 SPECK32/64 round key generation

Input: $K = (l_2, l_1, l_0, k_0)$
Output: $k_0, \ldots, k_{21}$
1: **for** $i = 0$ to 20 **do**
2: $\quad l_{i+3} = (k_i \boxplus S^{-7} l_i) \oplus i$
3: $\quad k_{i+1} = S^2 k_i \oplus l_{i+3}$
4: **end for**
5: **return** $k_0, \ldots, k_{21}$

round function. Given the SPECK32/64 master key $K = (l_2, l_1, l_0, k_0)$, the round keys are generated according to Algorithm 2.

III. Blind Side-Channel Analysis on Modular Addition

In this section, we systematically analyze the feasibility of blind SCA by focusing on modular addition, a core operation widely used in ARX-based block ciphers. We refer to this modular addition-targeted blind-SCA framework as MALEAK. To this end, we first formalize a general form of key-dependent nonlinear functions that simultaneously include a secret key and modular addition operations. We then conduct simulation-based experiments on these functions to examine whether blind SCA can effectively operate even when modular addition is selected as the analysis target.

A. Threat Model

Based on previously proposed blind SCA scenarios, we define an adversary model that is applicable to ARX-based block ciphers. We assume that the adversary can physically access the target device and measure a large number of side-channel traces. However, the adversary is assumed to have no information about the plaintext or ciphertext corresponding to each measurement. The adversary's goal is to recover the secret key using only side-channel traces by estimating the Hamming weights of intermediate values inside the cryptographic algorithm.

Following the blind SCA setting considered in a recent USENIX Security 2025 study [38], we assume oracle-assisted PoI selection. Concretely, for each target intermediate value v, the oracle returns the sample index t^* that maximizes the Pearson correlation between the leakage sample $P[t]$ and $HW(v)$ across traces. This assumption is introduced to decouple the feasibility of modular addition-based blind key recovery from the largely open PoI-selection problem in fully blind settings.

In our experiments, we emulate this oracle using true Hamming weights computed from the implementation under the fixed secret key used for trace acquisition. Importantly, true information is used only to fix PoIs. After PoIs are fixed, the subsequent key recovery procedure uses only the leakage samples at those PoIs and does not use plaintext or ciphertext information. Developing a practical PoI-selection method that does not rely on such true labels remains an important direction for future work.

B. Identifying Generalized Analysis Targets

To perform MALEAK, it is necessary to explicitly define a key-dependent nonlinear function inside the target cryptographic algorithm. The modular addition operation considered in this work serves as a primary nonlinear component in ARX-based block ciphers. Accordingly, we first define the simplest generalized key-dependent nonlinear functions involving modular addition. We then set these functions as analysis targets and perform blind SCA in a simulation environment to evaluate the attack feasibility on modular addition. This motivates the instantiation of these generalized functions for concrete ARX-based block ciphers in Section IV.

Equation (15) presents two generalized nonlinear functions that include both a secret key and modular addition. Unlike prior blind SCA targets that typically take a single input, our targets take two operands a and b. This two-input structure affects the theoretical distribution, PoI selection, and the likelihood computation.

$$\begin{aligned} g_{\text{general}}(a, b, k) &= (a \boxplus k) \oplus b, \\ h_{\text{general}}(a, b, k) &= (a \oplus k) \boxplus b \end{aligned} \tag{15}$$

C. Efficient Theoretical Joint Distribution Generation

In blind SCA, we generate the theoretical joint distribution for each candidate key k and use it both to guide the analysis design and to validate the experimental results. In general, for an n-bit modular addition, exhaustive enumeration over inputs $(a, b) \in \{0,1\}^n \times \{0,1\}^n$ requires considering 2^{2n} input pairs per key candidate, and the computational cost grows rapidly, especially when $n \geq 16$.

To overcome this bottleneck, MALEAK uses a carry-aware method that exploits the locality of carry propagation in modular addition. Since the computation in lower bits affects higher bits only through a carry, our approach splits the n-bit operation into w-bit blocks and treats the carry generated by a lower block as a conditioning variable for the upper block. Concretely, we first aggregate block-level partial distributions conditioned on the carry state for the nonlinear function, and then compose only these aggregated results to obtain the full joint distribution D.

As a concrete example, consider h_{general} with $n = 16$ and $w = 8$, where we decompose the inputs into the lower byte (a_L, b_L) and the upper byte (a_H, b_H). Algorithm 3 presents the proposed method as follows. It iterates over $(a_L, b_L) \in [0, 255]^2$, computes $t_L = (a_L \oplus key_L) + b_L$, and extracts the lower-byte output $o_L = t_L \bmod 2^8$ together with the carry to the upper byte $c = \lfloor t_L / 2^8 \rfloor \in \{0, 1\}$. Instead of storing individual input pairs, it aggregates $h^{(L)}_{\text{general}} = (HW(a_L), HW(b_L), HW(o_L))$ into the lower-block frequency table L_c conditioned on c. For the upper 8-bit block, it conditions on the incoming carry $c \in \{0, 1\}$, computes $o_H = ((a_H \oplus key_H) + b_H + c) \bmod 2^8$, and aggregates $h^{(H)}_{\text{general}} = (HW(a_H), HW(b_H), HW(o_H))$ into the upper-block frequency table U_c for each carry condition. Finally, it composes L_c and U_c under the same carry condition c and accumulates the full distribution over (h_a, h_b, h_o) using $h_a = HW(a_L) + HW(a_H)$, $h_b = HW(b_L) + HW(b_H)$,

Algorithm 3 Generation of the 16-bit theoretical joint distribution for h_{general}

Input: key (16-bit key candidate)
Output: $D(\cdot,\cdot,\cdot)$

$key_L \leftarrow key \mod 2^8$
$key_H \leftarrow (key \gg 8) \mod 2^8$
Initialize $L_0(\cdot,\cdot,\cdot) \leftarrow 0, \quad L_1(\cdot,\cdot,\cdot) \leftarrow 0$
Initialize $U_0(\cdot,\cdot,\cdot) \leftarrow 0, \quad U_1(\cdot,\cdot,\cdot) \leftarrow 0$
Initialize $D(\cdot,\cdot,\cdot) \leftarrow 0$
for $a_L = 0$ to 255 **do**
  for $b_L = 0$ to 255 **do**
    $t_L \leftarrow (a_L \oplus key_L) + b_L$
    $o_L \leftarrow t_L \mod 2^8$
    $c \leftarrow t_L \gg 8$
    $L_c(HW(a_L), HW(b_L), HW(o_L)) \leftarrow L_c(HW(a_L), HW(b_L), HW(o_L)) + 1$
  end for
end for
for $a_H = 0$ to 255 **do**
  for $b_H = 0$ to 255 **do**
    for $c \in \{0,1\}$ **do**
      $o_H \leftarrow ((a_H \oplus key_H) + b_H + c) \mod 2^8$
      $U_c(HW(a_H), HW(b_H), HW(o_H)) \leftarrow U_c(HW(a_H), HW(b_H), HW(o_H)) + 1$
    end for
  end for
end for
for $c \in \{0,1\}$ **do**
  for each entry $((h_{aL}, h_{bL}, h_{oL}), m_L)$ in L_c **do**
    for each entry $((h_{aH}, h_{bH}, h_{oH}), m_H)$ in U_c **do**
      $D(h_{aL}+h_{aH},\, h_{bL}+h_{bH},\, h_{oL}+h_{oH}) \leftarrow D(h_{aL}+h_{aH},\, h_{bL}+h_{bH},\, h_{oL}+h_{oH}) + \dfrac{m_L \cdot m_H}{2^{32}}$
    end for
  end for
end for
return D

and $h_o = HW(o_L) + HW(o_H)$. This produces the 16-bit theoretical joint distribution with respect to h_{general}. Assuming all input pairs occur uniformly, we use 2^{32} as the normalization factor during accumulation.

Our procedure naturally generalizes to $n > 16$, and the same holds for g_{general}. In particular, as long as carry propagation across blocks is handled correctly, we can decompose the computation into blocks of any size w, propagate carry conditions between adjacent blocks, and efficiently construct the h_{general}-based theoretical joint distribution for an n-bit modular addition.

Complexity: A naive exhaustive approach requires $O(2^{2n})$ operations per key candidate. For $n = 16$, this corresponds to processing $2^{32} \approx 4.29 \times 10^9$ input pairs. In contrast, the proposed method in Algorithm 3 requires 2^{16} operations to aggregate the lower block and $2 \cdot 2^{16}$ operations to aggregate the upper block under two carry conditions. The final composition stage is bounded by the Hamming weight range $0, \ldots, 8$, and even in the worst case it involves at most $2{\cdot}9^3{\cdot}9^3 = 2{\cdot}9^6 \approx 1.06{\times}10^6$ accumulation operations. Overall, the distribution can be constructed with roughly 1.26×10^6 operations per key candidate, yielding about a 3.4×10^3-fold reduction in computation compared to exhaustive enumeration for the setting $n = 16$ and $w = 8$.

D. Evaluation Metric

To quantitatively evaluate the performance of MALEAK, we use the guessing entropy (GE). For a given set of measured side-channel traces, we score and rank all key hypotheses, and define GE as the expected rank of the correct key. Hence, GE provides an indirect measure of how many traces an adversary must consider on average to identify the true key.

In this paper, we estimate GE by repeating the key recovery experiment over multiple independent trials and averaging the rank of the correct key across trials. A smaller GE indicates that the correct key is assigned a smaller rank (i.e., ranked closer to the top) among candidates, and ideally a GE close to 1 reflects strong attack performance.

E. Validation through Simulation Experiment

We validate MALEAK through simulation experiments. All experiments use simulated side-channel traces, and we assume that power leakage follows the Hamming weight leakage model in (16). For all sampling points v, the constants α and β are fixed to 1 and 0, respectively. We evaluate three noise conditions with $\sigma = 0.7, 1.0, 1.5$, as in [40].

$$P_v = \alpha HW(v) + \beta + N(0, \sigma^2) \tag{16}$$

For each modular addition setting ($n \in \{8, 16\}$), we conducted 1,000 independent trials. In each trial, we sampled the secret key uniformly at random, generated 100,000 simulated traces, and performed key recovery. We recorded the rank of the true key among all candidates and estimated the GE as the average rank over all trials. Figure 2 reports the GE curves for g_{general} and h_{general} in both the 8-bit and 16-bit modular addition settings.

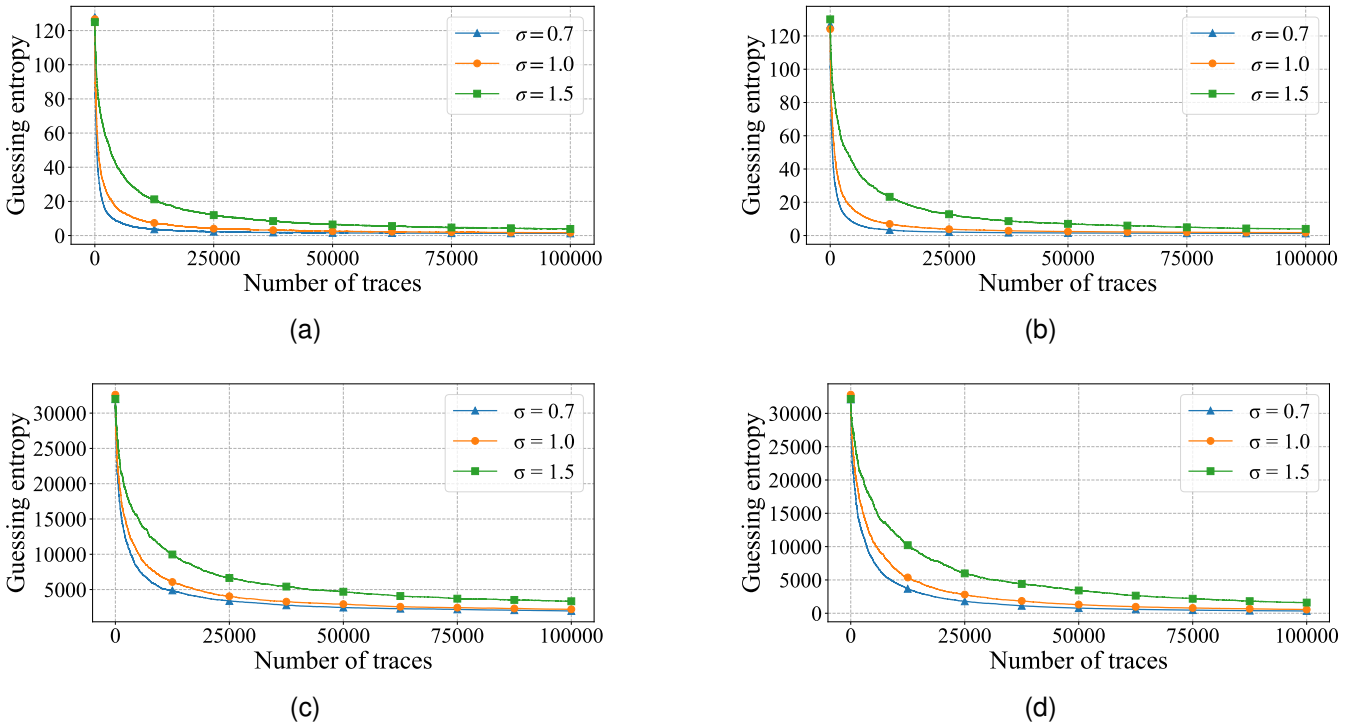

Fig. 2. GE as a function of the number of simulated traces for the generalized targets g_{general} and h_{general} under 8-bit and 16-bit modular addition. Traces are generated under the Hamming weight leakage model with additive Gaussian noise for $\sigma \in \{0.7, 1.0, 1.5\}$. GE decreases as more traces are accumulated, while larger σ slows the convergence. (a) 8-bit g_{general}. (b) 8-bit h_{general}. (c) 16-bit g_{general}. (d) 16-bit h_{general}.

Figures 2a and 2b report the results for the g_{general} and h_{general} functions, respectively, under the 8-bit modular addition setting, illustrating how the GE of the correct key evolves as the number of simulated traces increases. For both functions, GE decreases gradually with more traces, indicating that the rank of the correct key decreases toward 1 as more leakage observations are accumulated. As summarized in Table I, in the 8-bit setting the GE values of g_{general} are 1.23, 1.66, and 3.88 for $\sigma = 0.7, 1.0, 1.5$, respectively, whereas those of h_{general} are 1.26, 1.77, and 4.01 under the same noise conditions. Overall, GE increases as the noise level grows, while remaining close to 2 or below for $\sigma \leq 1.0$, which implies that the correct key is consistently ranked among the top candidates in the mid-noise conditions.

Figures 2c and 2d present the corresponding results under the 16-bit modular addition setting. Similar to the 8-bit case, GE decreases as the number of traces increases, demonstrating that additional traces improve the key-identification capability. Table I shows that for $\sigma = 0.7, 1.0, 1.5$, the GE values of g_{general} are 1968.67, 2208.29, and 3343.51, respectively, while those of h_{general} are 343.95, 559.12, and 1602.34. Although the 16-bit setting yields substantially larger GE values due to the expanded key space, the same overall trend is observed, and GE increases with the noise level.

In summary, across both the 8-bit and 16-bit modular addition settings, GE consistently decreases with the number of traces, confirming that the proposed MALEAK progressively moves the correct key closer to the top of the ranking as more leakage information becomes available. A quantitative comparison of the overall results is provided in Table I.

TABLE I
GE FOR THE g_{general} AND h_{general} IN A SIMULATION SETTING

	8-bit		16-bit	
	g_{general}	h_{general}	g_{general}	h_{general}
σ = 0.7	1.23	1.26	1968.67	343.95
σ = 1.0	1.66	1.77	2208.29	559.12
σ = 1.5	3.88	4.01	3343.51	1602.34

IV. APPLYING MALEAK TO HIGHT AND SPECK

In Section III, we introduced MALEAK and showed that modular addition nonlinearity can be exploited in blind SCA under various noise levels. In this section, we instantiate MALEAK for ARX-based block ciphers, focusing on HIGHT and SPECK.

A. Blind Side-Channel Analysis on HIGHT

1) Application of the Analysis Target to HIGHT: By comparing the HIGHT round function in (13) with the generalized key-dependent nonlinear forms in (15), we identify round-level relations whose modular addition structure matches the two generalized functions. In particular, the generalized forms can be instantiated in HIGHT by incorporating the auxiliary functions F_0 and F_1 in (12) into the input term. Accordingly, we define the following two nonlinear functions, g_{HIGHT} and h_{HIGHT}, as analysis targets for blind SCA on HIGHT:

$$g_{\mathrm{HIGHT}}(a,b,k) = (F_0(a) \boxplus k) \oplus b,$$
$$h_{\mathrm{HIGHT}}(a,b,k) = (F_1(a) \oplus k) \boxplus b, \tag{17}$$

where a and b are byte-wise values selected from the round internal state, and k denotes a HIGHT subkey generated by Algorithm 1. Since HIGHT operates on 8-bit words, all operations in (17) are defined over 8-bit values.

2) Analysis Scenario: We now describe how MALEAK is instantiated for HIGHT by targeting the nonlinear functions g_{HIGHT} and h_{HIGHT} defined within the HIGHT encryption process. The overall procedure follows the blind SCA framework of Clavier *et al.* [40]. For ease of exposition, we describe the scenario with respect to g_{HIGHT}.

Step 1: Generating the Theoretical Joint Distribution: To perform blind SCA on HIGHT, we first generate, for each subkey hypothesis k, the theoretical joint distribution associated with g_{HIGHT}. A key difference from conventional blind SCA settings is that the target function takes two uniformly distributed inputs a and b. For instance, AES typically assumes a single intermediate input, whereas HIGHT requires two inputs, a and b, to be considered simultaneously. Therefore, for each hypothesis k, our goal is to compute the joint distribution of $(HW(a), HW(b), HW(g_{\mathrm{HIGHT}}))$. We evaluate $g_{\mathrm{HIGHT}}(a,b,k)$ for all $a \in \mathbb{Z}_{2^8}$ and $b \in \mathbb{Z}_{2^8}$, accumulate the occurrence counts of each Hamming weight tuple, and normalize by the full input-space size 2^{16} to obtain the theoretical joint distribution.

Step 2: Generating the Empirical Joint Distribution: Under our adversary assumptions, the adversary uses the PoI-identification oracle to select, for each target intermediate value, the sample point that maximizes the Pearson correlation between the leakage and the target value's Hamming weight. Using the leakage samples observed at each PoI, the adversary estimates the Hamming weights of the intermediate values. Since we adopt the Variance Analysis method of Clavier *et al.* [40], we determine the constants α and β in the Hamming weight leakage model in (18).

$$P_v = \alpha HW(v) + \beta + N(0,\sigma^2) \tag{18}$$

In HIGHT, the intermediate value v considered here is 8-bit, and thus $HW(v)$ follows the binomial distribution $B(8,\frac{1}{2})$, which yields $E[HW(v)] = 4$ and $\mathrm{Var}[HW(v)] = 2$. Accordingly, we use (19) and (20) to estimate Hamming weights for HIGHT, where $\mathrm{Var}(N(0,\sigma^2))$ is determined by estimating the noise variance from the side-channel traces.

$$\alpha = \pm\sqrt{\frac{\mathrm{Var}(P_v) - \mathrm{Var}(N(0,\sigma^2))}{2}} \tag{19}$$

$$\beta = E(P_v) - 4\alpha \tag{20}$$

Following [40], we regard the standard deviation level in an input-independent quiet segment as the measurement noise level. While [40] selects this level heuristically, we estimate σ automatically as the most frequent value in the standard deviation trace computed from the collected side-channel traces. Here, the standard deviation trace denotes the time sample-wise standard deviation computed across an ensemble of aligned traces. The same noise estimation procedure is applied in our SPECK experiments as well.

In (19), the sign of α indicates whether the observed leakage increases or decreases with $HW(v)$. In our experimental setup, we use a single-ended acquisition with `MEASURE_POS` connected to the shunt-low sensing node. With this wiring, higher instantaneous current causes a larger voltage drop across the shunt resistor and thus a lower measured voltage at the sensing node, leading to a negative correlation between the measured sample P_v and $HW(v)$. Therefore, we fix the sign in (19) to the negative one (i.e., we take $\alpha < 0$). If no information is available to determine the polarity, one may determine the sign using the procedure described in [40]. The same sign-determination procedure is applied in our SPECK experiments as well.

Step 3: Comparing the Theoretical Joint Distributions with the Empirical One: Finally, we identify the correct subkey by comparing, for each subkey hypothesis, the theoretical joint distribution with the empirical joint distribution constructed from the side-channel traces. We apply the Maximum Likelihood criterion as in [40] and extend the formulation to account for the three intermediate values $(a, b, g_{\mathrm{HIGHT}})$. Concretely, we iteratively evaluate the likelihood of the estimated Hamming weight tuples under each key hypothesis.

3) GE-Based Success Criterion: In HIGHT, Algorithm 1 expands each 8-bit master key byte MK_t into eight distinct 8-bit subkeys. For a fixed target index t, the key schedule (together with the LFSR-defined constants δ_i) induces, for each $j \in \{0,\dots,7\}$, a deterministic mapping from a master key candidate $m \in \{0,\dots,255\}$ to the corresponding j-th subkey byte, which we denote by $s_j(m;\delta_i)$. We use this mapping to express all subkey posteriors over a common master key candidate index set and then fuse them.

Let $\mathbf{p}_j^{\mathrm{SK}} \in \mathbb{R}^{256}$ be the posterior vector for the j-th subkey, where the entry $\mathbf{p}_j^{\mathrm{SK}}[s]$ corresponds to the subkey candidate $s \in \{0,\dots,255\}$. We define a re-indexing operator $\mathcal{R}_j : \mathbb{R}^{256} \to \mathbb{R}^{256}$ by

$$\big(\mathcal{R}_j(\mathbf{p})\big)[m] = \mathbf{p}[s_j(m;\delta_i)]\,. \tag{21}$$

Applying $\mathcal{R}_j$ yields an aligned posterior over master key candidates,

$$\mathbf{p}_j^{\mathrm{MK}} = \mathcal{R}_j\big(\mathbf{p}_j^{\mathrm{SK}}\big)\,, \tag{22}$$

so that the index m refers to the same MK_t hypothesis for all j.

To combine information from the eight subkeys, we compute a fusion score by element-wise multiplication:

$$S^{\mathrm{MK}}[m] = \prod_{j=0}^{7} \mathbf{p}_j^{\mathrm{MK}}[m]. \tag{23}$$

We sort the 256 candidates in descending order of $S^{\mathrm{MK}}[m]$ and define the rank of the correct master key byte accordingly. Over multiple independent trials, we estimate the GE as the average rank of the correct candidate. Following prior work [38], we declare recovery successful when $\mathrm{GE} \le 10$, indicating that the correct key byte is concentrated among the top-ranked candidates.

B. Blind Side-Channel Analysis on SPECK

1) Application of Analysis Target to SPECK: Among the SPECK family with various parameter sets, we derive the analysis target for SPECK32/64, which has a 32-bit block size and a 64-bit key size. To this end, we decompose the round function in (14) into step-by-step operations. Since SPECK32/64 adopts a 16-bit word size, all operations in the round function are performed over 16-bit values. Let the state at the i-th round be (x_i, y_i). Then, the operations over two consecutive rounds can be expressed as follows, where x'_i denotes an intermediate value generated during the computation of x_{i+1}:

1) $x'_i = S^{-7}x_i \boxplus y_i$
2) $x_{i+1} = x'_i \oplus k_i$
3) $y_{i+1} = S^2 y_i \oplus x_{i+1}$
4) $x'_{i+1} = S^{-7}x_{i+1} \boxplus y_{i+1}$
5) $x_{i+2} = x'_{i+1} \oplus k_{i+1}$
6) $y_{i+2} = S^2 y_{i+1} \oplus x_{i+2}$

Among these steps, we focus on the intermediate results computed in Steps 2 and 4. Specifically, after applying an XOR between x'_i and k_i, a 7-bit right circular rotation is applied to the result, and then a modular addition is performed between the rotated value and y_{i+1}. This computation flow can be interpreted as having the same structural form as the generalized nonlinear function h_{general} in (15), except for the right circular rotation. Accordingly, we define the nonlinear function h_{SPECK} in (24) as the blind SCA analysis target for SPECK32/64.

$$h_{\text{SPECK}}(a, b, k) = (S^{-7}(a \oplus k)) \boxplus b. \tag{24}$$

Here, a and b are intermediate values selected within the SPECK32/64 round function, and k denotes a SPECK32/64 round key generated by Algorithm 2.

2) Analysis Scenario: We next describe a MALEAK instantiation for SPECK32/64 targeting the nonlinear function h_{SPECK} defined above for SPECK32/64. The overall procedure follows the framework of [40], as in the HIGHT case, while adjusting only the construction of the theoretical and empirical joint distributions to match the 16-bit word size and the form of the analysis target.

Step 1: Generating the Theoretical Joint Distribution: To perform blind SCA on SPECK32/64, we first generate, for each round key hypothesis k, the theoretical joint distribution of h_{SPECK}. As in HIGHT, the underlying nonlinear function takes two uniformly distributed intermediate inputs. Hence, the distribution can be defined by evaluating $h_{\text{SPECK}}(a, b, k)$ over all $a \in \mathbb{Z}_{2^{16}}$ and $b \in \mathbb{Z}_{2^{16}}$, accumulating the occurrence counts of each Hamming weight tuple, and normalizing by the full input-space size 2^{32}. In practice, however, we do not explicitly enumerate all 2^{32} input pairs. Instead, we generate the $n = 16$ theoretical joint distribution efficiently using Algorithm 3, and use the resulting normalized distribution for subsequent analysis.

Step 2: Generating the Empirical Joint Distribution: We again assume that the adversary uses the PoI-identification oracle to select PoIs. Using the leakage at the selected PoIs, the adversary estimates Hamming weights via Variance Analysis. To this end, we determine the constants α and β in the leakage model in (25):

$$P_v = \alpha HW(v) + \beta + N(0, \sigma^2). \tag{25}$$

In SPECK32/64, the intermediate value v considered here is 16-bit, and thus $HW(v)$ follows $B(16, \frac{1}{2})$, yielding $E[HW(v)] = 8$ and $\text{Var}[HW(v)] = 4$. Accordingly, we apply variance analysis using (26) and (27) to estimate Hamming weights:

$$\alpha = \pm\sqrt{\frac{\text{Var}(P_v) - \text{Var}(N(0, \sigma^2))}{4}}, \tag{26}$$

$$\beta = E(P_v) - 8\alpha. \tag{27}$$

Step 3: Comparing the Theoretical and Empirical Joint Distributions: Finally, we compare the theoretical and empirical joint distributions using the Maximum Likelihood criterion, as in the HIGHT case. Since SPECK32/64 also involves three intermediate values (a, b, h_{SPECK}), we use the corresponding extended formulation to iteratively compute the probability of each candidate key.

3) GE-Based Success Criterion: We define the success of our SPECK32/64 key-recovery attack using GE. As shown in Algorithm 2, the key schedule expands the 64-bit master key $K = (l_2, l_1, l_0, k_0)$ into 22 16-bit round keys $k_0, \ldots, k_{21}$. Since K can be uniquely reconstructed from (k_0, k_1, k_2, k_3) by inverting the schedule, we focus on recovering these four 16-bit round keys.

For each $i \in \{0, 1, 2, 3\}$, let $P_i^{\text{RK}}(r)$ denote the posterior probability assigned to the candidate $r \in \{0, \ldots, 2^{16} - 1\}$ for the round key k_i. After sorting the candidates in descending order of $P_i^{\text{RK}}(r)$, we define $\text{rank}(k_i)$ as the position of the true key k_i in the sorted list. Over multiple independent trials, we estimate the GE for k_i as the average of $\text{rank}(k_i)$.

To set a threshold comparable to the one used for HIGHT, we match the overall remaining brute-force complexity. For HIGHT, the heuristic criterion GE $\leq$ 10 corresponds to retaining at most the top 10 candidates for each of the 16 master key bytes, yielding at most 10^{16} byte combinations. For SPECK32/64, retaining the top B candidates for each of k_0, k_1, k_2, k_3 results in B^4 combinations. Solving $B^4 = 10^{16}$ yields $B = 10^4 = 10{,}000$.

Accordingly, we declare the recovery of a 16-bit round key k_i successful if GE $\leq$ 10,000. We regard the overall attack as successful when this condition holds for all $i \in \{0, 1, 2, 3\}$, since the inversion of Algorithm 2 then reduces the remaining master key search space to at most 10^{16} candidates.

V. Performance Evaluation

In this section, we experimentally evaluate MALEAK on HIGHT and SPECK. The evaluation uses both simulated traces and real power traces measured from an STM32F415 (ARM Cortex-M4).

A. Experimental Setup

1) Simulation Environment: To validate the key recovery performance of MALEAK in a simulation setting, we generated simulated traces under the Hamming-weight leakage model in (28) and evaluated key recovery performance. For all sampling points v, the constants α and β are fixed to 1 and 0, respectively, and $N(0, \sigma^2)$ denotes Gaussian noise. Here, σ denotes the standard deviation of the Gaussian noise. Following the noise setting procedure in Section III-E, we consider three noise conditions with $\sigma \in \{0.7, 1.0, 1.5\}$.

$$P_v = \alpha HW(v) + \beta + N(0, \sigma^2) \tag{28}$$

2) Real-World Environment: To verify the feasibility of key recovery of MALEAK in a real-world setting, we collected power traces using a ChipWhisperer-Husky and an STM32F415 microcontroller (ARM Cortex-M4) target board using a ChipWhisperer CW308 setup [58]. We configured the target clock to run at approximately $7.37\,\mathrm{MHz}$ and sampled the power consumption at approximately $29.48\,\mathrm{MS/s}$. To reduce high-frequency noise, we used a hardware low-pass filter with a $20\,\mathrm{MHz}$ cutoff frequency. For both HIGHT and SPECK32/64, we acquired 200,000 power traces under a fixed secret key. Each HIGHT trace contains 12,700 sample points, whereas each SPECK32/64 trace contains 7,144 sample points.

To emulate the oracle-assisted PoI selection defined in Section III-A, we compute the true $HW(v)$ for each target intermediate value v using the known implementation and the fixed secret key used during measurements, and select the sample index that maximizes the Pearson correlation between $HW(v)$ and the measured leakage $P[t]$. The true Hamming weights are used only for PoI identification. After PoIs are fixed, key recovery uses only the leakage samples at those PoIs and does not use plaintext or ciphertext information.

For HIGHT, we used the C reference implementation provided by the Korea Internet & Security Agency (KISA). For SPECK, we used the C reference implementation presented in [45].

TABLE II
GE FOR THE g_{HIGHT} AND h_{HIGHT} IN A SIMULATION SETTING

	g_{HIGHT}	h_{HIGHT}
$\sigma = 0.7$	1.25	1.21
$\sigma = 1.0$	7.38	7.47
$\sigma = 1.5$	52.83	48.81

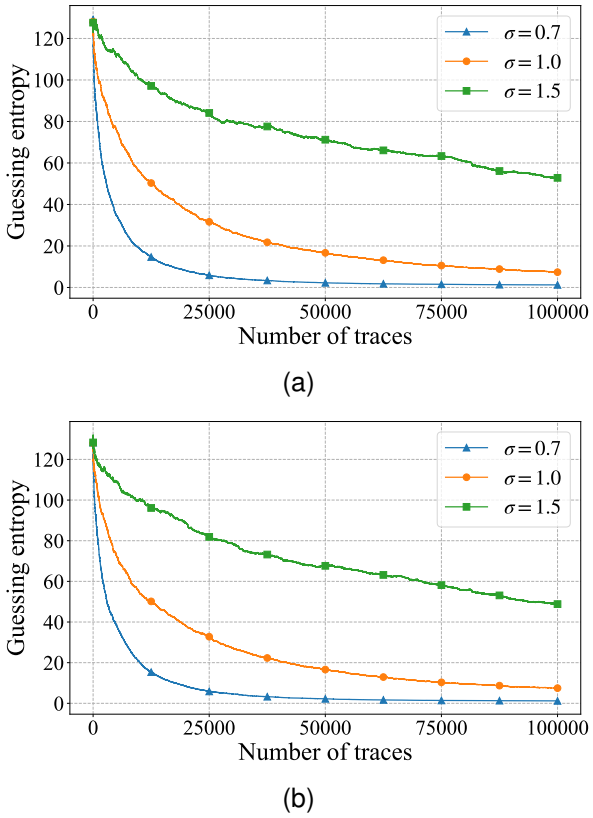

Fig. 3. GE versus the number of simulated traces for the HIGHT analysis targets g_{HIGHT} and h_{HIGHT} under the Hamming weight leakage model with additive Gaussian noise $\sigma \in \{0.7, 1.0, 1.5\}$. GE decreases with additional traces, whereas higher noise requires substantially more traces to reach the same rank. (a) g_{HIGHT}. (b) h_{HIGHT}.

B. Results on HIGHT

We present blind SCA experiments on HIGHT. In the simulation setting, we evaluate the recovery performance of the 8-bit subkeys involved in the round function, while in the real-world setting, we further reconstruct the corresponding master key byte from the recovered subkeys in addition to subkey recovery. We report the results for both settings focusing on the HIGHT analysis targets proposed in Section IV.

1) Evaluation on Simulation Power Traces: To evaluate the subkey recovery performance of blind SCA on HIGHT in the simulation setting, we performed 1,000 independent trials and computed the rank of the correct key for each trial. Each trial was performed with a randomly selected subkey. For uniformly distributed inputs, we generated 100,000 simulated power traces per trial and executed the subkey recovery procedure.

Table II summarizes the subkey recovery results obtained using simulated power traces. For $\sigma = 0.7$, both g_{HIGHT} and h_{HIGHT} achieve GE close to 1 (1.25 and 1.21, respectively), indicating that the correct subkey is almost always ranked first among the 256 candidates. For $\sigma = 1.0$, GE remains in the single digit range with GE values of 7.38 and 7.47. In contrast, under the higher noise condition $\sigma = 1.5$, GE increases to 52.83 for g_{HIGHT} and 48.81 for h_{HIGHT}, reflecting a substantially more challenging recovery regime.

Figures 3a and 3b show, for g_{HIGHT} and h_{HIGHT}, respectively, how GE changes with the number of simulated power traces under different noise levels. For both analysis targets, GE decreases monotonically as the number of traces increases, indicating that the correct key attains a smaller rank as more measurements are collected. Notably, for $\sigma = 1.5$, GE remains in the tens even at 100,000 traces. However, since the curves continue to decrease, the results suggest that additional traces can further improve the recovery performance in this high-noise setting.

2) Evaluation on Real Power Traces: In addition to the simulation setting, we conducted key recovery experiments on

TABLE III
GE FOR THE g_{HIGHT} AND h_{HIGHT} IN A REAL-WORLD SETTING

g_{HIGHT}		h_{HIGHT}	
SK_{21}	2.52	SK_4	3.62
SK_{55}	54.16	SK_{38}	1.33
SK_{81}	15.02	SK_{64}	1.72
SK_{115}	2.19	SK_{98}	12.35

HIGHT using real power consumption traces measured from the target device. In particular, we focus on recovering the master key byte MK_4 and the eight 8-bit subkeys expanded from MK_4.

According to Algorithm 1, a single 8-bit master key byte MK_4 is expanded into eight distinct 8-bit subkeys, four of which are used in round operations corresponding to g_{HIGHT}, and the remaining four are used in operations corresponding to h_{HIGHT}. Therefore, in this experiment, we target the eight subkeys derived from MK_4. Recovering these subkeys allows us to reconstruct MK_4 and validate master key recovery on real traces.

To evaluate key recovery feasibility on real traces, we performed 1,000 independent trials targeting the eight 8-bit subkeys derived from MK_4 and computed GE. All trials were conducted using a fixed master key. In each trial, we randomly selected 100,000 traces from the collected set of 200,000 power traces and performed subkey recovery. Table III reports the subkey recovery results. Among the eight subkeys derived from MK_4, five subkeys achieve an average rank of at most 10 when using 100,000 real power traces.

Figures 4a and 4b show, for the four subkeys used in g_{HIGHT} and the four subkeys used in h_{HIGHT}, respectively, how GE changes as a function of the number of real power traces. As in the simulation setting, increasing the number of traces tends to move the correct key toward rank 1, indicating that key recovery performance can further improve with more traces.

Finally, to assess master key recovery on real power traces, we aggregate the recovery results of the eight 8-bit subkeys derived from MK_4 and compute the GE of the master key byte MK_4. Following the same experimental procedure as in the subkey recovery evaluation, we performed 1,000 independent trials with a fixed master key. In each trial, we randomly selected 100,000 traces from the collected set of 200,000 real power traces and computed the GE of MK_4. The ranking of MK_4 candidates is obtained by applying the procedure described in Section IV-A3, where the candidate probabilities from each of the eight subkey recoveries are combined to produce a final ranking over the 256 candidates. The resulting GE evolution is shown in Figure 5. In particular, when all eight subkey results are jointly fused, the GE of MK_4 reaches 1 after approximately 35,000 traces. This outcome also clearly satisfies our success criterion, i.e., $GE \leq 10$, and demonstrates that, even without any plaintext or ciphertext information, the proposed method can reliably identify the HIGHT master key byte MK_4 given a sufficient number of real power traces.

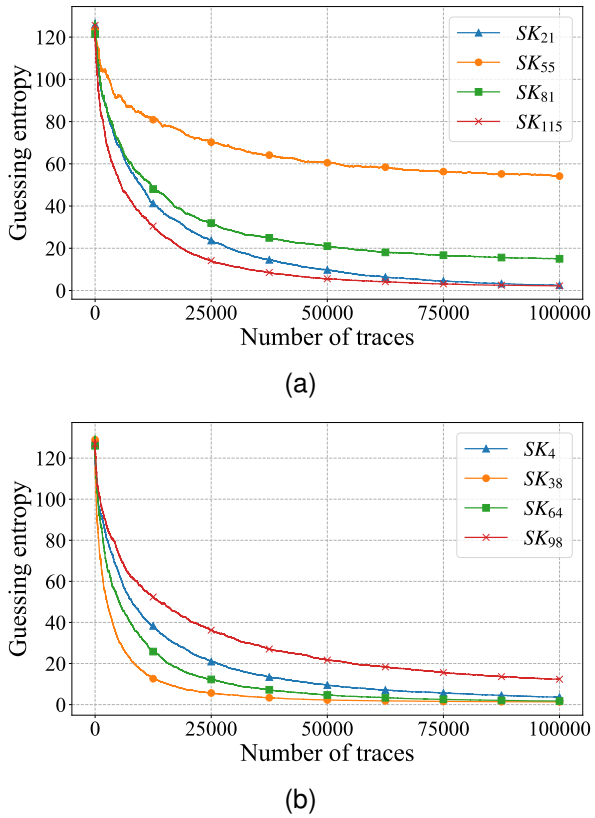

Fig. 4. GE for the eight HIGHT subkeys derived from the master key byte $MK4$ using real power traces, shown as a function of the number of traces. Each curve is obtained by repeatedly selecting a random subset of the measurement set and averaging the resulting rank of the correct subkey. (a) Subkeys recovered via g_{HIGHT}. (b) Subkeys recovered via h_{HIGHT}.

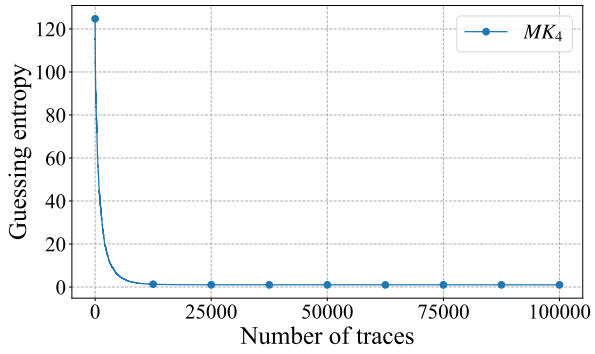

Fig. 5. GE for recovering the HIGHT master key byte $MK4$ from real power traces after fusing the posteriors of the eight corresponding subkey recoveries. Posterior fusion concentrates probability mass on the correct candidate and typically reaches rank 1 with fewer traces than individual subkey attacks.

C. Results on SPECK

Our experiments on SPECK32/64 aim to recover the 16-bit round key used in the round function. In what follows, we present key recovery results obtained using both simulated power traces and real power consumption traces, targeting the SPECK32/64 analysis target defined in Section IV.

1) Evaluation on Simulation Power Traces: To examine the feasibility of blind SCA-based key recovery for SPECK32/64 in a simulation setting, we conducted 1,000 independent ex-

TABLE IV
GE FOR THE h_{SPECK} IN A SIMULATION SETTING

	h_{SPECK}
$\sigma = 0.7$	1669.49
$\sigma = 1.0$	1860.46
$\sigma = 1.5$	2746.24

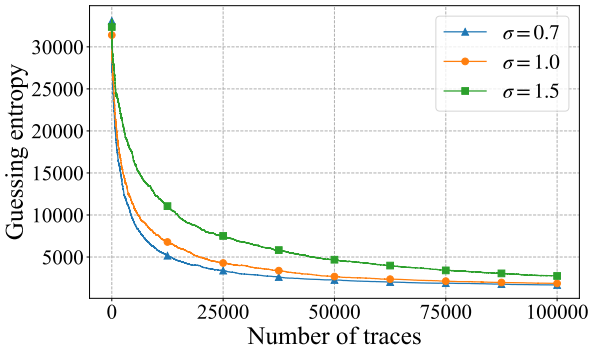

Fig. 6. GE versus the number of simulated traces for the SPECK32/64 analysis target h_{SPECK} under additive Gaussian noise $\sigma \in \{0.7, 1.0, 1.5\}$. Despite the 16-bit key-hypothesis space, accumulating more traces steadily promotes the correct key toward the top of the ranking.

TABLE V
GE FOR THE h_{SPECK} IN A REAL-WORLD SETTING

	h_{SPECK}
k_0	2038.03
k_1	301.23
k_2	7250.43
k_3	517.79

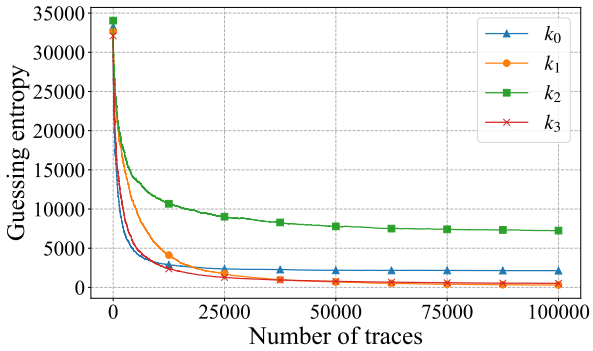

Fig. 7. GE for the SPECK32/64 round keys k_0–k_3 recovered from real power traces using the analysis target h_{SPECK}, shown as a function of the number of traces. Each curve is obtained via repeated random subsampling of the measurement set. GE decreases monotonically for all four keys, indicating practical key discrimination under measurement noise.

periments and computed GE. Each experiment was performed with a randomly selected 16-bit round key. For uniformly distributed inputs, we generated 100,000 simulated power traces per experiment and executed the key recovery procedure.

Table IV summarizes the round key recovery results for SPECK32/64 using simulated traces. When the noise standard deviation is $\sigma = 0.7$, the correct key attains an average rank of 1,669.49. This average rank increases to 1,860.46 for $\sigma = 1.0$ and to 2,746.24 for $\sigma = 1.5$. Although these ranks may appear less favorable than those observed for HIGHT, it is important to note that the target key space for SPECK32/64 is 16-bit (i.e., 65,536 candidates). In particular, the above ranks correspond to approximately the top 2.5%, 2.8%, and 4.2% of the entire candidate space, respectively, indicating that the attack still provides statistically meaningful discrimination even under the high-noise condition $\sigma = 1.5$.

Figure 6 shows how GE changes as a function of the number of simulated power traces under each noise condition. Consistent with the simulation results for HIGHT, we observe a clear decreasing trend in GE as the number of simulation traces increases, indicating that additional traces progressively reduce the rank of the correct key.

2) Evaluation on Real Power Traces: As in the HIGHT experiments, we performed blind SCA-based round key recovery for SPECK32/64 using real power consumption traces measured during SPECK32/64 encryption, targeting the four 16-bit round keys k_0, k_1, k_2, and k_3.

To evaluate feasibility in a real setting, we conducted 1,000 independent experiments under the same configuration as in the simulation setting and computed the GE for each target round key. All experiments were performed with a fixed secret key. We collected a total of 200,000 power traces, and in each experiment we randomly selected 100,000 traces from the collected set to perform key recovery. Table V reports the resulting GE values for the four round keys. Using 100,000 real traces, the GE values are 2038.03 for k_0, 301.23 for k_1, 7250.43 for k_2, and 517.79 for k_3, suggesting statistically meaningful key recovery across all four targets, even in the presence of measurement noise.

Figure 7 illustrates how GE evolves as a function of the number of real traces for each of the four round keys. As the number of traces increases, GE decreases monotonically for all keys, confirming that additional measurements progressively improve the key ranking performance in the real-world setting.

Finally, when compared with the success criterion defined in Section IV-B3, all four round keys satisfy $GE \leq 10{,}000$, and thus the master key can be recovered after an exhaustive search over the reduced candidate set. Therefore, these results demonstrate that the proposed method can recover the SPECK32/64 master key using only power trace information, even without any plaintext or ciphertext knowledge.

VI. CONCLUSION

In this paper, we proposed MALEAK, a blind side-channel analysis framework that exploits carry-induced leakages in modular addition to enable key recovery on ARX-based block ciphers without access to plaintext or ciphertext. By relying solely on side-channel traces, MALEAK operates under a more restricted adversary model than conventional SCA.

We validated MALEAK by defining two generalized key-dependent nonlinear functions that incorporate modular addition and conducting extensive simulation-based key recovery experiments under multiple noise levels. We further instantiated MALEAK for ARX-based block ciphers by deriving

cipher-specific attack procedures for HIGHT and SPECK, and evaluated them using both simulated traces and real power traces collected from an ARM Cortex-M4 device. Across these settings, the results confirm that modular addition nonlinearity can be statistically distinguished in a blind setting and that practical key recovery is feasible when suitable PoIs are available.

Several limitations remain. First, fully blind PoI selection is still an open problem. Following prior blind-SCA practice, we assumed oracle-assisted PoI selection. Second, our evaluation focused on 8-bit and 16-bit modular addition. Extending MALEAK to 32-bit or 64-bit word sizes is an important direction for future work. Third, incorporating recent deep-learning-based blind SCA techniques [38] into MALEAK may further improve robustness under high-noise conditions.

REFERENCES

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*. Springer, 1999, pp. 388–397.

[2] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2004, pp. 16–29.

[3] L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F.-X. Standaert, and N. Veyrat-Charvillon, "Mutual information analysis: a comprehensive study," *Journal of Cryptology*, vol. 24, no. 2, pp. 269–291, 2011.

[4] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, "Mutual information analysis: A generic side-channel distinguisher," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2008, pp. 426–442.

[5] N. Veyrat-Charvillon and F.-X. Standaert, "Mutual information analysis: how, when and why?" in *International workshop on cryptographic hardware and embedded systems*. Springer, 2009, pp. 429–443.

[6] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2002, pp. 13–28.

[7] O. Choudary and M. G. Kuhn, "Efficient template attacks," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2013, pp. 253–270.

[8] C. Rechberger and E. Oswald, "Practical template attacks," in *International Workshop on Information Security Applications*. Springer, 2004, pp. 440–456.

[9] T. Bartkewitz and K. Lemke-Rust, "Efficient template attacks based on probabilistic multi-class support vector machines," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2012, pp. 263–276.

[10] M. O. Choudary and M. G. Kuhn, "Efficient stochastic methods: Profiled attacks beyond 8 bits," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2014, pp. 85–103.

[11] L. Lerman and O. Markowitch, "Efficient profiled attacks on masking schemes," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1445–1454, 2018.

[12] G. Perin, Ł. Chmielewski, and S. Picek, "Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 337–364, 2020.

[13] G. Perin, L. Wu, and S. Picek, "Exploring feature selection scenarios for deep learning-based side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 828–861, 2022.

[14] O. Bronchain, F. Durvaux, L. Masure, and F.-X. Standaert, "Efficient profiled side-channel analysis of masked implementations, extended," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 574–584, 2022.

[15] P. Cao, C. Zhang, X. Lu, D. Gu, and S. Xu, "Improving deep learning based second-order side-channel analysis with bilinear cnn," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3863–3876, 2022.

[16] Y. Li, Y. Huang, Y. Guo, C. Ge, F. Kong, and Y. Ren, "Adaptive chosen-plaintext deep learning-based side-channel analysis," *IEEE Internet of Things Journal*, 2024.

[17] A. Liu, A. Wang, S. Sun, C. Wei, Y. Ding, Y. Wang, and L. Zhu, "Cl-sca: A contrastive learning approach for profiled side-channel analysis," *IEEE Transactions on Information Forensics and Security*, 2025.

[18] Y. Chen, B. Wang, C. Su, A. Li, Y. Tang, and G. Li, "Enhancing model generalization for efficient cross-device side-channel analysis," *IEEE Transactions on Information Forensics and Security*, 2025.

[19] L. Lerman, G. Bontempi, and O. Markowitch, "A machine learning approach against a masked aes: Reaching the limit of side-channel attacks with a learning model," *Journal of Cryptographic Engineering*, vol. 5, no. 2, pp. 123–139, 2015.

[20] L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F.-X. Standaert, "Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis)," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2015, pp. 20–33.

[21] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: a first study," *Journal of Cryptographic Engineering*, vol. 1, no. 4, pp. 293–302, 2011.

[22] W. Gao, J. Cai, Y. Yan, P. Guo, and T. Hu, "Side-channel analysis based on multiple leakage models ensemble," *IEEE Transactions on Dependable and Secure Computing*, 2025.

[23] L. Wu, Y.-S. Won, D. Jap, G. Perin, S. Bhasin, and S. Picek, "Ablation analysis for multi-device deep learning-based physical side-channel analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 3, pp. 1331–1341, 2023.

[24] B. Gierlichs, L. Batina, B. Preneel, and I. Verbauwhede, "Revisiting higher-order dpa attacks: Multivariate mutual information analysis," in *Cryptographers' Track at the RSA Conference*. Springer, 2010, pp. 221–234.

[25] B. Timon, "Non-profiled deep learning-based side-channel attacks with sensitivity analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 107–131, 2019.

[26] A. Alipour, A. Papadimitriou, V. Beroulle, E. Aerabi, and D. Hély, "On the performance of non-profiled differential deep learning attacks against an aes encryption algorithm protected using a correlated noise generation based hiding countermeasure," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 614–617.

[27] D. Kwon, S. Hong, and H. Kim, "Optimizing implementations of non-profiled deep learning-based side-channel attacks," *IEEE Access*, vol. 10, pp. 5957–5967, 2022.

[28] K. Imafuku, S. Kawamura, H. Nozaki, J. Sakamoto, and S. Osuka, "Non-profiled deep learning-based side-channel analysis with only one network training," *IEEE Access*, vol. 11, pp. 83 221–83 231, 2023.

[29] T. Tosun and E. Savas, "Zero-value filtering for accelerating non-profiled side-channel attack on incomplete ntt-based implementations of lattice-based cryptography," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 3353–3365, 2024.

[30] D. Kwon and S. Hong, "Side-channel attack on aradi in non-profiling scenarios," *IEEE Access*, 2025.

[31] Q. Wu, W. Cheng, F. Zhang, and S. Guilley, "Statistical analysis of non-profiling higher-order distinguishers against inner product masking," *IEEE Transactions on Information Forensics and Security*, 2025.

[32] L. Batina, B. Gierlichs, and K. Lemke-Rust, "Differential cluster analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2009, pp. 112–127.

[33] I. Savu, M. Krček, G. Perin, L. Wu, and S. Picek, "The need for more: unsupervised side-channel analysis with single network training and multi-output regression," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2024, pp. 113–132.

[34] Y.-S. Won, D.-G. Han, D. Jap, S. Bhasin, and J.-Y. Park, "Non-profiled side-channel attack based on deep learning using picture trace," *IEEE Access*, vol. 9, pp. 22 480–22 492, 2021.

[35] B. Gérard and F.-X. Standaert, "Unified and optimized linear collision attacks and their application in a non-profiled setting," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 175–192.

[36] L. Zhang, A. A. Ding, Y. Fei, and P. Luo, "Efficient nonprofiling 2nd-order power analysis on masked devices utilizing multiple leakage points," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 5, pp. 843–855, 2017.

[37] P. Rogaway, M. Bellare, and J. Black, "Ocb: A block-cipher mode of operation for efficient authenticated encryption," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 3, pp. 365–403, 2003.

[38] A. Rezaeezade, T. Yap, D. Jap, S. Bhasin, and S. Picek, "Breaking the blindfold: deep learning-based blind side-channel analysis," in *Proceedings of the 34th USENIX Conference on Security Symposium*, ser. SEC '25. USA: USENIX Association, 2025.

[39] Y. Linge, C. Dumas, and S. Lambert-Lacroix, "Using the joint distributions of a cryptographic function in side channel analysis," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 199–213.

[40] C. Clavier and L. Reynaud, "Improved blind side-channel analysis by exploitation of joint distributions of leakages," in *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 2017, pp. 24–44.

[41] C. Clavier, L. Reynaud, and A. Wurcker, "Quadrivariate improved blind side-channel analysis on boolean masked aes," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2018, pp. 153–167.

[42] P. Ravi, D. Jap, S. Bhasin, and A. Chattopadhyay, "Machine learning based blind side-channel attacks on pqc-based kems-a case study of kyber kem," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 01–07.

[43] M. Sarry, H. Le Bouder, E. Maaloouf, and G. Thomas, "Blind side channel analysis against aead with a belief propagation approach," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2023, pp. 127–147.

[44] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong *et al.*, "Hight: A new block cipher suitable for low-resource device," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2006, pp. 46–59.

[45] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The simon and speck lightweight block ciphers," in *Proceedings of the 52nd annual design automation conference*, 2015, pp. 1–6.

[46] C. Beierle, A. Biryukov, L. C. Dos Santos, J. Großschädl, L. Perrin, A. Udovenko, V. Velichkov, Q. Wang, and A. Biryukov, "Schwaemm and esch: lightweight authenticated encryption and hashing using the sparkle permutation family," *NIST round*, vol. 2, pp. 162 893–162 908, 2019.

[47] T. Beyne, Y. L. Chen, C. Dobraunig, and B. Mennink, "Elephant v2," *NIST lightweight competition*, 2021.

[48] P. Schwabe, R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, G. Seiler, D. Stehlé, and J. Ding, "CRYSTALS-KYBER," National Institute of Standards and Technology, Tech. Rep., 2022, available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[49] J. Daemen and V. Rijmen, *The design of Rijndael*. New York: Springer, 2002, vol. 2, https://doi.org/10.1007/978-3-662-60769-5.

[50] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1. 2: Lightweight authenticated encryption and hashing," *Journal of Cryptology*, vol. 34, no. 3, p. 33, 2021.

[51] K. Lemke, K. Schramm, and C. Paar, "Dpa on n-bit sized boolean and arithmetic operations and its application to idea, rc6, and the hmac-construction," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 205–219.

[52] M. Zohner, M. Kasper, and M. Stöttinger, "Butterfly-attack on skein's modular addition," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2012, pp. 215–230.

[53] C. H. Gebotys, B. A. White, and E. Mateos, "Preaveraging and carry propagate approaches to side-channel analysis of hmac-sha256," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 1, pp. 1–19, 2016.

[54] Y. Belenky, I. Dushar, V. Teper, V. Bugaenko, O. Karavaev, L. Azriel, and Y. Kreimer, "Carry-based differential power analysis (cdpa) and its application to attacking hmac-sha-2," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 3, pp. 1–29, 2023.

[55] G. Hatzivasilis, K. Fysarakis, I. Papaefstathiou, and C. Manifavas, "A review of lightweight block ciphers," *Journal of cryptographic Engineering*, vol. 8, no. 2, pp. 141–184, 2018.

[56] B. Koo, D. Roh, H. Kim, Y. Jung, D.-G. Lee, and D. Kwon, "Cham: A family of lightweight block ciphers for resource-constrained devices," in *International conference on information security and cryptology*. Springer, 2017, pp. 3–25.

[57] D. Hong, J.-K. Lee, D.-C. Kim, D. Kwon, K. H. Ryu, and D.-G. Lee, "Lea: A 128-bit block cipher for fast encryption on common processors," in *international workshop on information security applications*. Springer, 2013, pp. 3–27.

[58] C. O'flynn and Z. Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 243–260.