Blind Deep-Learning Side-Channel Analysis of XTS-AES on a Commercial RISC-V Microcontroller

Keerthi Kamalakshan*†, Dirmanto Jap*†, Trevor Yap*†, Shivam Bhasin*†,
*Temasek Laboratories, Nanyang Technological University, Singapore,
†National integrated Centre For Evaluation, Nanyang Technological University, Singapore
{keerthi.kamalakshan,djap,trevor.yap,sbhasin}@ntu.edu.sg,

***Abstract*—The XTS-AES mode is widely deployed to provide storage encryption in embedded systems, where hardware accelerators are commonly used to protect sensitive firmware stored in external flash. Existing side-channel attacks against XTS-AES typically assume access to operation-level leakage, requiring precise segmentation of individual AES executions or exploiting implementation-specific temporal structures. Such assumptions are difficult to satisfy for commercial hardware accelerators, where cryptographic operations are tightly integrated and exposed only through coarse-grained leakage.**

This paper presents a blind side-channel analysis methodology for hardware XTS-AES implementations that operates directly on complete power traces without requiring operation-level segmentation or knowledge of execution boundaries. The proposed approach adapted the blind deep learning-based side channel analysis to recover the XTS-AES encryption key from full-trace measurements. We further show that recovery of key can help to reconstruct the sector-specific XTS tweak, eliminating the need to recover the second XTS key.

The proposed methodology is evaluated on the hardware XTS-AES of a commercial ESP32-C3 microcontroller. Experimental results demonstrate successful recovery of the encryption key, reconstruction of the sector-specific tweak, and demonstrates how this can be used to recover the flash contents. The results highlight the practical feasibility of blind whole-trace side-channel analysis against the security of hardware-accelerated XTS-AES designs.

***Index Terms*—Blind side-channel analysis, Flash Encryption, XTS-AES, AES accelerator, Embedded Security.**

I. Introduction

Firmware confidentiality is a foundational security requirement for Internet of Things (IoT) devices, where an adversary with physical access to the device can attempt to extract proprietary code, cryptographic secrets, or sensitive configuration data directly from non-volatile storage. To defend against this class of attack, modern System-on-Chip (SoC) platforms increasingly integrate hardware-based flash encryption, transparently encrypting firmware images at rest so that raw flash contents remain confidential even if physically extracted.

The de facto standard underlying most memory encryption schemes is XTS-AES, specified in IEEE P1619 [1] and NIST SP 800-38E [2]. Built on the algebraic properties of the XEX tweakable block cipher construction [3], XTS-AES is now embedded in dedicated hardware accelerators across mobile platforms, storage controllers, and IoT SoCs. Even though, side-channel analysis (SCA) [4], [5] has shown that the device can leak key material through measurable channels such as power consumption or electromagnetic (EM) emanation, the tweak construction in XTS-AES has significantly increase the difficulty for attacker to break when using traditional side-channel analysis.

XTS-AES operates on two independent AES keys, K_1 and K_2, rather than a single key as depicted in Figure 1. For each data unit (typically a disk or flash sector), the sector index i marked as a *tweak*, which is first encrypted under K_2 to produce a pseudorandom value T. This value is raised to successive powers of a primitive element $\alpha \in \mathrm{GF}(2^{128})$, one power per 128-bit block within the sector, yielding a distinct whitening mask for every block. In other words, the i-th mask T_i is computed iteratively as $T_i = \alpha \otimes T_{i-1}$, with $T_0 = T$. The plaintext block P_i is XORed with the mask T_i, encrypted under K_1, and XORed again with the same mask before being written out, the XEX ("xor-encrypt-xor") structure of Rogaway's tweakable block cipher [3]. In short, K_2 renders the per-block masks unpredictable, while K_1 protects the data itself.

The two-key structure is precisely what enlarges the side-channel attack surface of XTS-AES relative to a single-key AES mode. An adversary must, in principle, defeat two separate key operating over two distinct computational paths, the AES rounds themselves, and the $\mathrm{GF}(2^{128})$ modular multiplication that propagates K_2's output across a sector.

Prior works exploit the two operations separately. Luo et al. [6], [7] recovered both the tweak key K_2 and encryption key K_1 from FPGA-based XTS-AES by targeting the $\mathrm{GF}(2^{128})$ modular multiplication in the tweak-generation path. Unterluggauer and Mangard [8] extended this to an embedded SoC via EM-based DPA against a disk-encryption workflow. Krüger et al. [9] pushed further, breaking software XTS-AES running under a full operating system, where caches, branch prediction, and scheduling noise obscure the signal; their attack, built on the CRESTS trace-segmentation algorithm [10], exploits the known chronological ordering of XTS-AES sub-operations to segment and re-associate trace fragments despite this noise.

What unites these attacks is a dependence on *temporal separability* in software, the tweak AES computation, the $\mathrm{GF}(2^{128})$ multiplication, and the data AES execution occur in distinct, identifiable regions of the trace, letting an attacker isolate and target each stage independently. This is precisely the property that breaks down in hardware. In dedicated accelerators, such as the AES engine integrated into the Espressif

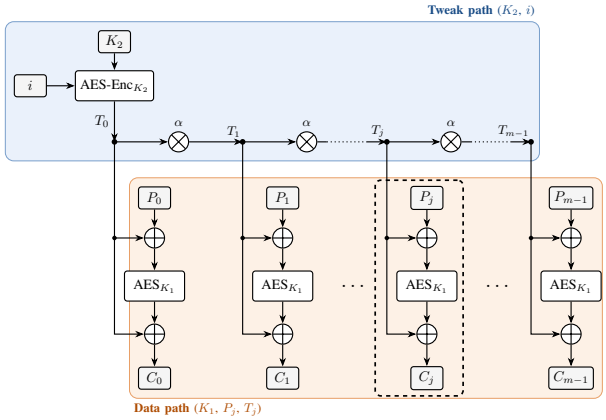


Fig. 1: XTS-AES encryption of one data unit (sector i).

ESP32-C3, which performs flash encryption during boot using two independent 128-bit XTS-AES keys, the sub-operations are structurally intertwined within the pipeline rather than executed as a sequential instruction stream. The resulting EM or power trace collapses into a single, temporally compact burst with no clear boundaries between sub-operations. This lack of temporal separability makes operation-level segmentation infeasible and blocks direct application of software-oriented attacks, including any technique that specifically targets the $\mathrm{GF}(2^{128})$ multiplication stage in isolation.

Hardware implementations of XTS-AES exhibit leakage that differs not only in structure but also in nature. The leakage originates primarily from Hamming-distance transitions in pipelined circuitry rather than software-style memory access patterns, complicating leakage modelling [11], [12], and is further obscured by countermeasures such as masking and clock jitter increasingly present in commercial accelerators [13].

Despite the ESP32-C3's widespread deployment in IoT devices that rely on its hardware flash encryption for firmware confidentiality, no general methodology exists for attacking hardware XTS-AES under these conditions. Existing attacks on the ESP32-C3 rely on algebraic relationships between consecutive tweaks under strong input-control assumptions [14], limiting their applicability in realistic, attacker-constrained settings.

In this work, we close this gap by developing a *blind side-channel attack* methodology that requires neither operation-level segmentation nor exploitable temporal structure in the captured traces. Instead, our approach operates directly on complete power traces of the hardware XTS-AES accelerator, enabling key extraction without prior knowledge of the internal execution boundaries. We demonstrate the practicality of this methodology by discussing how the approach can be extended to break flash encryption. The target device is *CLS-Ready Certified*,[1] satisfying industry security requirements against cyber threats, highlighting that the demonstrated attack represents a realistic threat against deployed systems rather than a laboratory scenario.

Our main contributions are as follows:

- We propose a blind side-channel attack methodology for hardware XTS-AES implementations that operates directly on complete power traces without requiring operation-level segmentation, temporal annotations, or prior knowledge of the execution boundaries.
- We demonstrate that the recovered encryption key can be used to derive the sector-specific XTS tweak directly from the collected side-channel traces, without requiring recovery of the second XTS key (K_2). This enables practical decryption while reducing the complexity of the attack compared to recovering both XTS keys.

The remainder of this paper is organized as follows. Section II presents the necessary background. Section III includes the state-of-the-art techniques. Section IV describes the proposed blind profiled DL-SCA methodology. Section V details the trace analysis framework for recovering the encryption key, reconstructing the sector-specific tweak. Section VI includes the evaluation results, followed by limitations and countermeasures in Section VII. Section VIII concludes the paper.

II. Background

A. Flash Encryption and XTS-AES

To protect firmware confidentiality against attackers with physical access to the external SPI flash, Espressif implements the flash encryption based on XTS-AES.[2] The encryption key is provisioned into eFuse during manufacturing and is not exposed on any software-accessible interface. The encryption and decryption are performed by a dedicated hardware accelerator that fetches the key material directly from the eFuse block on each invocation, so that neither key ever transits a general-purpose register or bus visible to software.

a) Flash Encryption Flow: XTS-AES is a tweakable block cipher mode of operation and is derived from Rogaway's XEX (Xor-Encrypt-Xor) construction [3] extended with ciphertext stealing to support data units whose length is not a multiple of the block size as depicted in Figure 1.

XTS-AES uses two independent k-bit keys, K_1 and K_2, where $k \in \{128, 256\}$. A data unit (typically a disk sector or flash page) is divided into m blocks of 128 bits, $P_0, P_1, \ldots, P_{m-1}$, and encrypted under a sector index i. For each block j, the ciphertext is computed as

$$C_j = \mathsf{AES}_{K_1}(P_j \oplus T_j) \oplus T_j, \qquad T_j = \mathsf{AES}_{K_2}(i) \otimes \alpha^j, \quad (1)$$

where $\otimes$ denotes multiplication in the finite field $\mathrm{GF}(2^{128})$ defined by the primitive polynomial $x^{128} + x^7 + x^2 + x + 1$, and α is the primitive element corresponding to x. The value T_j is referred to as the *tweak*. The tweak schedule produces a deterministic, position-dependent mask that breaks the codebook property of ECB while ensuring that identical plaintext blocks at different positions yield different ciphertexts.

Two properties of XTS-AES are central to side-channel analysis. First, the tweak $T_0 = \mathsf{AES}_{K_2}(i)$ depends only on K_2 and the sector index. Second, the intermediate value $P_j \oplus T_j$ entering the data-path AES invocation under K_1 is *masked*

[1] csa.gov.sg/our-programmes/certification-and-labelling-schemes/cls-ready/

[2] https://docs.espressif.com/projects/esp-iot-solution/en/latest/security_solution/flash_encryption.html

by the tweak: unlike textbook AES, the attacker does not directly control the input to the K_1 encryption. Any side-channel attack on K_1 must therefore either recover K_2 (and thus T_j) beforehand, or model the joint leakage of the tweak and data-path operations.

B. Side-Channel Analysis

Side-Channel Analysis (SCA) exploits the leakage dependency to recover secret material, the AES encryption key and tweak keys without mathematically breaking the underlying cipher. This threat is especially pertinent to flash encryption, where a single, long-lived key set protects the entire encrypted region over the device's lifetime, so that one successful key recovery compromises the confidentiality of all stored data.

The feasibility and strength of an SCA attack hinge on the adversary's assumptions, most notably, whether the plaintext or ciphertext is known, and whether a device identical to the target is available for characterisation. These assumptions give rise to three broad paradigms: *non-profiled*, *profiled*, and *blind* analysis, which we describe in turn and later revisit in the specific context of XTS-AES.

1) Non-Profiled Side-Channel Analysis: In a *non-profiled* attack scenario, the adversary only has access to the target device along with publicly available variable (e.g., plaintext or ciphertext). Various techniques have been developed for key recovery under this setting. For instance, Correlation Power Analysis (CPA) [15], Mutual Information Analysis (MIA) [16], collision attacks [17], and more recently, deep learning-based attacks [18], [19] have been widely studied and applied.

In case of XTS-AES, the adversary do not have direct access to the input of AES encryption as the plaintext is tweaked. Moreover, the simultaneous presence of tweak related and data-path related intermediates within overlapping clock windows further degrades the signal-to-noise ratio making it difficult to recover the secret key. We will demonstrate this using CPA during a preliminary study in Section V.

2) Profiled Side-Channel Analysis: In a *profiled* attack scenario, the adversary assumed to possess a device identical to the target, a *clone device*, on which the leakage can be characterised. The adversary has full control over the clone, including the ability to set the secret key, and can acquire an arbitrary number of traces from known, random public variable (plaintexts/ciphertexts). Profiling traces are collected from the clone device, whereas the attack traces are acquired from the target device. A profiled attack therefore decomposes into a *profiling phase* and an *attack phase*.

The commonly used profiling attack includes template attacks [20] and deep learning-based side-channel analysis (DLSCA). In the profiling phase, a distinguisher $\mathcal{F}$ is built from the profiling traces. This distinguisher approximates the conditional probability mass function $\Pr(\boldsymbol{T} \mid Z = z)$, where Z denotes the sensitive intermediate.

In the attack phase, $\mathcal{F}$ returns a probability score for each hypothetical sensitive value, i.e. $\boldsymbol{y}_i = \mathcal{F}(\boldsymbol{t}_i)$ for an attack trace $\boldsymbol{t}_i$. The log-likelihood score for each key hypothesis k^* is then accumulated as $s_{N_a}(k^*) = \sum_{i=1}^{N_a} \log\big(\boldsymbol{y}_i[z_{i,k^*}]\big)$, where N_a is the number of attack traces, and $z_{i,k^*} = C(p_i, k^*)$ are the hypothetical sensitive values under key hypothesis k^*, with p_i the public variable associated with trace $\boldsymbol{t}_i$ and C the targeted cryptographic primitive. The rank of the correct key k^* is defined by its position within sorted array of the log-likelihood score $G = [G_0, G_1, \ldots, G_{|\mathcal{K}|-1}]$. DL-SCA instantiates $\mathcal{F}$ with a neural network trained to predict the sensitive intermediate directly from raw or preprocessed traces. When applying DLSCA, we denote GE as average rank of the correct key over multiple independent experiments and $NTGE$ the minimum number of traces required to achieve $GE = 0$.

3) Blind Side-Channel Analysis: In side channel attacks, it is assumed that the adversary knows the public variable (e.g. plaintext/ciphertext). This assumption is often violated in flash encryption, where the inputs and outputs of the on-chip cryptographic engine (i.e. AES_{K_1}) are internal to the device and never exposed on an observable bus. The *blind* attack scenario addresses precisely this case: the adversary knows neither the public variable, and instead exploits the fact that the joint distribution between the input and output of a key-dependent non-linear function varies with the value of the secret. This setting was initially proposed by Linge *et al.* [21], who demonstrated that the secret key can be estimated solely from side-channel traces, without any knowledge of the processed data. The approach is then extended in [22], [23]. The core intuition of blind SCA is to (a) generate, for each key hypothesis k^*, a theoretical joint distribution of the non-linear function, (b) estimate an empirical joint distribution from the measured side-channel traces, and (c) statistically compare the theoretical and empirical distributions to recover the secret.

With respect to XTS-AES, as tweak prevents the adversary from accessing the actual input to AES_{K_1}, even when the plaintext and ciphertext are known. As such, the blind paradigm most faithfully captures the flash-encryption threat model and is the setting on which the remainder of this work focuses.

III. Related Work

Over the past few years, researchers have demonstrated side-channel key recovery against AES implementations, including hardware implementations on commodity microcontrollers. Attacks specifically targeting XTS-AES are comparatively scarce, and each prior effort makes assumptions that limit its applicability to a realistic, unmodified commercial accelerator such as the ESP32-C3. Luo et al. [6] analysed XTS-AES in the context of self-encrypting drives implemented on FPGA, combining simple power analysis (SPA) with CPA to characterise leakage from the tweak and data encryption paths. Their analysis establishes that XTS-mode hardware leaks exploitable information, but stops short of full key recovery. Unterluggauer and Mangard [8] exploited the algebraic structure of consecutive tweaks within a data unit ($T_{j+1} = T_j \otimes \alpha$) to mount a DPA-style attack against a memory-encryption SoC, achieving full key recovery without segmenting individual traces, but using fault injection techniques. Instead, their attack requires the ability to collect encryptions at chosen, consecutive data-unit offsets and combines whole-trace leakage across this

TABLE I: State-of-the-art attacks on XTS-AES. ● = property present, ○ = absent.

Work	H/W	Temporal Inseparability	Algebraic Inseparability	Blind Attack	Passive (No Fault)
CRESTS [10]	○	○	●	○	●
Kruger et al. [9]	○	○	●	○	●
Luo et al. [6]	●	○	●	○	●
Unterluggauer et al. [8]	●	●	○	○	○
Courdesses [14]	●	●	○	○	○
This Paper	●	●	●	●	●

structured input set using the known algebraic relationship between tweaks. More recently, Courdesses [14] demonstrated practical key recovery on the ESP32 family using the same differential tweak relationship, achieving full key recovery via CPA, they also used fault attacks to dump the flash contents. This is the closest prior attack on our target device, but as with Unterluggauer and Mangard [8], it depends on triggering encryption of chosen, consecutive flash offsets so that the algebraic relationship between successive tweaks can be exploited across traces, an input-control assumption that does not hold in many realistic deployment settings where offset selection is constrained or unavailable to the attacker.

For software AES, Kruger et al. [10] proposed CRESTS, a chronology-based segmentation framework that decomposes long EM traces into per-operation segments using the regular, instruction-level timing signature of software execution, then attacks each segment independently with DL-SCA. CRESTS achieves strong full-key recovery on software targets, but its core assumption that sub-operation boundaries are recoverable from the temporal structure of a single trace is precisely the property that collapses in a hardware accelerator, where the entire computation occurs as a single, fixed-duration burst. Kruger et al. [9] extended this line of work with a profiled attack combining algebraic side-channel analysis (ASCA) and CPA against a microprocessor implementation, again relying on the ability to segment and localise leakage to specific instructions. As with CRESTS, this approach is architecturally tied to software-style sequential execution and offers no direct path to attacking a monolithic hardware pipeline.

Taken together, these works reveal two distinct forms of separability that prior key recovery attacks depend on: *temporal* separability, which lets software-oriented methods segment a single trace into per-operation windows, and *algebraic* separability, which lets hardware-oriented methods forgo trace segmentation but instead requires chosen-input control over consecutive tweaks across multiple traces on the target device. Neither assumption holds for our experimental setting, the ESP32-C3's accelerator produces a single, structurally fused leakage singals that cannot be segmented, and our threat model does not assume the attacker can select consecutive flash offsets to encrypt. This is the gap our work closes. We present a profiled deep learning based *blind* end-to-end DL-SCA methodology that operates directly on whole-trace, unsegmented leakage from the hardware accelerator, without relying on either form of separability, temporal or algebraic. In contrast to previous works, our method does not require the knowledge of the input and output in the attack phase, albeit requiring profiling phase on clone device. By learning

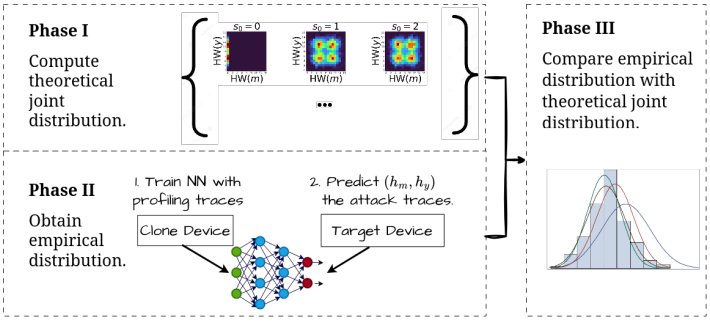


Fig. 2: Overview of Profiled DL-BSCA

directly from the compact, fused leakage burst produced by the pipeline, our approach sidesteps the structural and input-control assumptions that every prior technique depends on. To our knowledge, this constitutes the first full key-recovery blind attack on hardware XTS-AES that requires neither chosen-input control over tweak sequences nor operation-level segmentation.

IV. PROFILED DEEP LEARNING-BASED BLIND SCA

A. Threat Model

We consider an attacker who has physical access to a clone of the target device, as described in Section II-B. The clone device is assumed to be functionally identical to the target device and provides the attacker with the ability to perform side-channel measurements and conduct offline analysis. For the target device, we assume that the XTS-AES keys, including both the encryption key K_1 and the tweak key K_2, are provisioned during device programming and remain fixed during operation. However, the attacker does not have direct access to the secret keys (K_1, K_2), the plaintext (pt), or the tweak value (T) processed by the target device. Consequently, the attack is performed in a *blind* setting, where neither the cryptographic inputs nor the internal key material of the target device are known to the adversary.

B. Profiled DL-BSCA: Methodology

Our attack methodology is motivated by the DL-BSCA framework proposed in [24], which enables side-channel analysis in a blind setting. We adapted the proposed framework, which we refer to as Profiled DL-BSCA. As depicted in Figure 2, the proposed methodology consists of three phases: P1 Computing the theoretical joint distribution, P2 Obtaining the empirical distribution from the measured traces, and P3 Comparing the empirical distribution with the theoretical joint distribution to identify the correct secret-dependent hypothesis. We formally describe each step in below. Subsequently, we also describe how the Profiled DL-BSCA framework is adopted for XTS-AES to recover both the encryption key and the tweak in Section IV-C.

Thoughout the rest of the paper, the term m denotes a *public* variable (a plaintext) and y the *sensitive* intermediate variable derived from the public variable. For example,

$$y = \text{Sbox}(m \oplus k^*), \tag{2}$$

Symbol	Meaning
$\mathcal{B},\ \mathcal{H}$	bit-width; $\mathcal{H} = \{0, \ldots, \mathcal{B}\}$
$k^*,\ k$	correct key byte; key hypothesis, $k \in \mathcal{K}$
$m,\ y$	public variable; sensitive variable, Eq. (2)
(h_m^*, h_y^*)	*true* HW pair $\big(\mathrm{HW}(m), \mathrm{HW}(y)\big) \in \mathcal{H}^2$
(h_m, h_y)	*estimated* HW pair, output of the DNN
$P_k(\cdot,\cdot)$	theoretical joint distribution under hypothesis k
σ_m, σ_y	std. dev. of the estimation error on h_m, h_y
$\boldsymbol{\ell}_i$	i-th attack trace; N = number of attack traces

where k^* is the secret key byte under attack. The bit width of m and y (e.g. $\mathcal{B} = 8$ for AES) is denoted as $\mathcal{B}$, and $\mathcal{H} = \{0, 1, \ldots, \mathcal{B}\}$ for the set of attainable Hamming weights, and $\mathcal{K}$ for the key space. Table above summarizes the notation used in the rest of the paper.

The attack compares an *empirical* joint distribution recovered from the target traces against *theoretical* joint distributions pre-computed for every key hypothesis. It proceeds in three phases, P1–P3 as depicted in Figure 2. The detailed description of different steps is as follows:

P1 ***Pre-computing the theoretical joint distribution.*** To obtain the theoretical joint distribution for a key hypothesis k and a randomly chosen message $m \in \{0,1\}^{\mathcal{B}}$, we use Equation 3. The equation gives the probability of observing a particular pair of Hamming weights (h_m^*, h_y^*), where h_m^* is the Hamming weight of the message m and h_y^* is the Hamming weight of the S-box output $\mathrm{Sbox}(m \oplus k)$.

$$\begin{aligned} P_k(h_m^*, h_y^*) \ &\triangleq\ \Pr\big[(h_m^*, h_y^*) \,\big|\, k\big] \\ &=\ \frac{\big|\{\, m : \mathrm{HW}(m) = h_m^*,\ \mathrm{HW}(\mathrm{Sbox}(m \oplus k)) = h_y^* \,\}\big|}{2^{\mathcal{B}}}. \end{aligned} \tag{3}$$

In practice, we compute this distribution by considering every possible message m for each key hypothesis k. For each message, we calculate the two Hamming weights and count how many times each pair (h_m^*, h_y^*) occurs. Dividing these counts by the total number of messages, $2^{\mathcal{B}}$, gives the corresponding probabilities.

This process produces a lookup table containing one distribution for each key hypothesis. Since each Hamming weight can take values from 0 to $\mathcal{B}$, the table has size $|\mathcal{K}| \times (\mathcal{B}+1)^2$.

The table requires $|\mathcal{K}| \cdot 2^{\mathcal{B}}$ S-box evaluations to compute. Importantly, this computation does not require any power or EM measurements. It depends only on the mathematical structure of the cipher and can therefore be performed once offline. The resulting table can then be reused for all subsequent attacks.

P2 ***Obtaining the empirical distribution.*** We obtain the empirical distribution by training a deep neural network (DNN) $\mathcal{N}$ to infer the Hamming-weight pair (h_m^*, h_y^*) from each leakage trace. Given a trace $\boldsymbol{\ell} \in \mathbb{R}^D$, the network outputs a probability distribution over all possible Hamming-weight pairs:

$$\begin{aligned} \mathcal{N} :& \mathbb{R}^D \to [0,1]^{(\mathcal{B}+1)^2}, \\ & \boldsymbol{\ell} \mapsto \big(\Pr[(h_m^*, h_y^*) = (a,b) \mid \boldsymbol{\ell}]\big)_{(a,b)\in\mathcal{H}^2}. \end{aligned} \tag{4}$$

During profiling, the corresponding message m and key k are known, allowing us to compute the ground truth pair (h_m^*, h_y^*) for each trace. We encode this pair as a single class index $c = h_m^*(\mathcal{B}+1) + h_y^*$ and train the DNN using standard supervised classification. Since each Hamming weight can take values from 0 to $\mathcal{B}$, the network has $(\mathcal{B}+1)^2$ output classes; for $\mathcal{B} = 8$, this results in $9^2 = 81$ classes. The trained DNN therefore provides, for each attack trace, an empirical probability distribution over the possible Hamming-weight pairs.

Difference from DL-BSCA. Our profiling procedure differs from DL-BSCA [24] because our threat model assumes access to a clone device. Consequently, the Hamming-weight pair associated with each profiling trace can be determined directly from the known message and key, providing ground-truth labels (h_m^*, h_y^*) for supervised training. In contrast, DL-BSCA considers a fully blind setting without a clone device, where profiling traces are unlabelled and therefore require an explicit point-of-interest (PoI) selection and a dedicated labelling procedure. Our approach consequently avoids both PoI selection and surrogate labelling during profiling.

P3 ***Matching the empirical against the theoretical distribution.*** At attack phase, each attack trace $\boldsymbol{\ell}_i$ is passed through the trained network to obtain the empirical estimate

$$(h_m, h_y)_i \ =\ \arg\max_{(a,b)\in\mathcal{H}^2} \mathcal{N}(\boldsymbol{\ell}_i)_{(a,b)}, \tag{5}$$

which serves as the empirical label of trace i in phase P3.

We model the estimate produced by Eq. (5) as the true pair corrupted by additive noise,

$$(h_m, h_y)_i \ =\ (h_m^* + \epsilon_{m,i},\ h_y^* + \epsilon_{y,i}) \tag{6}$$

where $\epsilon_{m,i}$ and $\epsilon_{y,i}$ are Gaussian noise with standard deviations σ_m and σ_y, respectively, and $\epsilon_{m,i}$ and $\epsilon_{y,i}$ assumed independent. The standard deviations σ_m, σ_y are estimated on the profiling set, where the ground truth is known.

Write $\boldsymbol{o}_i = (h_m, h_y)_i$ for the i-th observation. By Bayes' rule,

$$\Pr[k \mid \boldsymbol{o}_i] = \frac{\Pr[\boldsymbol{o}_i \mid k] \cdot \Pr[k]}{Pr[\boldsymbol{o}_i]}. \tag{7}$$

Since the denominator $\Pr[\boldsymbol{o}_i]$ is a normalization term independent of the key, so we can ignore it. Furthermore, $\Pr[k]$ is assumed to be uniformly distributed, hence we can conclude that

$$\Pr[k \mid \boldsymbol{o}_i] \ \propto\ \Pr[\boldsymbol{o}_i \mid k]. \tag{8}$$

Assuming the N observations are mutually independent given k,

$$\begin{aligned} \Pr\big[k \mid \boldsymbol{o}_1, \ldots, \boldsymbol{o}_N\big] \ &\propto\ \prod_{i=1}^{N} \Pr[\boldsymbol{o}_i \mid k] \\ &=\ \Pr[\boldsymbol{o}_N \mid k] \cdot \Pr\big[k \mid \boldsymbol{o}_1, \ldots, \boldsymbol{o}_{N-1}\big], \end{aligned} \tag{9}$$

so the score can be updated incrementally, one trace at a time.

The per-trace likelihood is obtained by marginalising over the (unknown) true Hamming-weight pair, using the law of total probability:

$$\Pr[\boldsymbol{o}_i \mid k] \ =\ \sum_{(h_m^*, h_y^*)\in\mathcal{H}^2} \underbrace{\Pr\big[\boldsymbol{o}_i \mid (h_m^*, h_y^*)\big]}_{\text{noise model, Eq. (6)}} \cdot \underbrace{P_k(h_m^*, h_y^*)}_{\text{theoretical, Eq. (3)}}. \tag{10}$$

The second factor is read off the table built in P1. The first factor is the probability of the realised noise, which under Eq. (6) factorises as

$$\begin{aligned}&\Pr\big[\boldsymbol{o}_i \mid (h_m^*, h_y^*)\big] \\ &\quad = \Pr\big[\epsilon_{m,i} = h_m^* - h_{m,i}\big] \cdot \Pr\big[\epsilon_{y,i} = h_y^* - h_{y,i}\big] \\ &\quad = \Big(\frac{1}{\sigma_m\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{h_{m,i}-h_m^*}{\sigma_m})^2}\Big) \cdot \Big(\frac{1}{\sigma_y\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{h_{y,i}-h_y^*}{\sigma_y})^2}\Big).\end{aligned} \tag{11}$$

In practice the product in Eq. (9) underflows quickly, so we accumulate log-likelihoods instead and define the score of hypothesis k after N traces as

$$d_N(k) \;=\; \sum_{i=1}^{N} \log \sum_{(h_m^*, h_y^*) \in \mathcal{H}^2} \Pr\big[\boldsymbol{o}_i \mid (h_m^*, h_y^*)\big] \cdot P_k(h_m^*, h_y^*), \tag{12}$$

which is order-equivalent to $\Pr[k \mid \boldsymbol{o}_1, \ldots, \boldsymbol{o}_N]$.

Attack Metrics. We sort all hypotheses by decreasing score $d_N(k)$ into the key-guessing vector $\boldsymbol{G} = (G_0, \ldots, G_{|\mathcal{K}|-1})$, so that G_0 is the most likely and $G_{|\mathcal{K}|-1}$ the least likely candidate. The position of a candidate in $\boldsymbol{G}$ is its *key rank*; an adversary willing to enumerate the top o candidates succeeds whenever the rank of k^* is below o.

The *guessing entropy* GE denotes the rank of the correct key k^*, averaged over independent attack experiments (we use 100 experiments throughout Section VI). Since ranks are 0-indexed, $GE = 0$ means that k^* is ranked first on average with N attack traces; we then consider the key to be successfully recovered, and report the smallest such N. In our work, we perform divide and conquer approach and recover one byte at a time. The traces can then be reused to recover next byte.

C. Adapting Profiled DL-BSCA to XTS-AES

As in classical side-channel analysis on AES, we also conduct the attack in a divide-and-conquer manner. The sensitive variable of the first-round S-box output that we are targeting is

$$Y^b = \mathrm{Sbox}\big(P_0^b \oplus T_0^b \oplus K_1^b\big), \qquad b = 0, \ldots, 15. \tag{13}$$

where P_0^b, T_0^b, K_1^b, and Y^b denote the b-th byte of P_0, T_0, K_1, and Y, respectively.

For standard AES, the attack can be formulated using the known plaintext m and the sensitive variable $y = \mathrm{Sbox}(m \oplus k^*)$, where k^* is the correct key byte. Thus, the attack involves a known input variable m and a key-dependent sensitive variable y. In XTS-AES, however, the first-round S-box input contains the additional unknown tweak byte, $P_0^b \oplus T_0^b \oplus K_1^b$. This appears to prevent a direct recovery of K_1, since T_0^b is unknown and depends on K_2.

Our key observation is that this apparent dependency can be avoided by changing how the S-box input is decomposed. Instead of recovering the tweak first, we first absorb the unknown tweak into the variable used as the input of the attack. Once K_1 has been recovered, the same S-box computation can then be reformulated to recover the tweak. This leads to the two-step procedure illustrated below.

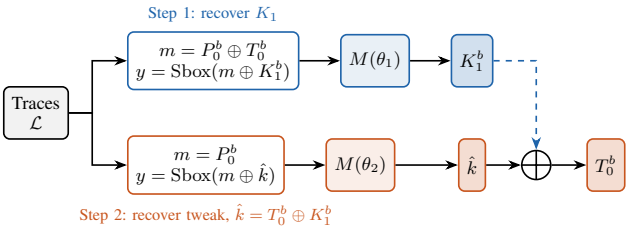


Fig. 3: Overview of the Proposed Approach.

Recovering the K_1 of XTS-AES. We first recover K_1 without explicitly recovering either T_0 or K_2. To do so, we absorb the unknown tweak into the input variable of the S-box relation. Specifically, we set

$$\begin{aligned} m \;&=\; P_0^b \oplus T_0^b, \qquad y \;=\; Y^b \\ &\qquad\quad = \mathrm{Sbox}((P_0^b \oplus T_0^b) \oplus K_1^b) \\ &\qquad\quad = \mathrm{Sbox}(m \oplus K_1^b), \end{aligned} \tag{14}$$

which has exactly the same form as Eq. (2). Therefore, the Profiled DL-BSCA framework described in Section IV-B can be applied directly. The DNN is trained to predict $\big(\mathrm{HW}(P_0^b \oplus T_0^b), \mathrm{HW}(y)\big)$, and K_1^b is recovered without first determining T_0 or K_2.

The key point is that the attack only requires the algebraic relation between m and y to have the required form; it does not require the actual value of m to be explicitly known. Hence, the unknown tweak does not need to be recovered before attacking K_1. This property is particularly important in the blind setting considered in our work.

Recovering the Tweak of XTS-AES. Once K_1^b is known, we recover the corresponding tweak byte by using the same S-box computation with a different decomposition of its input. We now set

$$\begin{aligned} m \;&=\; P_0^b, \\ y \;&=\; \mathrm{Sbox}(P_0^b \oplus (T_0^b \oplus K_1^b)) \;=\; \mathrm{Sbox}(m \oplus \hat{k}), \\ \hat{k} \;&\triangleq\; T_0^b \oplus K_1^b. \end{aligned} \tag{15}$$

Formally this is again an instance of Eq. (2), with the *effective* key $\hat{k}$. Running Profiled DL-BSCA on this formulation yields $\hat{k}$, from which the tweak follows immediately:

$$T_0^b \;=\; \hat{k} \oplus K_1^b, \tag{16}$$

since K_1^b was already recovered previously. Repeating this per byte recovers T_0 in a divide-and-conquer manner. Figure 3 provides an overview of the overall procedure.

An important practical advantage of this two-step formulation is that both attacks use the *same set of side-channel traces*. No additional measurement campaign on the target device is required. The only difference is the labeling of the input variable: the first attack uses $\mathrm{HW}(P_0^b \oplus T_0^b)$, whereas the second uses $\mathrm{HW}(P_0^b)$. Moreover, the two Profiled DL-BSCA models can be trained and evaluated independently, allowing the two attack stages to be computed in parallel.

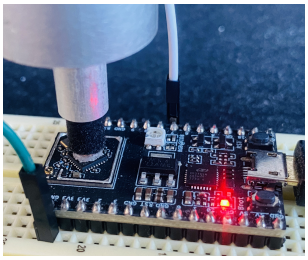


Fig. 4: EM probe positioned on the ESP32-C3.

V. Analysis Procedure

A. Preliminary Study: Non-Blind DL-SCA on Hardware AES

Before analyzing XTS-AES, we first characterize the leakage behavior of the hardware AES core on the ESP32-C3 using a non-blind DL-SCA analysis. This preliminary study serves two purposes: it establishes the measurement and acquisition methodology for the underlying AES implementation and provides a baseline for evaluating the additional tweak-dependent structure introduced by XTS-AES. Since XTS-AES relies on the same AES primitive, characterizing the leakage of the standalone AES core provides a reference point for understanding the leakage observed in the XTS-AES implementation. Hence, in this section, we conduct an analysis in a white-box setting, where the inputs to the AES execution are fully known.

1) Measurement Setup and Trace Acquisition: We initially invoked the hardware AES engine through the standard ESP-IDF AES API. Although the resulting traces reflected system-level activity, they also contained substantial background noise and peripheral operations, which made precise analysis computationally difficult, particularly when processing over one million traces. Our EM acquisition setup used for the measurements is depicted in Figure 4.

```
// Enable AES peripheral clock and disable DMA
REG_SET_BIT(SYSTEM_PERIP_CLK_EN1_REG, SYSTEM_CRYPTO_AES_CLK_EN);
REG_CLR_BIT(SYSTEM_PERIP_RST_EN1_REG, SYSTEM_CRYPTO_AES_RST);
REG_WRITE(AES_DMA_ENABLE_REG, 0);

// AES-128 encrypt mode
REG_WRITE(AES_MODE_REG, 0x0);

// Load plaintext
for (int i = 0; i < 4; i++) {
    REG_WRITE(AES_TEXT_IN_BASE + (i * 4), ((const uint32_t *)input_buffer)[i]);
}

// Trigger signal high
gpio_set_level(TRIGGER_GPIO, 1);

// Start encryption and wait for completion
REG_WRITE(AES_START_REG, 1);
while (REG_READ(AES_IDLE_REG) == 0);

// Trigger signal low
gpio_set_level(TRIGGER_GPIO, 0);
// Read ciphertext
for (int i = 0; i < 4; i++) {
    ((uint32_t *)output_buffer)[i] = REG_READ(AES_TEXT_OUT_BASE + (i * 4));
}
```

Listing 1: Direct register-level invocation of the ESP32-C3 hardware AES engine, bypassing the ESP-IDF high-level API and DMA control.

To isolate the AES encryption activity and obtain cleaner leakage signatures, we adopted a direct register-based invocation of the ESP32-C3 hardware AES engine, bypassing the high-level API layers and DMA control (Listing 1). This low-level approach substantially reduced the trace length, capturing only the essential encryption phase. The same acquisition strategy is later reused for the XTS-AES analysis.

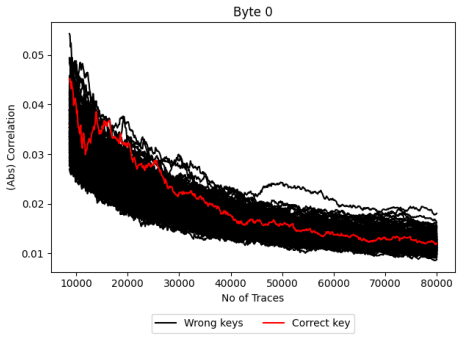


Fig. 5: Preliminary study using CPA.

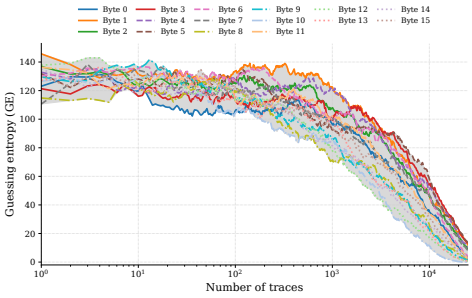


Fig. 6: Guessing entropy per key byte using DL-SCA

2) Trace Alignment and Correlation Analysis: Owing to the shortened trace duration and low signal amplitude, precise alignment was essential before applying correlation-based analysis. We have used cross-correlation and reference-based alignment techniques to synchronize traces across multiple acquisitions, improving the temporal consistency of the AES traces.

As shown in Figure 5, the key-rank evolution remained inconsistent as the number of traces increased, indicating that the residual leakage was too weak to enable reliable key recovery under first-order CPA assumptions. The results suggest that the hardware co-processor employs internal side-channel countermeasures that hinder key recovery via classical CPA.

3) Breaking the AES Core with DL-SCA: To further investigate the residual leakage from the ESP32-C3 hardware AES accelerator, which proved resistant to classical first-order CPA, we employed DL-SCA as describe in Section II-B2. Unlike correlation-based methods, DL-SCA leverages neural networks to learn complex, non-linear relationships between electromagnetic leakage and secret-dependent intermediate values, enabling successful key recovery even in the presence of countermeasures such as desynchronisation or masking [25].

We evaluated DL-SCA using a CNN with an identity (ID) leakage model. Model selection was performed via a random hyperparameter search over 100 candidate CNNs. Each candidate was trained on 50,000 traces and evaluated on 25,000 attack traces. For robust evaluation, we ran 100 independent

TABLE II: NTGE (GE in brackets, if $GE > 0$) for each key byte in the DLSCA experiment.

Byte	NTGE (GE)	Byte	NTGE (GE)	Byte	NTGE (GE)	Byte	NTGE (GE)
0	(3.62)	4	(10.28)	8	(1.45)	12	(0.65)
1	(12.01)	5	(13.55)	9	(1.22)	13	(0.85)
2	(9.55)	6	(4.0)	10	(0.03)	14	(3.33)
3	(13.67)	7	(7.46)	11	(3.06)	15	(2.46)

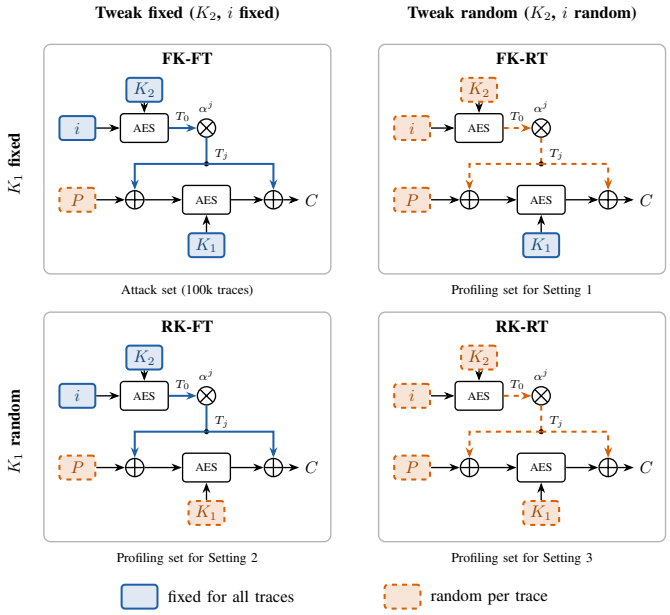


Fig. 7: The four acquisition datasets, marked on the XTS data flow. The plaintext P is random in every dataset. The derived tweak T_0, T_j is fixed exactly when both K_2 and i are fixed. The key K_1 used in FK-RT is the same as in FK-FT.

inference trials per candidate, each using a randomly sampled subset of the attack data, and recorded the NTGE as the primary metric. The best-performing model was selected as the one achieving the lowest NTGE. Figure 6 depicts the guessing entropy plot per key byte, and Table II provides more numerical details. We observed that it achieves better result compared to previous CPA.

Inferences: The inferences from the preliminary study are as follows: (1) The register-level acquisition and alignment methodology developed here yields traces of sufficient quality for profiled attacks and can be transferred unchanged to XTS-AES. (2) The failure of CPA alongside the success of DL-SCA establishes that profiled deep-learning attacks, rather than classical correlation analysis, are the appropriate tool for evaluating the ESP32-C3 cryptographic accelerators. We therefore adopt the similar framework when turning to the XTS-AES implementation, where the tweak derivation introduces additional secret-dependent intermediates beyond those examined here.

B. Datasets for Evaluating the XTS-AES Engine

For the analysis, we acquired four datasets from various settings, as illustrated in Figure 7. The datasets are designed to evaluate the proposed attack under different combinations of fixed and randomized XTS parameters. In particular, K_1, K_2, the sector index i, and the plaintext P are varied across the datasets. Since the tweak T_0 is derived from K_2 and i, it remains fixed when both parameters are fixed and becomes randomized when either parameter varies. A summary of the acquired datasets is provided in Table III, where F and R denote fixed and randomized, respectively.

TABLE III: Summary of datasets (F = fixed, R = random).

Dataset	K_1	K_2	i	T_0	P	Role
FK-FT	F	F	F	F	R	Attack set (all settings)
FK-RT	F	R	R	R	R	Profiling, Setting 1 (20k)
RK-FT	R	F	F	F	R	Profiling, Setting 2 (100k)
RK-RT	R	R	R	R	R	Profiling, Setting 3 (100k)

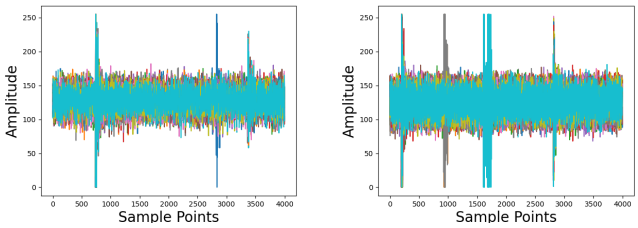
The Fixed Key-Fixed Tweak (FK-FT) dataset keeps both XTS keys and the sector index i fixed, resulting in a fixed tweak T_0, while the plaintext P is randomized. This dataset serves as the common attack set for all three experimental settings. In the Fixed Key-Random Tweak (FK-RT) dataset, K_1 is fixed, whereas K_2 and i are randomized, resulting in a randomized tweak. The Random Key-Fixed Tweak (RK-FT) dataset randomizes K_1 while keeping K_2 and i fixed, and therefore maintains a fixed tweak. Finally, the Random Key-Random Tweak (RK-RT) dataset randomizes both keys and the sector index, resulting in a randomized tweak. In all four datasets, the plaintext P is uniformly randomized.

We construct three attack settings by varying the conditions and number of traces used during profiling, while keeping FK-FT as a common attack dataset. In the Fixed Key, Random Tweak setting, 20 000 traces from FK-RT are used for profiling and 10 000 traces from FK-FT are used for the attack. In the Random Key, Fixed Tweak setting, 100 000 traces from RK-FT are used for profiling and 60 000 traces from FK-FT are used for the attack. Finally, in the Random Key, Random Tweak setting, 100 000 traces from RK-RT are used for profiling and 60 000 traces from FK-FT are used for the attack. These settings progressively introduce variability in the profiling key and tweak, enabling the attack to be evaluated under different profiling conditions while maintaining a common attack set for consistent comparison. To highlight differences in trace distributions, Figure 8 presents the RK-RT profiling traces alongside the FK-FT attack traces. The two sets exhibit distinct distributions, with peaks at different locations. Nevertheless, we will show later that the profiled DL-BSCA can rank the correct key among the top 10 candidates.

VI. Evaluation Results

We evaluated the proposed attack under three scenarios of increasing difficulty. We first consider the most favorable setting for the attacker, where the key is fixed across all traces, but the tweak could be random (FK-RT). This preliminary experiment is used to establish the feasibility of the attack methodology and to characterize the exploitable leakage of the underlying implementation.

After establishing the baseline case, we consider a scenario in which the attacker has partial knowledge of the tweak. Specifically, we assume that the attacker can identify the



(a) Profiling Traces (RK-RT). (b) Attack Traces (FK-FT).

Fig. 8: XTS-AES traces used for the RK-RT experiments.

memory address associated with the unencrypted tweak key. Under this setting, the key is randomized across traces during training to reduce potential bias arising from a fixed key. A fixed key could allow the model to learn key-dependent patterns rather than leakage characteristics associated with the targeted intermediate. Randomizing the key therefore provides a more representative evaluation of the model's ability to learn the underlying leakage characteristics.

We then consider a more challenging scenario in which the attacker has no prior knowledge of either the key or the tweak. Accordingly, both the key and the tweak are randomized across traces during the training phase. This setting reflects a more restrictive threat model and evaluates the model's ability to learn the targeted leakage characteristics without relying on knowledge of the key or tweak.

A. FK-RT Experiment (Settings 1)

As shown in Figure 7 (Settings 1), K_1 is fixed across all traces, while the sector value i is randomized. We use 20k traces for profiling and evaluate the attack using 10k attack traces. The resulting NTGE values are reported in Table IV, which gives the number of attack traces required to recover each of the 16 key bytes, i.e., to reach a guessing entropy of zero. The corresponding guessing entropy (GE) curves are shown in Figure 9.

All 16 key bytes are successfully recovered within the available attack traces, with the maximum NTGE being 8081 traces. This demonstrates that the unprotected XTS-AES implementation on the ESP32-C3 exposes exploitable EM side-channel leakage under this threat model. However, the attack complexity varies substantially across the key bytes. The NTGE ranges from only 14 traces for byte 4 to 8081 traces for byte 10, with a median of 765 traces. Excluding byte 10 as an outlier, the mean NTGE is approximately 873 traces.

Since complete key recovery requires all 16 bytes to be successfully recovered, the overall attack cost is determined by the slowest-converging byte. In this case, byte 10 requires 8081 traces and therefore determines the practical attack cost. In contrast, bytes 0, 4, 8, and 15 converge within approximately 250 traces, whereas bytes 6, 7, 10, and 11 require more than 1700 traces. This variation indicates that the magnitude and learnability of the exploitable leakage differ considerably across the key-byte positions.

TABLE IV: NTGE for each key byte in the FK-RT scenario.

Byte	NTGE	Byte	NTGE	Byte	NTGE	Byte	NTGE
0	237	4	**14**	8	243	12	391
1	804	5	1570	9	1526	13	726
2	575	6	1887	10	**8081**	14	904
3	318	7	1774	11	1914	15	208

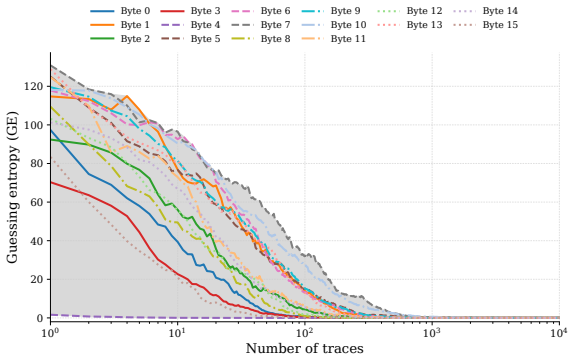


Fig. 9: Guessing entropy per key byte in the FK-RT scenario.

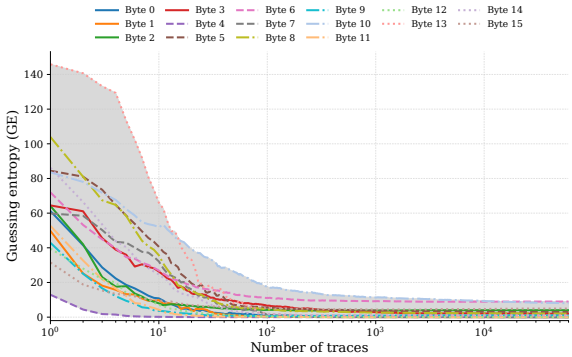


Fig. 10: Guessing entropy per key byte in the RK-FT Experiment.

TABLE V: NTGE (GE) for each key byte in the RK-FT experiment.

Byte	NTGE (GE)	Byte	NTGE (GE)	Byte	NTGE (GE)	Byte	NTGE (GE)
0	28618	4	22	8	(2.0)	12	1063
1	982	5	(3.0)	9	(1.0)	13	(1.0)
2	(4.0)	6	(9.0)	10	(8.05)	14	(1.0)
3	(2.0)	7	782	11	312	15	(4.9)

B. RK-FT Experiment (Settings 2)

We next consider Setting 2 in Figure 7, where K_1 is randomized while the sector value i remains fixed. We use 100k traces for profiling and 60k traces for the attack. Table V reports the NTGE for each of the 16 key bytes, while the corresponding GE curves are shown in Figure 10.

In this setting, the correct key is not ranked first for all bytes, even after exhausting the 60k attack traces. For such cases, we report the minimum GE achieved over the complete set of attack traces rather than an NTGE value. The values shown in parentheses in Table VI indicate these non-zero GE values, where the correct key does not reach rank one.

The results demonstrate the increased difficulty of key recovery when the key is randomized during profiling. Of

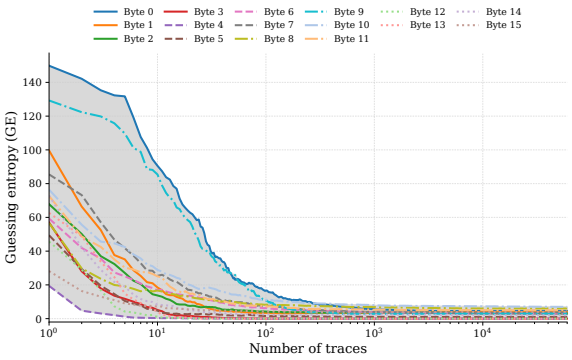


Fig. 11: Guessing entropy per key byte in the RK-RT Experiment.

TABLE VI: NTGE (GE) for each key byte in the RK-RT experiment.

Byte	NTGE (GE)	Byte	NTGE (GE)	Byte	NTGE (GE)	Byte	NTGE (GE)
0	(4.6)	4	59	8	(6.0)	12	59
1	(4.0)	5	(1.0)	9	(3.0)	13	41
2	(3.0)	6	(3.0)	10	(7.0)	14	(2.0)
3	430	7	(3.8)	11	(5.0)	15	(4.0)

TABLE VII: NTGE (GE) for each key byte in the RK-RT scenario for $K_1 \oplus T_0$ recovery.

Byte	NTGE (GE)	Byte	NTGE (GE)	Byte	NTGE (GE)	Byte	NTGE (GE)
0	(1.0)	4	(1.0)	8	(1.0)	12	(5.2)
1	(6.0)	5	(2.0)	9	(2.0)	13	(7.0)
2	(3.0)	6	(6.0)	10	1928	14	(9.1)
3	16	7	(2.0)	11	(1.0)	15	(8.0)

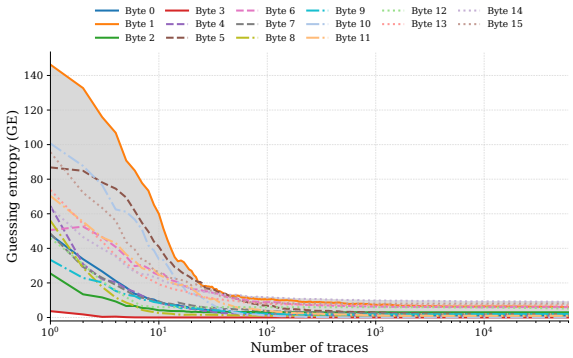


Fig. 12: Guessing entropy per key byte in the RK-RT scenario for $(K_1 \oplus T_0)$ recovery.

the 16 key bytes, only 6 are recovered directly, with the correct key ranked first. Nevertheless, the correct key remains among the top-5 candidates for 14 bytes and among the top-10 candidates for all 16 bytes. Thus, although complete key recovery is not achieved within the available attack traces, the attack substantially narrows the key space for every byte.

Compared with the FK-RT setting, this experiment represents a more challenging threat model in which the adversary must train across varying keys while the sector value remains fixed. The degradation in GE performance reflects the additional difficulty introduced by key randomization. Despite this increased difficulty, the correct value for every key byte remains within the top-10 candidates, indicating that the underlying EM leakage remains exploitable under this setting.

C. RK-RT Experiment (Settings 3)

We finally consider Setting 3 in Figure 7, where both K_1 and the sector value i are randomized. As in the previous experiment, we use 60 000 traces for attack evaluation. Table VI reports the NTGE for each of the 16 key bytes, while the corresponding guessing entropy curves are shown in Figure 11.

Similar to previous setting, Table VI denote the performance for each bytes under the RK-RT Experiment.

The results show a further degradation in key-recovery performance compared with the previous settings. Only 4 of the 16 key bytes are directly recovered, i.e., the correct key byte reaches rank one. Nevertheless, the correct key byte is ranked within the top-5 candidates for 14 bytes, while all 16 key bytes remain within the top-10 candidates. Thus, although complete key recovery is not achieved within the available attack traces, the attack consistently reduces the candidate space for every key byte.

This setting represents the most challenging and practically realistic threat model considered in our evaluation, as the adversary has no prior knowledge of either the secret key or the sector-dependent tweak. Consequently, the profiling phase must account for variations in both K_1 and i, requiring the models to learn leakage characteristics across the corresponding range of secret values. Despite this increased variability and the resulting degradation in recovery performance, the correct value remains within the top-10 candidates for every key byte, demonstrating that exploitable EM leakage persists even under this challenging setting.

D. Recovering Tweak after recovering K_1

After recovering the secret key K_1, we next consider recovering the secret tweak T_0. We adopt the same Profiled DL-BSCA framework described in Section IV, but modify the attack target to account for the recovered key. As described in Section IV-C, the targeted intermediate depends on the XOR combination of the plaintext, tweak, and key. Therefore, when K_1 is known, the same leakage can be exploited to recover the remaining unknown tweak component.

In particular, instead of directly targeting individual bytes of K_1, we construct hypotheses for the combined value $K_1^b \oplus T_0^b$ for each byte b. The previously collected traces can be reused for this purpose, since no additional measurements are required. For each candidate value, the corresponding intermediate leakage hypothesis is evaluated using the trained DNN model, and the candidates are ranked according to their accumulated likelihood. This produces a guessing-entropy curve for each byte of the $K_1 \oplus T_0$ value. Once the correct value of $K_1^b \oplus T_0^b$ is identified, the corresponding tweak byte can be directly obtained by XORing it with the recovered key byte K_1^b. This process is repeated independently for all 16 bytes to reconstruct the complete tweak T_0.

Table VII reports the number of traces required to reach guessing entropy zero (NTGE) for each of the 16 tweak bytes, while the corresponding guessing entropy curves are shown in Figure 12. As in the key-recovery experiments, we reuse

TABLE VIII: NTGE (GE) for each key byte in the RK-RT experiment: Random Search + BOHB.

Byte	NTGE (GE)	Byte	NTGE (GE)	Byte	NTGE (GE)	Byte	NTGE (GE)
0	4	4	59	8	(2.0)	12	59
1	(4.0)	5	(1.0)	9	(3.0)	13	41
2	1031	6	(3.0)	10	(1.4)	14	(2.0)
3	430	7	(3.8)	11	(5.0)	15	(0.5)

TABLE IX: NTGE (GE) for each key byte in the RK-RT scenario for $K_1 \oplus T_0$ recovery: Random Search + BOHB.

Byte	NTGE (GE)	Byte	NTGE (GE)	Byte	NTGE (GE)	Byte	NTGE (GE)
0	(1.0)	4	(1.0)	8	(1.0)	12	(5.2)
1	(6.0)	5	(2.0)	9	(2.0)	13	(4.0)
2	(3.0)	6	26916	10	1928	14	(0.9)
3	16	7	(2.0)	11	(1.0)	15	(2.6)

the previously collected 60,000 attack traces, requiring no additional trace acquisition for the tweak-recovery stage.

The results show that recovering the tweak is more challenging than recovering K_1 under the same threat model. Only 2 of the 16 tweak bytes are directly recovered, i.e., the correct value reaches rank one within the available traces. Nevertheless, the correct tweak value is ranked within the top-5 candidates for 10 bytes, while all 16 tweak bytes remain within the top-10 candidates. These results demonstrate that the traces collected for key recovery can also be effectively reused to extract information about the secret tweak, despite the additional uncertainty introduced by the randomized key and sector value.

VII. Discussion

A. Improving the Attack Using Bayesian Optimization HyperBand (BOHB)

Yap *et al.* [26] proposed a multifidelity optimization method, Bayesian Optimization HyperBand (BOHB), which combines Bayesian Optimization and HyperBand to efficiently search for optimal hyperparameters in deep learning models. BOHB efficiently explores the hyperparameter space by allocating greater computational resources to promising configurations while terminating poorly performing configurations at an early stage. This enables a more efficient search for high-performing configurations compared with evaluating all configurations for the same computational budget.

In the previous experiments, we observed that the recovery performance varied considerably across key bytes, with some bytes achieving a non-zero guessing entropy even after exhausting the available attack traces. We therefore investigated whether improved hyperparameter selection could enhance the recovery performance for these challenging bytes. To this end, we applied BOHB to the DNN hyperparameter search and evaluated its effect on both K_1 recovery and $K_1 \oplus T_0$ recovery in the RK-RT setting.

The resulting key-recovery performance is reported in Table VIII, while the corresponding results for $K_1 \oplus T_0$ recovery are shown in Table IX. Compared with the results obtained using the original random-search configuration, BOHB improves the recovery performance for several of the previously challenging bytes. In particular, some bytes that previously remained at higher guessing entropy can be recovered or brought to substantially lower GE values after hyperparameter optimization.

B. Comparison with Existing Works

Table X compares our key-recovery results with the existing results reported by Krüger *et al.* [9], which is the most recent work that breaks software XTS-AES. The comparison should be interpreted in the context of the differences in experimental methodology and threat assumptions. In [9], the attacks target the first AES round and exploit SubBytes or T-Table implementation with the traces segmented around the relevant software operations. Their analysis is performed on software-based AES implementations and relies on CPA in a non-profiling setting.

TABLE X: Performance Comparison against Existing Work

Related Work	# Traces	Target Implementation	Leakage Function	No. of Key Bytes Recovered $O(12)^*$
TinyAES [9]	94,857	Software	SubBytes	5
OpenSSLAES [9]	84,601	Software	SubBytes	13
OpenSSLAES [9]	84,601	Software	T-Table	13
This Work (Random)	60,000	Hardware	SubBytes	16
This Work (Random + BOHB)	60,000	Hardware	SubBytes	16

* The rank below 12 is considered for comparison with [9] results.

The authors note that knowledge of either the plaintext or ciphertext is required for the attack. In contrast, our attack targets the hardware AES implementation integrated into the ESP32-C3 and uses EM traces without relying on operation-level segmentation.

Under these differing conditions, our approach recovers the complete key with fewer attack traces in the evaluated settings as shown in Table X. More importantly, the proposed methodology extends beyond key recovery: the same traces can subsequently be reused to recover the $K_1 \oplus T_0$ value and derive the corresponding tweak after K_1 recovery. In contrast, recovery of the tweak key is identified as future work in [9]. These differences highlight that our evaluation addresses a distinct attack setting involving a hardware AES implementation, EM leakage, randomized secrets, and subsequent tweak recovery, rather than representing a direct reproduction of the experimental conditions of prior software-based attacks.

C. Implications for Flash Encryption

Although the ESP32-C3 supports XTS-AES for flash encryption, directly evaluating the side-channel leakage of the flash-encryption path introduces significant practical constraints. In particular, obtaining a new ciphertext value requires rewriting the corresponding flash contents before the device performs the encryption operation. Each new input requires an additional flash write cycle and, in typical experimental settings, a subsequent device reboot before the corresponding power or EM trace can be acquired.

Table XI shows the time and memory required data acquisition for a commercial ESP32-C3 flash encryption. For a single reboot, it requires 5 MB of memory per trace, and needs around 9 seconds for trace acquisition. While a relatively small experiment involving 10 000 traces already corresponds to approximately 50 GB of memory, collecting one million

traces would require approximately 5 TB of memory. More importantly, repeated erase/write operations are constrained by the endurance of the physical flash blocks, which is typically on the order of 10^5 erase/write cycles. Thus, scaling the experiment to the trace volumes commonly required for deep-learning-based side-channel analysis would place a substantial burden on the flash memory and may eventually affect its reliability or usability.

TABLE XI: Overhead for flash-encryption trace acquisition.

# Traces	Memory	Time
1	5 MB	9 s
10K	50 GB	25 h
100K	500 GB	10.4 days
1M	5 TB	104.2 days

The trace-acquisition time further amplifies this limitation. For our experimental setup, acquiring a single trace requires ≈9 seconds. Therefore, collecting one million traces would require ≈ 104.2 days, even before accounting for the additional time required for flash rewriting, device reboot, synchronization etc.

For these reasons, the present work focuses on the XTS-AES execution path rather than the flash-encryption workflow. The ability to recover K_1 and the corresponding sector tweak T_0 has direct implications for the security of XTS-AES-based flash encryption. Nevertheless, the demonstrated recovery of K_1 and a sector-specific T_0 establishes that the side-channel leakage exposes cryptographic information that can directly undermine the confidentiality of protected flash data. For example, for a 4 MB onboard flash, the total storage capacity is $4{,}194{,}304$ bytes. Assuming a 4 KB flash-sector size, this corresponds to 1024 sectors. Since the XTS-AES key is shared across the encrypted flash region, recovering the key from a single sector can potentially compromise the encryption of the remaining sectors. However, the tweak is location-dependent and must be determined or reconstructed for each relevant flash address.

D. Limitations of Profiled DL-BSCA

A key limitation of the proposed Profiled DL-BSCA methodology is the sensitivity of hyperparameter selection. In this work, we primarily rely on random search to identify suitable hyperparameter configurations for the deep learning models. However, our experiments show that the hyperparameter space is relatively large and that only a subset of configurations leads to effective models. Consequently, finding suitable configurations can require substantial computational effort, particularly when the leakage characteristics vary across key bytes.

Beyond the previously investigated BOHB approach, several methods have been proposed to enhance hyperparameter optimization for DL-SCA [27], [28], [29]. However, these approaches are computationally expensive and time-consuming, and are therefore left for future work. As such, an important direction to investigate is to develop efficient hyperparameter optimization strategies specifically tailored to Profiled DL-BSCA. In addition to reducing the computational cost of model selection, future studies could investigate how to automatically identify suitable model architectures and hyperparameter configurations based on the characteristics of the underlying leakage.

E. Countermeasures

The results of our attack also provide insights into potential countermeasures for protecting XTS-AES implementations against side-channel attacks. A hiding-based countermeasure, such as introducing additional noise or applying a shuffling strategy, could be considered to reduce the exploitable leakage of the AES computation. However, such countermeasures generally increase the number of traces required for a successful attack rather than eliminating the underlying leakage, and their effectiveness may therefore diminish as more traces are collected. Masking the tweak alone, however, is unlikely to provide sufficient protection. In the considered hardware implementation, the tweak can be masked during its generation but must ultimately be unmasked before being processed by the hardware AES engine. Our attack directly targets the AES computation to recover K_1, and therefore does not rely on directly observing the unmasked tweak during its generation. Moreover, the tweak itself can subsequently be recovered by exploiting the leakage from the AES computation, even after K_1 has been recovered. Consequently, protecting only the tweak-generation stage does not address the leakage exploited by our attack. Effective protection should therefore focus on the AES computation itself, for example, potentially through masking or incorporating appropriate side-channel countermeasures into the hardware AES implementation.

VIII. Conclusion

This work presents a blind side-channel analysis methodology for hardware-accelerated XTS-AES that operates directly on complete power traces without requiring operation-level segmentation or prior knowledge of cryptographic execution boundaries. The proposed approach overcomes the temporal assumptions commonly made by existing attacks and enables key recovery from coarse-grained whole-trace measurements.

We demonstrated the practical effectiveness of the methodology against the hardware XTS-AES implementation of a commercial ESP32-C3 microcontroller. The recovered AES key was further leveraged to reconstruct the sector-specific XTS tweak, thereby eliminating the need to independently recover the second XTS key. This ultimately enables recovery of encrypted flash contents from the target device.

Our findings highlight that hardware acceleration and the absence of explicit software execution boundaries do not inherently prevent side-channel exploitation. In particular, the results demonstrate the effectiveness of whole-trace, implementation-agnostic DL-SCA against hardware cryptographic implementations in embedded devices. These observations motivate the inclusion of such attacks in the security evaluation of hardware-assisted flash-encryption mechanisms, especially when assessing the confidentiality of protected data on commercial embedded platforms.

Generative AI Usage

The authors used generative artificial intelligence (AI) tools to assist with language editing and proofreading during the preparation of this manuscript.

REFERENCES

[1] "IEEE standard for cryptographic protection of data on block-oriented storage devices," IEEE, Tech. Rep. P1619, 2007.

[2] M. Dworkin, "Recommendation for block cipher modes of operation: The XTS-AES mode for confidentiality on storage devices," National Institute of Standards and Technology, Tech. Rep. SP 800-38E, 2010.

[3] P. Rogaway, "Nonce-Based Symmetric Encryption," in *Fast Software Encryption (FSE)*, 2004, pp. 348–358.

[4] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Advances in Cryptology – CRYPTO*, 1996, pp. 104–113.

[5] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology – CRYPTO*, 1999, pp. 388–397.

[6] C. Luo, Y. Fei, and A. A. Ding, "Side-channel power analysis of xts-aes," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, pp. 1330–1335.

[7] C. Luo, Y. Fei, A. A. Ding, and P. Closas, "Comprehensive Side-Channel Power Analysis of XTS-AES," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, pp. 2191–2200, 2019.

[8] T. Unterluggauer and S. Mangard, "Exploiting the Physical Disparity: Side-Channel Attacks on Memory Encryption," in *Constructive Side-Channel Analysis and Secure Design – COSADE*, 2016.

[9] P. Krüger, S. Wildermann, and J. Teich, "Breaking Confidentiality of XTS-AES Encrypted Data at Rest on Microprocessors Using Electromagnetic Side-Channel Attacks," in *IEEE International Symposium on Hardware Oriented Security and Trust – HOST*, 2025, pp. 450–461.

[10] Krüger, Paul and Wildermann, Stefan and Teich, Jürgen, "CRESTS: Chronology-based Reconstruction for Side-Channel Trace Segmentation for XTS-AES on Complex Targets," in *Proceedings of the 17th European Workshop on Systems Security*, ser. EuroSec '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 37–43. [Online]. Available: https://doi.org/10.1145/3642974.3652286

[11] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[12] A. Ito, R. Ueno, and N. Homma, "Deep Learning Side-Channel Attack against Hardware Implementations of AES," *Microprocessors and Microsystems*, 2021.

[13] K. Kamalakshan, D. Jap, S. Bhasin, and K. M. Abdellatif, "Em-vision: Evaluating security features in commercial risc-v against electromagnetic side-channel," in *2026 Asia-Pacific International Symposium and Exhibition on Electromagnetic Compatibility (APEMC)*. IEEE, 2026, pp. 1–4.

[14] K. Courdesses, "Breaking the flash encryption feature of Espressif's parts," https://courk.cc/breaking-flash-encryption-of-espressif-parts, 2024.

[15] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *Cryptographic Hardware and Embedded Systems – CHES 2004*, ser. LNCS, vol. 3156. Springer, 2004, pp. 16–29.

[16] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, "Mutual Information Analysis: A Generic Side-Channel Distinguisher," in *Cryptographic Hardware and Embedded Systems – CHES 2008*. Springer, 2008, pp. 426–442.

[17] K. Schramm, T. J. Wollinger, and C. Paar, "A New Class of Collision Attacks and Its Application to DES," in *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers*, ser. Lecture Notes in Computer Science, T. Johansson, Ed., vol. 2887. Springer, 2003, pp. 206–222. [Online]. Available: https://doi.org/10.1007/978-3-540-39887-5_16

[18] L. Wu, G. Perin, and S. Picek, "Weakly Profiling Side-channel Analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 3, p. 707–730, Nov. 2024. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/11901

[19] I. Savu, M. Krcek, G. Perin, L. Wu, and S. Picek, "The Need for MORE: Unsupervised Side-Channel Analysis with Single Network Training and Multi-output Regression," in *Constructive Side-Channel Analysis and Secure Design - 15th International Workshop, COSADE 2024, Gardanne, France, April 9-10, 2024, Proceedings*, ser. Lecture Notes in Computer Science, R. Wacquez and N. Homma, Eds., vol. 14595. Springer, 2024, pp. 113–132. [Online]. Available: https://doi.org/10.1007/978-3-031-57543-3_7

[20] O. Choudary and M. G. Kuhn, "Efficient Template Attacks," in *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, ser. Lecture Notes in Computer Science, A. Francillon and P. Rohatgi, Eds., vol. 8419. Springer, 2013, pp. 253–270. [Online]. Available: https://doi.org/10.1007/978-3-319-08302-5_17

[21] Y. Linge, C. Dumas, and S. Lambert-Lacroix, "Using the Joint Distributions of a Cryptographic Function in Side Channel Analysis," Cryptology ePrint Archive, Paper 2013/859, 2013. [Online]. Available: https://eprint.iacr.org/2013/859

[22] C. Clavier and L. Reynaud, "Improved Blind Side-Channel Analysis by Exploitation of Joint Distributions of Leakages," in *CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, W. Fischer and N. Homma, Eds.

[23] C. Clavier, L. Reynaud, and A. Wurcker, "Quadrivariate Improved Blind Side-Channel Analysis on Boolean Masked AES," in *Constructive Side-Channel Analysis and Secure Design*, J. Fan and B. Gierlichs, Eds. Cham: Springer International Publishing, 2018, pp. 153–167.

[24] A. Rezaeezade, T. Yap, D. Jap, S. Bhasin, and S. Picek, "Breaking the blindfold: Deep learning-based blind side-channel analysis," in *34th USENIX Security Symposium (USENIX Security 25)*. Washington, D.C.: USENIX Association, 2025, pp. 5777–5796. [Online]. Available: https://www.usenix.org/conference/usenixsecurity25/presentation/rezaeezade

[25] E. Cagli, C. Dumas, and E. Prouff, "Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing," in *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, ser. Lecture Notes in Computer Science, W. Fischer and N. Homma, Eds., vol. 10529. Springer, 2017, pp. 45–68.

[26] T. Yap, S. Bhasin, and L. Weissbart, "Train wisely: Multifidelity bayesian optimization hyperparameter tuning in deep learning-based side-channel analysis," in *International Conference on Selected Areas in Cryptography*. Springer, 2024, pp. 294–315.

[27] R. Y. Acharya, F. Ganji, and D. Forte, "Information theory-based evolution of neural networks for side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022.

[28] J. Rijsdijk, L. Wu, G. Perin, and S. Picek, "Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 677–707, 2021.

[29] L. Wu, G. Perin, and S. Picek, "I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis," *IEEE Transactions on Emerging Topics in Computing*, vol. 12, no. 2, pp. 546–557, 2022.

APPENDIX

HYPERPARAMETER SEARCH

For all our experiments, we uses the Multi-Layer Perceptron (MLP) architecture for the neural network. The hyperparameter search space is given in Table XII. Additionally, for simplicity, we set the pooling stride equal to the pooling size. We train the MLPs using categorical cross entropy loss. We train the MLP with 100 epochs.

TABLE XII: Hyperparameter search space.

Hyperparameter	Options
Number of Dense Layers	1 to 8 in a step of 1
Neurons per layer	$10, 20, 50, 100, 200, 300, 400, 500$
Batch size	100 to 1000 in a step of 100
Activation function	$ReLU, SeLU, ELU$ or $tanh$
Optimizer	Adam or RMSprop
Learning Rate	$1e-3, 1e-4, 5e-4, 1e^{-5}, 5e^{-5}$
Weight Initializer	Random Uniform or Glorot Uniform or He Uniform