BACHELOR'S THESIS COMPUTING SCIENCE

RADBOUD UNIVERSITY NIJMEGEN

Deep Learning-based Side-Channel Attack: Mamba Approach

Author:
Beatrise Bērtule
s1105119

Daily Supervisor:
Lizzy Grootjen

First Supervisor/Assessor:
Prof. Lejla Batina

Second Assessor:
Dr. Stjepan Picek

January 22, 2026

Abstract

Side Channel Attacks (SCAs) exploit unintentional physical leakages, such as power consumption or electromagnetic emissions, to recover secret cryptographic keys. Unlike attacks that target theoretical weaknesses within cryptographic algorithms, SCAs focus on the physical signals emitted while a device performs encryption or decryption. While traditional statistical approaches, such as Template Attacks (TA) for profiled SCAs, have proven effective, deep learning-based approaches show greater potential due to their ability to automatically learn complex patterns from raw power traces. Deep Learning-based Side-Channel Attacks (DL-SCAs) deploy neural network architectures for side-channel attacks. Architectures designed to process context-dependent data, such as multilayer perceptrons (MLP), convolution neural networks (CNNs) and hybrid attention-based architectures, have been successful. In our study, we propose a hybrid neural network architecture that deploys Mamba block architecture, a recently proposed selective state-space model, has not yet been widely explored for this purpose. Our model utilizes bidirectional encoders based on stacked Mamba blocks with residual connections which enables the model to capture temporal context in both - forward and backward - directions within power traces. We target the first round of the AES algorithm, concretely, we treat the output of the S-box function as the sensitive intermediate value and evaluate the proposed approach under unprotected and protected attack scenarios. Experimental results demonstrate that the Mamba-based model successfully recovers the secret key, achieving fast guessing entropy convergence and effective key recovery. While the proposed approach is competitive with a baseline MLP, our findings highlight the potential of selective state-space models for modeling temporal leakage patterns in side-channel attacks.

Contents

1 Introduction

Cryptographic algorithms such as the Advanced Encryption Standard (AES) are designed to guaranty strong confidentiality through encryption and decryption. However, the physical application of these algorithms can leak information through side channels - measurable physical phenomena such as power consumption [15], electromagnetic emissions [25], or timing variations [14]. While cryptanalysis target the theoretical weaknesses within the cryptographic algorithms, side-channel attacks (SCAs) exploit the physical side-channel leakages to recover secret information even when the algorithm is mathematically secure [26].

Classical SCA techniques are differential power analysis (DPA) [15] and template attacks (TA) [5]. Both methods rely on statistical techniques to correlate measured physical leakages with secret-dependent intermediate values processed by the device. In recent years, machine learning (ML), in particular, deep learning (DL), have advanced the state of art of side-channel attacks [24]. DL models can learn complex patterns from raw side-channel leakage data, reduce the need for feature extraction, and exhibit robustness against misalignment and even some side-channel attack countermeasures [24]. Commonly used architectures within the deep-learning based side-channel attack (DLSCA) domain are multilayer perceptrons (MLPs), convolution neural networks (CNNs) [13, 18, 20, 30], as well as hybrid attention-based neural networks [9, 17, 12].

1.1 Problem Statement

Despite the extensive exploration of conventional deep-learning architectures, sequence-oriented models specifically designed to capture contextual and long-range dependencies remain largely unexplored within the side-channel attack (SCA) domain. Power consumption traces are inherently sequential signals, and in practice the physical leakage caused by sensitive computations can be temporally spread across multiple time samples [18]. Consequently, deep learning architectures that explicitly model temporal context and long-range dependencies may offer advantages for side-channel analysis, as they can better capture distributed leakage patterns and correlate them with underlying cryptographic secrets.

Our study explores whether Mamba, a selective state-space model, can be effectively adapted for profiled side-channel attacks on the AES algorithm. Mamba extends classical state-space models with a selectivity mechanism that allows the model to selectively retain informative components of a sequence while suppressing noise.

1.2 Research Question

The central research question of our study is:

To what extent Mamba architecture's ability to model temporal context can enable an effective key recovery performance in deep learning–based side-channel attacks on AES algorithm?

In our study, we evaluate whether Mamba can learn meaningful patterns directly from the temporal leakage

and successfully correlate these patterns with secret cryptographic variables. To achieve this, we design a Mamba-based neural network. To effectively model temporal dependencies of power consumption leakage, the proposed model employs bidirectional encoders composed of Mamba blocks. The forward Mamba encoder models dependencies in the forward temporal direction, while the backward Mamba encoder captures dependencies in the reverse temporal direction. Then the model is trained to map side-channel leakage with a sensitive intermediate value, the output of the AES S-box, which is subsequently used to recover the secret key byte. The effectiveness of the proposed approach is evaluated with standard side-channel analysis metrics - key rank and guessing entropy [24]. Finally, we discuss the effectiveness of the proposed model based on the obtained evaluation results.

1.3 Contributions

The main contributions of this work to DL-SCA are as follows:

- We propose a hybrid neural network that utilizes bidirectional encoders based on stacked Mamba blocks with residual connections. This design aims to capture temporal context in both forward and backward directions of power traces.
- We explore the capacity of the Mamba architecture, specificaly, the selective-state space mechanism, to explicitly model temporal context within power consumption traces.
- We assess the proposed Mamba-based model on unprotected and protected AES side-channel leakage datasets, and benchmark its performance against an MLP model - a successful model proposed by prior research.

2 Preliminaries

This chapter provides the necessary background knowledge to understand the applied experimental methodology. It briefly introduces embedded devices and discusses the cryptographic algorithm used in this study, namely the Advanced Encryption Standard (AES). The chapter then explains how such algorithms can leak sensitive information through side-channel emissions and describes how deep learning methods can be applied to map side-channel data to secret cryptographic values.

2.1 Embedded Devices

Embedded devices are specialized computer systems designed to perform dedicated tasks within larger systems. Unlike general-purpose computers, embedded devices are optimized for efficiency, low power consumption, and cost-effectiveness [8]. Embedded microcontrollers such as ARM Coretx-M and ATMega frequently execute cryptographic algorithms. Hence, they handle highly sensitive data such as private keys and credentials [3, 18]. As most of these devices ar largely deployed by Internet of Things (IoT), they have become high-value target for attackers [11, 24]

2.2 Advanced Encryption Standard Algorithm

The Advanced Encryption Standard (AES) is a widely used symmetric block cipher standardized by NIST [19]. It operates on fixed-size 128-bit data blocks and supports key sizes of 128, 192, or 256 bits. AES deploys concepts of a substitution–permutation network (SPN) where several operations of substitution boxes (S-box) and permutations boxes (P-box) are applied to transform a plaintext into ciphertext.

2.2.1 Execution Flow of AES

AES execution consists of multiple transformation rounds. Algorithm 1 shows the general execution flow of AES. The algorithm takes a `State` which represents an intermediate value and `CipherKey` which corresponds to the cipher key. Before we apply the main transformation rounds, we apply `ExpandKey` operation to derive the round keys from the cipher key. Then `AddRoundKey` operator combines the state with the round key with the use of bitwise xor operator.

Algorithm 1: AES Encryption

Input: State, CipherKey
Output: Encrypted State

```
1 ExpandedKey ← KeyExpansion(CipherKey);
2 AddRoundKey(State, ExpandedKey[0]);
3 for i ← 1 to Nr do
4     Round(State, ExpandedKey[i]);
5 FinalRound(State, ExpandedKey[Nr]);
```

Each AES round consists of four operations: `SubBytes`, `ShiftRows`, `MixColumns`, and `AddRoundKey`. The number of rounds depends on the key size. Algorithm 2 illustrates the general execution flow of a single AES round. The algorithm takes the current state as input along with the corresponding round key, and the state is processed sequentially through the four operations. In this work, we focus in more detail only on the

`SubBytes` operation.

Algorithm 2: AES Round Transformation

```
Input: State, RoundKey
1 SubBytes(State);
2 ShiftRows(State);
3 MixColumns(State);
4 AddRoundKey(State, RoundKey);
```

The final round of AES differs slightly as it excludes the `MixColumns` operation. Algorithm 3 shows the execution flow of the final round.

Algorithm 3: AES Final Round Transformation

```
Input: State, RoundKey
1 SubBytes(State);
2 ShiftRows(State);
3 AddRoundKey(State, RoundKey);
```

These transformations ensure non-linearity, diffusion, and key dependency throughout the algorithm.

2.2.2 SubByte Transformation

The SubByte transformation is the first operation in each AES round. In this step, each byte $a_{i,j}$ of the state array is replaced with a corresponding SubByte $S(a_{i,j})$ using an 8-bit substitution box (S-Box). Table 2.1 visualizes the S-Box lookup table in hexadecimal representation. For example, byte 0x00 will be mapped to 0x52, and byte 0x52 will be mapped back to 0x00. Such transformation introduces non-linearity into the cipher. The AES S-Box is derived from the multiplicative inverse over the finite field $GF(2^8)$ which is known for its strong non-linear properties.

2.3 Side-Channels Attacks

While cryptanalysis target the mathematical foundations of a cryptographic algorithm, Side-Channel Attacks (SCA) exploit the physical implementation of them. These attacks rely on unintentional physical leakage that occurs during the execution of cryptographic algorithms. Given the observable part of side channel leakage such as power consumption [15], execution time [14], or electromagnetic emissions [25], the adversary can extract secret information as the physical characteristics of a hardware device are statistically correlated with the processed data [15, 26].

2.3.1 Side-Channel Data

Side-channel data are physical signals that reveal how a cryptographic device processes data internally. A vulnerability arises because the device's power consumption is not constant over time, but depends on the intermediate values processed by the device [26]. In particular, these intermediate values are often derived from secret information, such as cryptographic keys. For example, in an AES implementation, the output of the S-box depends on a known plaintext byte and an unknown secret key byte [19]. As a result, the corresponding power leakage is correlated with this secret-dependent value. With large number of side-channel traces, this dependence can be exploited, and an attacker can distinguish correct key hypotheses from the incorrect ones. This principle forms the basis of side-channel attacks, where physical measurements are analyzed to recover secret keys without directly breaking the underlying cryptographic algorithm.

	0x0	1x0	2x0	3x0	4x0	5x0	6x0	7x0	8x0	9x0	Ax0	Bx0	Cx0	Dx0	Ex0	Fx0
0x0	0x63	0x7C	0x77	0x7B	0xF2	0x6B	0x6F	0xC5	0x30	0x01	0x67	0x2B	0xFE	0xD7	0xAB	0x76
1x0	0xCA	0x82	0xC9	0x7D	0xFA	0x59	0x47	0xF0	0xAD	0xD4	0xA2	0xAF	0x9C	0xA4	0x72	0xC0
2x0	0xB7	0xFD	0x93	0x26	0x36	0x3F	0xF7	0xCC	0x34	0xA5	0xE5	0xF1	0x71	0xD8	0x31	0x15
3x0	0x04	0xC7	0x23	0xC3	0x18	0x96	0x05	0x9A	0x07	0x12	0x80	0xE2	0xEB	0x27	0xB2	0x75
4x0	0x09	0x83	0x2C	0x1A	0x1B	0x6E	0x5A	0xA0	0x52	0x3B	0xD6	0xB3	0x29	0xE3	0x2F	0x84
5x0	0x53	0xD1	0x00	0xED	0x20	0xFC	0xB1	0x5B	0x6A	0xCB	0xBE	0x39	0x4A	0x4C	0x58	0xCF
6x0	0xD0	0xEF	0xAA	0xFB	0x43	0x4D	0x33	0x85	0x45	0xF9	0x02	0x7F	0x50	0x3C	0x9F	0xA8
7x0	0x51	0xA3	0x40	0x8F	0x92	0x9D	0x38	0xF5	0xBC	0xB6	0xDA	0x21	0x10	0xFF	0xF3	0xD2
8x0	0xCD	0x0C	0x13	0xEC	0x5F	0x97	0x44	0x17	0xC4	0xA7	0x7E	0x3D	0x64	0x5D	0x19	0x73
9x0	0x60	0x81	0x4F	0xDC	0x22	0x2A	0x90	0x88	0x46	0xEE	0xB8	0x14	0xDE	0x5E	0x0B	0xDB
Ax0	0xE0	0x32	0x3A	0x0A	0x49	0x06	0x24	0x5C	0xC2	0xD3	0xAC	0x62	0x91	0x95	0xE4	0x79
Bx0	0xE7	0xC8	0x37	0x6D	0x8D	0xD5	0x4E	0xA9	0x6C	0x56	0xFB	0xEA	0x65	0x7A	0xAE	0x08
Cx0	0xBA	0x78	0x25	0x2E	0x1C	0xA6	0xB4	0xC6	0xE8	0xDD	0x74	0x1F	0x4B	0xBD	0x8B	0x8A
Dx0	0x70	0x3E	0xB5	0x66	0x48	0x03	0xF6	0x0E	0x61	0x35	0x57	0xB9	0x86	0xC1	0x1D	0x9E
Ex0	0xE1	0xF8	0x98	0x11	0x69	0xD9	0x8E	0x94	0x9B	0x1E	0x87	0xE9	0xCE	0x55	0x28	0xDF
Fx0	0x8C	0xA1	0x89	0x0D	0xBF	0xE6	0x42	0x68	0x41	0x99	0x2D	0x0F	0xB0	0x54	0xBB	0x16

Table 2.1: AES S-box lookup table.

2.3.2 Leakage Models

To effectively deploy side-channel attack techniques, a sensitive intermediate value is typically mapped to an estimated leakage value through a leakage model [3, 7, 20, 22, 30]. The actual physical leakage L produced by a device is commonly modeled as the sum of a deterministic component and random noise [7]. Equation (2.1) expresses the physical leakage mathematically where δ denotes the unknown deterministic leakage function, V_k corresponds to a sensitive intermediate value that depends on the secret key k, and B represents additive noise. Since the true leakage function δ is unknown to the attacker, a leakage model m is used as an approximation of the physical behavior [7]. A leakage model is therefore a mathematical function that predicts the physical leakage, such as power consumption or electromagnetic emissions, produced by a cryptographic device when processing a specific sensitive value.

$$L = \delta(V_k) + B \tag{2.1}$$

The main assumption of the proposed approach is that the leakage caused by a sensitive value is predictable enough for a simple model to capture the statistical correlation between the secret-dependent computation and the measured side-channel traces.

Commonly used leakage models include the Hamming Weight (HW) model and the Identity (ID) model.

- **Hamming Weight leakage model**.
  The HW model assumes that the physical leakage is proportional to the number of bits in the sensitive variable v that are set to 1. Equation (2.2) mathematically represents the HW leakage model function m_{HW} for an n-bit value $v = (b_{n-1}, \ldots, b_1, b_0)$ where $b_i \in \{0, 1\}$.

$$m_{HW}(v) = \sum_{i=0}^{n-1} b_i \tag{2.2}$$

- **Identity leakage model**.
  The Identity model is a more flexible approach that makes no assumptions about the linear relationship between bits. Instead, it treats each possible value of the sensitive variable as a distinct class with its own unique physical signature. Equation (2.3) mathematically expresses the ID leakage model $m_{ID}(v)$.

$$m_{ID}(v) = v \tag{2.3}$$

2.3.3 Side-Channel Attacks

Side-channel attacks can be categorized into non-profiling (direct) and profiling (two-stage) attacks scenarios [24]. In non-profiling attack scenarios, the attacker has no access to the target clone device. Consequently, the adversary runs statistical analysis on the side-channel data directly to correlate the side-channel leakage to a cryptographic secret, and obtain the best guess on the secret value [23]. In contrast, profiling attacks assume access to a clone device. In the first phase, the attacker constructs a model that characterizes the side-channel leakage behavior of the device. In the second phase, this model is used to exploit the leakage observed from the target device in order to recover the secret key [11].

To build the link between the data processed by a device and the physical signals emitted during that process, leakage models are applied. The application of leakage models for side-channel attacks usually follow four distinct steps [7]: (1) collect a set of N physical leakage traces L while the target devices processes known plaintexts or ciphertexts p_i, (2) for every possible key candidate k, calculate the sensitive intermediate value $v_{ik} = f(p_i, k)$, where where f is a known cryptographic primitive such as the AES S-box function, (3) apply a leakage model m to the derived intermediate values to obtain hypothetical leakage estimates, (4) use a statistical distinguisher, such as the Pearson correlation coefficient (ρ), to evaluate the relationship between the measured traces L and the estimates M_k. The predicted key k^* is the one that maximizes the relationship of Equation (2.4) where $M_k = m(v_k)$ stand for the hypothetical leakage estimates under a leakage model m.

$$k^* = \text{argmax}_k |\rho(L, M_k)| \tag{2.4}$$

While side-channel data provide access to physical measurements, attackers do not observe the sensitive intermediate values directly. Instead, they observe noisy physical signals that are dependent on these values. To relate the measured leakage to the underlying secret-dependent computation, attackers rely on leakage models that approximate how intermediate values are reflected in the physical domain.

Typically, for side-channel attacks on the AES encryption algorithm the sensitive variables are recovered in parts with divide-and-conquer approach [24] which makes the attack computationally more feasible. To recover the full key, the approach described above gets repeated for each sub-byte key until the full key is recovered. For research purposes, one key-byte recovery sufficient to show effectiveness of an attack [24].

2.4 Deep Neural Networks

Deep neural networks (DNN) map a signal features $N \subseteq \mathbb{F}$ to a class label $C \subseteq \mathbb{N}$. DNNs can be described with a function $f_\theta : \mathscr{X} \to \mathscr{Y}$ where $\mathscr{X} \subseteq \mathbb{R}^F$ and $\mathscr{Y} \subseteq \mathbb{R}^N$ [24, 21]. Equation (2.5) shows the function where the initial layer f_1 operates on the input $x \in \mathscr{X}$ and the final layer f_L produces the output $y \in \mathscr{Y}$. Each layer f_l is typically parameterized by a subset of the parameters θ.

$$f_\theta(x) = f_L \circ f_{L-1} \circ \cdots \circ f_1(x). \tag{2.5}$$

Each layer f_ℓ consists of a set of neurons that apply a linear transformation followed by a nonlinear activation function [3]. Equation (2.6) demonstrates such computation of the ℓ-th layer where $\mathbf{W}_\ell$ and $\mathbf{b}_\ell$ denote the trainable weight matrix and bias vector, respectively, and $\phi_\ell(\cdot)$ represents a nonlinear activation function.

$$f_\ell(x) = \phi_\ell(\mathbf{W}_\ell x + \mathbf{b}_\ell), \tag{2.6}$$

The output of the network prior to normalization is referred to as the logits. To obtain a probability distribution over the target classes, the softmax function $\sigma(\cdot)$ is applied to the logits [24]. The predicted class label for an input x is then given by Equation (2.7) where $|\mathscr{C}|$ denotes the number of target classes.

$$\hat{y} = \arg \max_{i \in \{1,\ldots,|\mathscr{C}|\}} \sigma\big(f_\theta(x)\big)_i \tag{2.7}$$

2.4.1 MAMBA Architecture

State Space Models

State Space Models (SSMs) are a class of sequence models that map one-dimensional input $x(t) \in \mathbb{R}$ to an output $y(t) \in \mathbb{R}$ through an N-dimensional latent state $h(t) \in \mathbb{R}^N$. The foundation of an SSM is a continuous time system defined by two primary equations - the state equation $h'(t)$ which describes how the latent state $h(t)$ evolves over time, and the output equation $y(t)$ which maps the hidden state to the final output. Equations (2.8) and (2.9) describes the state and the output equation respectively with parameters $\mathbf{A} \in \mathbb{R}^{N\times N}$, $\mathbf{B} \in \mathbb{R}^{N\times 1}$, and $\mathbf{C} \in \mathbb{R}^{1\times N}$.

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t) \tag{2.8}$$
$$y(t) = \mathbf{C}h(t) \tag{2.9}$$

To operate on discrete sequences, the continuous parameters $\mathbf{A}$ and $\mathbf{B}$ are transformed into discrete parameters $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ through a mathematical process called discretization [10]. Equation (2.10) expresses the Zero-Order Hold (ZOH) discretization technique where Δ represents the step size, and exp denotes the matrix exponential.

$$\overline{\mathbf{A}} = \exp(\Delta\mathbf{A}) \tag{2.10}$$
$$\overline{\mathbf{B}} = (\Delta\mathbf{A})^{-1}(\exp(\Delta\mathbf{A}) - \mathbf{I}) \cdot \Delta\mathbf{B} \tag{2.11}$$

Equation (2.12) described the new state space model with discretized parameters.

$$h_t = \overline{\mathbf{A}}h_{t-1} + \overline{\mathbf{B}}x_t \tag{2.12}$$
$$y_t = \mathbf{C}h_t \tag{2.13}$$

Traditional SSMs are fundamentally time invariant, which means that the systems matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ remain static across all token, independent of the input data. This leads to major problems of lack of content-awareness and fixed compression. Because the parameters are fixed, the model lacks a mechanism to selectively prioritize salient information or suppress irrelevant noise within a sequence. As a result the model compresses the entire sequence history into a latent state with uniform weights [10].

The Selection Mechanism

To address the limitations of traditional SSMs, selective state space models introduce input-dependent dynamics by allowing the parameters $\mathbf{B}$, $\mathbf{C}$, and Δ to vary over time [10]. This input dependence enables the model to selectively compress the sequence history, retaining relevant information while discarding noise.

Equation (2.18) defines the new parameterization of $\mathbf{B}$, $\mathbf{C}$, and Δ. Equations (2.14), (2.15) and (2.16) define the specifics of each parametrization function. At each time step t, the model generates distinct parameters $\mathbf{B}_t$ and $\mathbf{C}_t$ as functions of the current input token. The parameter $\mathbf{B}_t$ controls how much of the current input is written into the latent state, while $\mathbf{C}_t$ determines how much of the latent state contributes to the current output. In addition, the discretization step size Δ is made input-dependent. Mechanistically, Δ acts as a memory control parameter: a large value of Δ_t effectively resets the state, allowing the model to focus on the

current input, whereas a small Δ_t preserves the previous state and attenuates the influence of the current input.

Together, these mechanisms provide selective SSMs with content-aware memory updates that are absent in traditional time invariant models.

$$\mathbf{s}_B(x) = \text{Linear}_N(x) \tag{2.14}$$
$$\mathbf{s}_C(x) = \text{Linear}_N(x) \tag{2.15}$$
$$\mathbf{s}_\Delta(x) = \text{Broadcast}_D(\text{Linear}_1(x)) \tag{2.16}$$
$$\tau_\Delta = \text{softplus} \tag{2.17}$$

$$\mathbf{B}_t = \mathbf{s}_B(x_t), \quad \mathbf{C}_t = \mathbf{s}_C(x_t), \quad \Delta_t = \tau_\Delta(\mathbf{s}_\Delta(x_t)) \tag{2.18}$$

Together, these mechanisms provide selective SSMs with content-aware memory updates that are absent in traditional time invariant models.

Hardware-Aware Computation

By making the model time-varying, selective SSMs lose the ability to be computed as fast convolutions. Traditional state space models can be expressed as convolutions, which enables highly parallel training. However, input-dependent parameters break this equivalence and require a recurrent formulation. To maintain computational efficiency, Mamba employs a hardware-aware parallel algorithm based on three key techniques: kernel fusion, selective scan and recomputation [10].

Mamba Block

The Mamba block is a sequence modeling unit that serves as the fundamental building block of the Mamba architecture. It is designed to replace the standard "Attention + MLP" stack found in Transformer architectures with a single layer that scales linearly with sequence length [10]. Figure 2.1 visualizes the Mamba block.

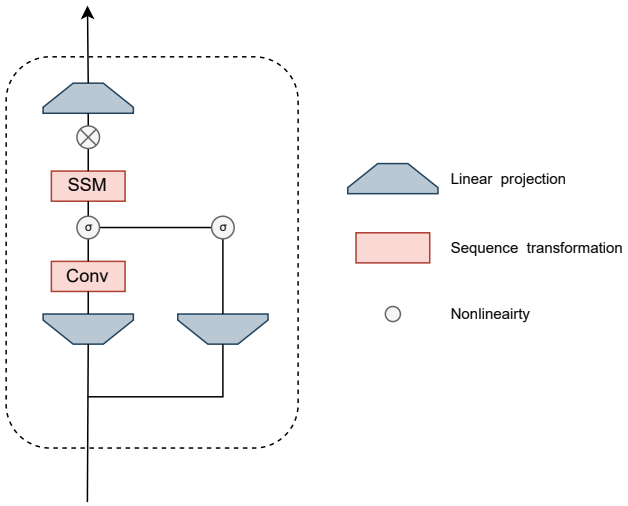

Figure 2.1: Mamba block.

The Mamba block expands the embedded sequence with dimension D to a higher dimensional space through a linear layer to create a higher dimensional hidden state. The expanded signal is the processed by a 1D convolution to capture local dependencies. Following a SiLU/Swish activation, the signal enters the

core Selective SSM (S6) layer where the model achieves context-awareness with the dynamically generated transition parameters (Δ, **B**, **C**). In a parallel branch, the expanded sequence is passed through a separate activation to act as a multiplicative gate, which is combined with the SSM output to modulate information flow. Finally, the gated result is projected back to the original dimension D. This architecture allows multiple Mamba blocks to be stacked with residual connections.

2.5 Deep-Learning based Side-Channel Attack

Figure 2.2 presents the general workflow of deep learning-based side-channel attack (DL-SCA) execution.

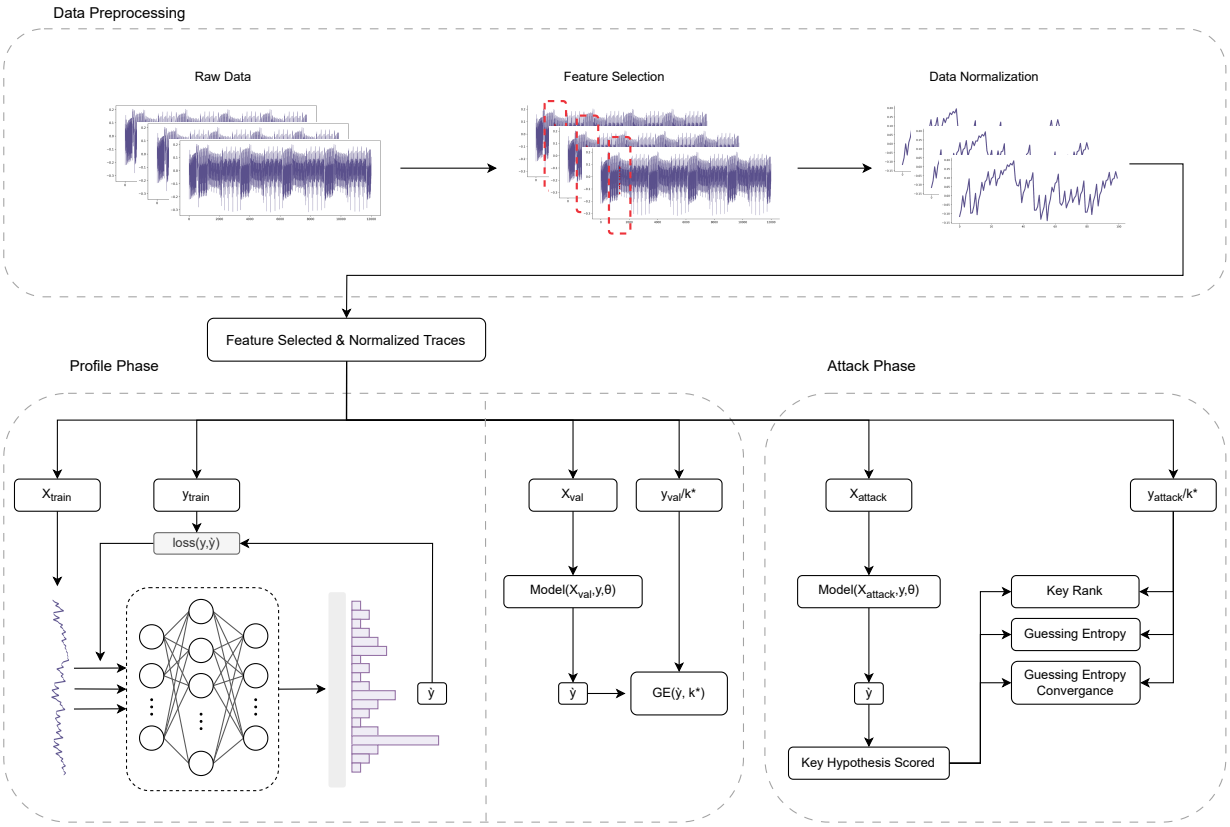

Figure 2.2: General workflow of DL-SCA execution.

2.5.1 Threat Model

The DL-SCA workflow shown in Figure 2.2 follows a profiled side-channel attack execution flow [24]. Profiled side-channel attacks are the most powerful class of side-channel attacks [3] with two-phase process: a profiling phase and an attack phase. The attacker is assumed to have access to a clone device - a device that is identical to the target device. This assumption allows the attacker to build a model of the clone device, and use that model to attack the target device to retrieve sensitive information.
Profiled deep-learning based side-channel attacks consist of two phases:

- **Profiling Phase**.
  In the profiling phase, the attacker collects a large number of side-channel traces from the clone device. For each trace, the attacker controls the plaintexts and knows the corresponding key. This enables the attacker to label each trace with the intermediate value of interest (the S-box output). With the labeled data the attacker can train a deep-learning model to learn patterns in the raw power measure that correspond to secret-dependent computations.

- **Attack Phase**.
  In the attack phase, the attacker uses the previously trained model to recover the secret key from the

target device. Unlike during profiling, the attacker now only observes traces from the real target device, for which the key is unknown. For each new trace, the model evaluates how likely the measured leakage corresponds to each possible value of the targeted intermediate variable (the S-box output). These likelihoods are then combined across multiple attack traces to form a key hypothesis score, typically by accumulating log-likelihoods. The correct key hypothesis gradually emerges as the one with the highest aggregated score. Once the most likely key value is identified for the targeted byte, the attacker repeats the process for the remaining key bytes until the full key is recovered. This phase requires far fewer traces than non-profiled attacks because the model already captures the leakage characteristics learned from the clone device.

2.5.2 Data Acquisition

We require side-channel leakage data to perform a profiled side-channel attack. In a profiled scenario, we assume that the attacker has full control of a clone device of the target device. This allows the attacker to collect leakage traces with known plaintexts and keys. Power consumption traces are typically captured with an oscilloscope or a purpose-built acquisition platform such as ChipWhisperer [3, 11, 18]. The captured traces are stored together with auxiliary information - most commonly plaintexts and keys — which is used to label and organize the collected measurements.

Apart from collection of your own measurements, researchers commonly use publicly available side-channel datasets [24]. These benchmark collections contain raw side-channel leakage traces with additional data such as plaintexts and ciphertexts, which makes the experiments reproducible and allows for direct comparison across studies. These dataset usually contain predefined profile/attack splits, target AES-128 (with or without side-channel countermeasures), and employ either fixed or randomized keys across traces. Among these resources, ASCAD dataset [3] is by far the most widely used benchmark to evaluate DL-SCA models.

2.5.3 Label Preparation

The most common place to attack the AES algorithm is the first round, right after the SubByte step, more specifically, the output of the S-box function [3, 9, 18, 20, 22]. For deep learning-based side channel attacks, each side-channel trace must be associated with a label derived from an assumed leakage model. Under the identity-based leakage model, the leakage target corresponds directly to the S-box output value. Equation (2.19) defines the label y_j for the j-th trace and the i-th target byte, where p_{ji} and k_{ji} represent the i-th byte of the j-th plaintext and key, respectively.

$$y_j = \mathrm{S-box}\left[\, p_{ji} \oplus k_{ji} \,\right] \tag{2.19}$$

2.5.4 Feature Selection

Although raw side-channel traces can be passed directly to a neural network, previous research has shown that such an approach can often be inefficient [20]. The dimensionality of the data can be extremely large, which often causes high computational complexity. Moreover, the traces contain large amounts of noise - points that do not hold much information - which can reduce the model efficiency [20] . To address this problem, feature selection is applied - only Points of Interest (POIs), i.e., time samples that carry the most information about the leaked secret, are selected for profiling and attack.

We can distinguish between three types of POI selection:

- **Refined Points of Interest (RPOI) Selection**.
  In RPOI selection, we choose features based on Signal to Noise Ratio (SNR) measure. The SNR at a given time sample t is defined as the ratio between the variance of the signal power $s(t)$ and the variance of the noise power $s(n)$. The SNR formula can be viewed in Equation (2.20).

$$\mathrm{SNR}(t) = \frac{\mathrm{Var}[s(t)]}{\mathrm{Var}[n(t)]} \tag{2.20}$$

The RPOI method selects Points of Interest (POIs) that correspond to the highest SNR values, as these points carry the strongest correlation with the processed secret. However, since only the peaks are retained, we disregard the broader computational context. This can be problematic as the leakage of a target operation may be context dependent and rely on prior computations.

- **Optimized Points of Interest (OPOI) Selection**.
  In OPOI selection, we choose an optimized window - a range of time sample points - from the full execution of the algorithm that contains the relevant leakage. The window is defined based on the main SNR peak: we include the peak itself along with its neighboring points to ensure that the selected region not only captures the strongest leakage but also preserves the surrounding computational context.
- **Non-Optimized Points of Interest (NOPOI) Selection**.
  In NOPOI selection, we don't choose POIs explicitly. Instead we use the full trace directly for profiling and attack. While this approach can result in high computational complexity, it eliminates the risk of excluding potentially relevant regions of the trace - a limitation that may occur with OPOI selection.

Equation (2.21) mathematically represents the optimized window that contains these peak SNR values where W_t denotes the the window centered at time stamp t, and s denotes the chosen window size.

$$W_t = \{ t \in \mathbb{Z} \mid t - s \leq t \leq t + s \} \tag{2.21}$$

2.5.5 Model Implementation

Once features are selected and labels are prepared, we can adapt and train a model to extract secret information. Neural network models are adapted so that they can process side-channel traces and output a prediction related to the secret key. Typically, models take a sequence of power or electromagnetic measurements, while the output corresponds to a probability distribution over key-dependent classes defined by a chosen leakage model. Then we train the model to map trace features with intermediate cryptographic values. Then these learned representations are used to rank key hypotheses based on their likelihood. Consequently, the model must be designed such that the input dimensionality matches the number of features (time samples), and the output layer produces a valid probability distribution over the target classes.

2.5.6 Attack Execution

The trained model predicts a probability distribution over the S-box output values P_j over the attack traces. Given that the plaintexts p_j are known for the attack traces, we can calculate the potential S-box output values under a particular key candidate k. As the AES algorithm is byte-oriented, it's common to attack one key byte at a time, denoted as the i-th byte. The target byte of a key contains 8 bits, which gives us a total of $2^8 = 256$ potential key byte values. For every key candidate byte value, we compute the hypothetical labels. Under the scenario that the key candidate byte value is the correct one, these labels are the S-box function output values. The computation of hypothetical labels for the i-th byte of the key candidate $h(k)$ can be viewed in Equation (2.22) where p_{ji} denotes the i-th byte of plaintext for all attack traces $j = 0, \ldots N-1$.

$$h(k_i) = \mathrm{S-box}\left[p_{ji} \oplus k_i\right] \tag{2.22}$$

To score a key candidate, we accumulate the model's probabilities[1] assigned to those hypothetical labels across all attack traces. The final vector contains the total (log-)likelihood probabilities for all key candidates, referred to as the *scores* of the key candidates. The computation of the key candidate scores $s(k_i)$ can be viewed in Equation (2.23) where $(P_j(h(k_i))$ denotes the model's probability assigned to the hypothetical label for trace j under key candidate byte k_i.

$$s(k_i) = \sum_{j=0}^{N} \log\big(P_j(h(k_i))\big) \tag{2.23}$$

[1] We convert to logarithmic space for numerical stability and add a small constant ($\varepsilon > 0$) to avoid undefined values.

We compute $s(k_i)$ for every key-byte candidate $k_i \in \{0,\ldots,255\}$. Once we sort these scores in descending order, we get a **key guessing vector** g where $g[0]$ is the most likely candidate, $g[1]$ the second most likely, and so on. The vector therefore contains the 256 key-byte candidates ordered by their (log-)likelihood of being the correct key byte. The computation of the key guessing vector g can be viewed in Equation (2.24) where argsort function sorts an array elements in order of decreasing values of their probabilities.

$$g = \text{argsort}\,|\{s(k_i)\}_{k_i=0}^{255}| \tag{2.24}$$

In a real life attack scenario, the attacker uses the key guessing vector g to recover secret information by brute-force trials. The attacker attempts to decrypt a captured cipher-text or verify a known plaintext for each candidate until the correct key is found. The number of decryption trials required equals the rank of the correct key. Even if the models top prediction is not correct, the key appears high in the rank and only a small number of brute-force trials is required.

2.5.7 Attack Evaluation

We evaluate how well the trained model performs on the attack traces with SCA-specific metrics and the known key [30]: key rank and guessing entropy (convergence).

- **Key Rank**.
  Key rank is the index (position) of the correct key within the key guessing vector. The rank represents how many keys the attacker has to brute-force to reach the correct key.

- **Guessing Entropy**.
  Guessing entropy (GE) is the average key rank across multiple realizations of the attack, each computed with a random subset of the attack traces. Guessing entropy therefore summarizes the expected effort required to recover the key. The computation of guessing entropy is defined in Equation (2.25) where $\text{rank}_j(g)$ is the key rank of the j-th trace and M is the total number of the random subset of the attack traces.
  $$GE = \frac{1}{M}\{\text{rank}_j(g)\}_{j=0}^{M} \tag{2.25}$$
  It is common to visualize guessing entropy convergence - how guessing entropy decreases as more attack traces are processed. This illustrates both the rate at which the attack becomes effective and the approximate rank the attack converges to. This metric is useful to evaluate the feasibility of the attack in real-world scenarios with limited data access.

We use SCA-specific evaluation metrics because basic classification metrics such as accuracy are often too binary - they only tell us whether the predicted label is correct or not. However, a model with lower accuracy can still be valuable to an attacker if the correct key appears high enough in the rank [22].

3 Related Work

Since 2016 [18], deep learning–based side-channel attacks have been extensively studied. Over the years, researchers have explored various directions within this field. This chapter focuses specifically on prior work related to hybrid architectural designs for deep learning models in the context of side-channel attacks.

3.1 Background

Deep learning–based side-channel attacks (DL-SCAs) have become an active research direction only within the past decade. In 2016, Maghrebi et al. conducted the first systematic study of DL techniques for SCA context [18]. Since then, the field has grown rapidly [24], driven by the ability of deep learning models to automatically handle noisy measurements [13], operate directly on raw power traces [20], and remain effective even against certain side-channel countermeasures [3]. Over time, researchers have explored the DL-SCA problem from several complementary angles [24]. These angles cover (1) DL architecture choices, where several DL architecture types - CNNs and MLPs [1, 13] along with some hybrid attention-based models [9, 12, 17] - have been studied; (2) data acquisition and design, where several publicly available dataset have been proposed to enable consistent evaluation across works [3]; (3) feature engineering, where feature selection scenarios and dimensionality reduction methods have been explored to extract the side channel leakage points [20, 21]; (4) model hyperparameter optimization, where different model hyperparameter affect the attack performance [2, 29]; and (5) attack evaluations metrics where several measures have been explored to assess the effectiveness of an attack [22].

3.2 DL Architecture Choice and Design

Among the various architectural approaches explored, CNNs and MLPs have been the most widely adapted. Both architectures have been successfully applied to attacks on various AES implementations in profiled and non-profiled scenarios [1, 13, 20, 27]. In previous studies, vanilla variants of these models - standard CNNs and MLPs - are often used to explore general optimization properties of DL-SCA. These baseline architectures are typically chosen to assess the performance of feature selection techniques [20] and to compare different hyperparameter search approaches [2, 29]. Their architectural simplicity allows researchers to recognize the impact of individual optimization choices.

3.3 Attention for DL-SCA

Recently, researchers have shifted their focus on more complex DL architecture designs - hybrid networks that combine different complementary neural components. These models typically aim to exploit the ability of CNNs to detect local patterns within power traces together with other mechanisms that are able to model broader temporal dependencies within the trace. In practice, such architectures frequently embed attention mechanisms to enhance the model's ability to focus on informative segments of the trace.

Several models have been proposed that utilize modified attention mechanisms after convolution to capture long-range context, enhance CNN-extracted features, and reduce overall noise. Attention mechanisms and their variants [28] have been widely adopted in fields such as Natural Language Processing (NLP) and Automatic Speech Recognition (ASR). In the context of side-channel attacks, attention has been used to better extract relevant side-channel leakage features. Representative examples include the following models:

- First, Lu et al. [17] proposed a model, composed of three main modules: an encoder that uses locally connected layers or lightweight CNNs followed by bi-directional LSTMs to extract and combine fine-grained features from long traces, an attention mechanism to identify and focus on the most informative time steps, and a classifier to map the selected features to key predictions. This design avoids heavy fully connected layers, reduces dimensionality early, and selectively combines features, making it suitable for long, high-dimensional traces in practical SCA scenarios.

- Next, Feng et al. [9] proposed HACNN-SCA, a model that consists of three main modules; convolution module that extracts local patterns from power traces, a hybrid attention module that combines channel attention and spatial attention to enhance the representation of the relevant features of the local patterns, and a classification module that uses a fully connected layers to make predictions [9]. This architecture emphasizes the relevant parts of side channel leakage and suppresses the overall noise to boost the effectiveness of side-channel attacks.

- Finally, He et al. [12] proposed AMCNNet, a model that consists of three main components: a multi-scale convolution module that captures features at different scales, a feature extraction module that enhances important channels and uses self-attention to capture long-range dependencies, and a classification module that makes prediction. Similarly to Feng et al. proposed model, this architecture enhances the relevant features, and effectively captures both local and global dependencies within the power traces as well.

3.4 Mamba for DL-SCA

While attention-based mechanisms represent a significant advance, these architectures still depend on attention operations that become computationally expensive for long power traces and may fail to model global temporal structure efficiently [10, 28]. Recent progress in sequence-based architectures, particularly the emergence of Mamba blocks [10] that make use of selective state-space models, offers an alternative that can capture long-range dependencies with linear-time complexity. To date, only one work has explored Mamba block application. Zhaobin Li et al. proposed a CNN-Residual Mamba module hybrid architecture [16]. Their model consists of four modules: convolution module, residual Mamba module, MLP module, and a fully connected classification module. The Mamba module consist of three Mamba blocks stacked with a residual connection. However, the authors do not clearly specify which ASCAD dataset variant was used in their experiments. Based on the reported number of traces, it is likely that the dataset corresponds to ASCADv1 dataset.

Compared to our approach, their model captures long-range dependencies only in one temporal direction. In contrast, our work explicitly models explores bidirectional long-range dependencies which allows our network to exploit both past and future contextual information within power traces. To the best of our knowledge, the authors do not explore a bidirectional Mamba configuration, nor do they analyze its potential impact on side-channel attack performance.

4 Methodology

This chapter describes the methodology used to conduct the experiments and address the research question. We begin by briefly restating the research objective, followed by an introduction to the datasets used. Next, we present the two models considered in our study - the baseline MLP and the Mamba model. Finally, we outline the general experimental setup.

4.1 Research Objective

The objective of our research is to evaluate the effectiveness of the Mamba Block architecture for deep learning-based side-channel attacks. Originally designed to model context-dependent sequential data, Mamba can efficiently capture long-range dependencies. When applied to power consumption traces, these properties may offer significant advantages as side-channel leakage can be temporally distributed across multiple time samples, which creates complex contextual dependencies within the traces. In typical SCA architectures, convolution layers are used to extract local leakage features, while attention mechanisms are often added to model broader temporal context [12, 17, 9]. Mamba blocks may offer an alternative to attention mechanism without the computational overhead of self-attention, which can potentially enable more efficient extraction of global patterns from traces.

In our research, we follow a general workflow of deep learning-based side channel execution, described in Section 2.5. We consider a baseline multilayer perceptron (MLP) model proposed by Perin et al. in their work on feature-selection scenarios for deep-learning-based side-channel analysis [20]. The MLP represents a commonly used architecture in deep learning–based side-channel analysis and serves as a reference for evaluating the performance gains offered by the Mamba model. We propose our own model with bidirectional recurrent Mamba encoder blocks. Such bidirectional recurrent approach has been explored in context of attention to model long-range dependencies along both temporal direction of sequence data [6, 4]. Both models are trained to map feature-selected side-channel traces to key-dependent intermediate cryptographic values. Then the output probabilities of the cryptographic values are used to rank key hypotheses and ultimately recover the secret key. We execute DL-SCA on two separate datasets that contain power consumption traces of AES on both - unprotected and protected - versions of the encryption algorithm. This experimental setup enables a direct comparison between the Mamba-based model and the traditional baseline, which allows us to assess Mamba's ability to extract meaningful leakage information.

4.2 Side-Channel Datasets

We deployed two datasets to run our experiments. Both of the dataset cover two main side-channel attack scenarios - attacks on unprotected and protected implementations of AES-128 encryption algorithm. Table 4.1 summarizes statistical information of all data sets.

Dataset	Algorithm	Profile	Attack	Features	Platform	Countermeasure	Key	Target Byte
CW-Target	TinyAES	50'000	10'000	12'000	ARM Cortex-M4	None	Fixed	0
ASCADv1	AES	50'000	10'000	700	ATMega8515	Masked	Fixed	2

Table 4.1: Statistical summary of the side-channel datasets.

4.2.1 CW-Target Dataset

These traces were already collected and provided to us by the Digital Security lab. We named the dataset CW-Target. The side-channel traces were recorded with the ChipWhisperer platform and contain raw power consumption measurements of an unprotected TinyAES-128 algorithm deployed on a 32-bit ARM Cortex-M4 microcontroller. Each trace was recorded together with a plaintext and a secret key, fixed across all samples. The traces are pre-aligned and capture approximately the first four rounds of the algorithm execution. Figure 4.1 shows an example of a raw power traces, collected with ChipWhisperer.

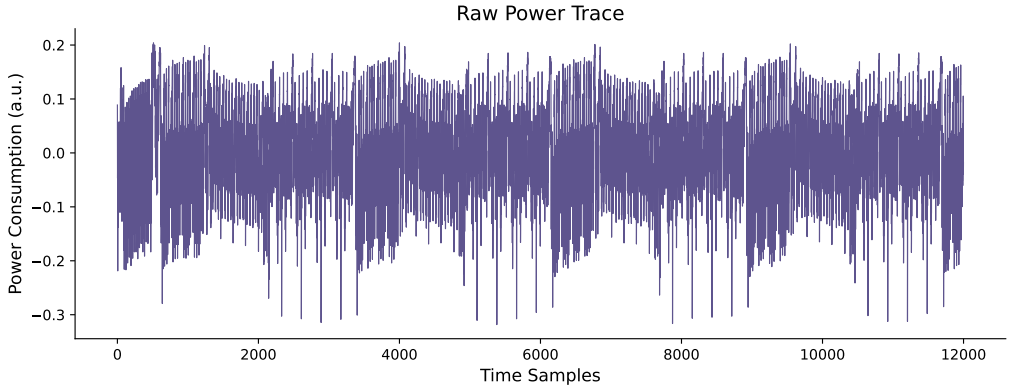

Figure 4.1: Example of raw power traces, collected with ChipWhisperer.

4.2.2 ASCADv1 Dataset

These traces contain raw power consumption measurements from a first-order Boolean-masked AES-128 algorithm executed on a 8-bit ATMega8515 MCU microcontroller [3]. Each trace is provided by the corresponding plaintext and a secret key. The secret key is fixed across samples. The traces are pre-aligned and cropped to a window that approximately covers teh target leakage point [3]. Figure 4.2 shows an example of a raw power traces, taken from ASCADv1 dataset.

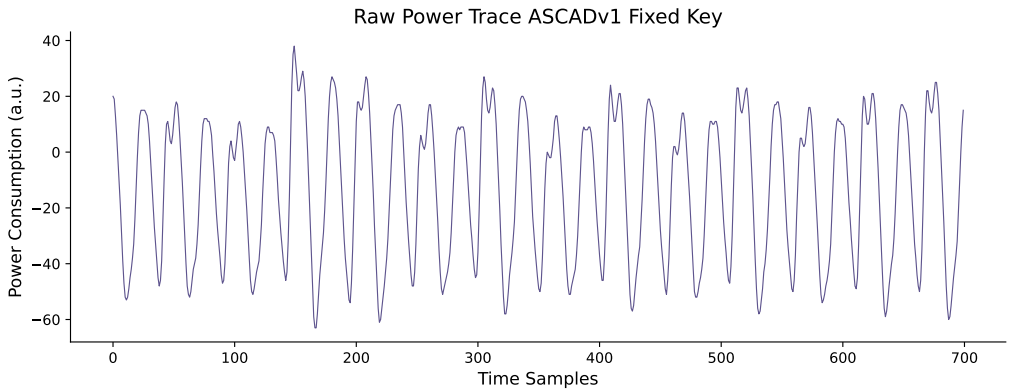

Figure 4.2: Example of raw power traces, ACSADv1.

4.3 Proposed Neural Network Architectures

To assess the attack efficiency of the Mamba model, we implement a baseline Multilayer Perceptron (MLP) for comparison.

Algorithm 4: MLP Forward Pass

Input: Input features $x \in \mathbb{R}^{B \times T}$
Output: Logits $y \in \mathbb{R}^{B \times K}$

$h \leftarrow \text{Linear}(x; T, N_{neurons}; \theta_1)$ $h \leftarrow \text{SELU}(h)$;
for $i \leftarrow 1$ **to** N_{hidden} **do**
    $h \leftarrow \text{Linear}(h; N_{neurons}, N_{neurons}; \theta_i)$;
    $h \leftarrow \text{SELU}(h)$;
$y \leftarrow \text{Linear}(h; N_{neurons}, K; \theta_{out})$
return y

4.3.1 MLP Model

The architecture of our MLP classifier follows the design proposed by Perin et al. in their work on feature-selection scenarios exploration [20]. Figure 4.3 visualizes the general schema of the MLP model deployed. The proposed MLP model takes a pre-process side-channel traces and outputs a probability distribution over all 256 S-box output values.

Algorithm 4 shows the forward pass of the proposed MLP model. The model takes a power traces of shape $\mathbb{R}^{B \times T}$ where B correspond to batch size and T corresponds to the number of temporal features, and outputs logits of shape $\mathbb{R}^{B \times K}$ where K corresponds to the number of target classes. The model maps the features of a power traces to a hidden representation of $N_{neurons}$ units, applies two layers of $N_{neurons}$ units, and finally outputs a probability distribution over K target classes. The model applies SeLU activation function between the first three layers, and uses cross-entropy loss function.

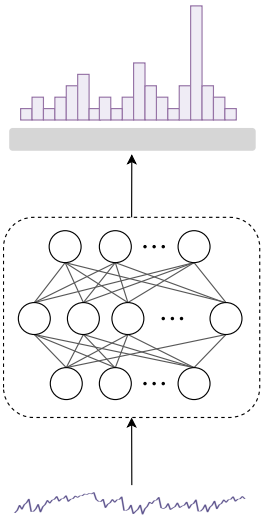

Figure 4.3: Baseline MLP model architecture.

4.3.2 Mamba Model

Figure 4.4 visualizes the proposed Mamba architecture. The models processed a power trace, represented as a one-dimensional sequence. The output layer produces a probability distribution over 256 classes that correspond to all the possible values of the targeted AES S-box output under the Identity (ID) leakage model.

Our proposed model consists of three main modules: convolution module, bidirectional Mamba block module and classification module.

- **Convolution Module Design**.
  The convolution module converts a raw power trace into a latent feature representation. It consists of

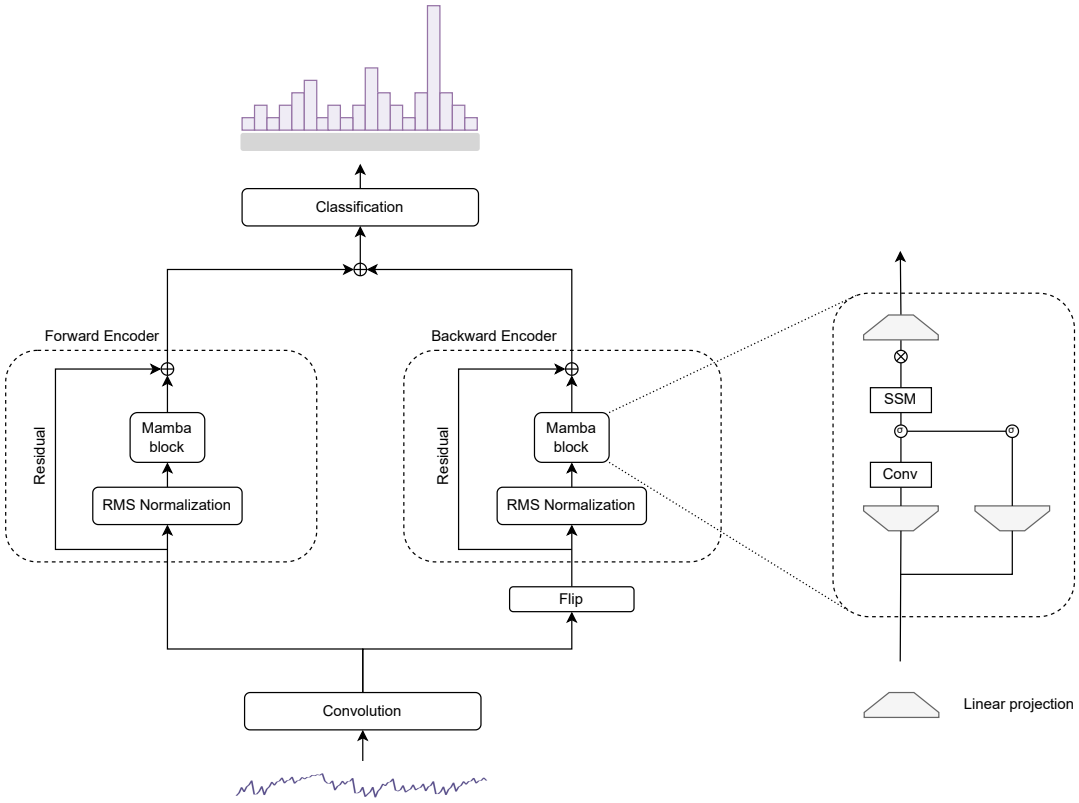

Figure 4.4: Mamba model architecture.

1D convolution layers applied along the temporal dimension of the trace. The output of the module is a sequence of embedded feature vectors.

- **Bidirectional Mamba Module Design**.
  The bidirectional Mamba module deploys Mamba blocks, to model long-range dependencies within the embedded power traces. The module consists of two parallel encoders - a forward Mamba encoder and a backward Mamba encoder, each composed of a stack of Mamba blocks followed by residual connections and layer normalization. The forward Mamba encoder processes the embedded power trace in the forward direction (from left to right) to capture dependencies in the original temporal direction. To model context that may not be fully represented in the forward direction alone, the backward Mamba encoder processes the sequence in reverse order - the embedded power trace is flipped and passed through the backward Mamba encoder to model dependencies in the backwards direction (from right to left). The combination of forward and backward Mamba encoder blocks allow the Mamba module to model long-range dependencies in both temporal directions. The output from both encoders are then concatenated along the the last feature dimension and passed to the subsequent classification module.

- **Classification Module Design**.
  The classification module consists of a single linear transformation layer that maps the high-dimensional representations to the target class space.

Algorithm 5 demonstrates the forward pass of the proposed Mamba architecture. The model takes a power traces of shape $\mathbb{R}^{B\times T}$ where B correspond to batch size and T corresponds to the number of temporal features, and outputs logits of shape $\mathbb{R}^{B\times K}$ where K corresponds to the number of target classes. The convolution layer embeds the power traces to D_{model} dimensions. The embedded trace is then processed in parallel by forward and backward Mamba encoder blocks: the forward encoder receives the original trace, while the backward encoder receives the reversed trace. Each trace passes through N_{LAYERS} Mamba encoder blocks, for a total of $2 \times N_{blocks}$ blocks. Since the Mamba blocks are connected through a residual connection, the last token's output from each encoder captures a compressed summary of the sequence. These last hidden

Algorithm 5: Bidirectional Mamba Forward Pass

Input: Raw signal $x \in \mathbb{R}^{B \times T}$
Output: Logits $y \in \mathbb{R}^{B \times K}$

// Convolution Module
$x \leftarrow \text{Reshape}(x, [B, 1, T])$; // output shape $\in \mathbb{R}^{B \times 1 \times L}$
$z \leftarrow \text{Conv1d}(x; D_{model}; \theta_{conv})$; // output shape $\in \mathbb{R}^{B \times D_{model} \times L}$
$z \leftarrow \text{Reshape}(z, [0, 2, 1])$; // output shape $\in \mathbb{R}^{B \times L \times D_{model}}$

// Bidirectional Mamba Module
$z_{fwd} \leftarrow z$;
$z_{bwd} \leftarrow \text{Flip}(z, \text{dim} = 1)$;
for $i \leftarrow 1$ **to** N_{blocks} **do**
  $z_{fwd} \leftarrow \text{Block}^{(i)}_{fwd}(z_{fwd})$; // forward Mamba blocks
  $z_{bwd} \leftarrow \text{Block}^{(i)}_{bwd}(z_{bwd})$; // backward Mamba blcoks

// Classification Module
$h_{fwd} \leftarrow z_{fwd}[:, L, :]$;
$h_{bwd} \leftarrow z_{bwd}[:, L, :]$;
$h_{combined} \leftarrow \text{Concat}([h_{fwd}, h_{bwd}], \text{dim} = -1)$; // output shape $\in \mathbb{R}^{B \times 2D_{model}}$
$y \leftarrow \text{Linear}(h_{combined}; \theta_{net})$; // output shape $\in \mathbb{R}^{B \times K}$
return y

states are concatenated to form a single feature vector of shape $\mathbb{R}^{B \times 2D_{model}}$, which contains information from both temporal directions. The concatenation features are then passed to the classification module where they are mapped to a probability distribution over K classes.

4.3.3 Hyperparameter Search

We conducted a hyperparameter search for the proposed Mamba model to identify a configuration of the parameters that yields the best attack performance. Once we found such a configuration, we adjusted the number of neurons per layer in the MLP model to approximately match the number of learnable parameters of the Mamba model. Matching the parameter count helps eliminate size-related bias, as the Mamba architecture is inherently more complex than a standard MLP and could otherwise outperform the MLP simply due to higher model capacity. The controlled learnable parameter count help us isolate the effect of the model architecture itself which allows us to attribute the performance differences to architectural characteristics rather than to differences in model size.

We experimented with different combinations of kernel sizes $k = \{3, 5\}$ and stride $s = \{1, 3, 5\}$. The kernel size of 3 together with stride of 3 showed the best results. Configurations where the stride was smaller than the kernel size—for example, a kernel of 3 with stride 1 or a kernel of 5 with stride 3—did not perform well, as the average Ge and GE convergence exceeded the final reported values. In addition, a kernel size and stride of 5 led to even higher average GE and GE convergence. We then deployed the model with multiple convolution layers followed by ReLu activation and batch normalization but that did not seem to improve the current state of the model. Finally, we experimented with different model dimensions $D_{model} = \{32, 64\}$ and number of stacked Mamba block $N_{blocks} = \{2, 4\}$. The combination of model dimension of 64 and 2 Mamba encoder blocks (4 Mamba blocks in total) showed the best results.

For the Mamba model, we selected a model dimension D_{model} of 64, and number of Mamba block N_{blocks} of 2. Hence, our proposed Mamba model expands the dimension of the embedded traces further to 64 dimensions and applies four Mamba encoder: two forward encoder blocks and two backward encoder blocks. The convolution module consists of a single one-dimensional convolution layer with a kernel size and stride of 3. To approximately match the number of learnable parameters of the Mamba model, we set the number of neurons per hidden layer$N_{neurons}$ of 128 in the MLP model. Table 4.2 summarized the chosen model hyper parameters.

Model	Learnable Parameters	Hyperparameter	Symbol	Value
MLP Architecture	155'000	Number of Neurons	$N_{neurons}$	128
		Number of Hidden Layers	N_{hidden}	2
Mamba Architecture	148'000	Model Dimension	D_{model}	64
		Number of Mamba Encoders	N_{blocks}	2

Table 4.2: Hyperparameters and experimental setup used across all experiments.

4.4 Experimental Setup

4.4.1 Data Preparation

We targeted different bytes for each dataset, and we performed feature selection only on the unprotected CW-Target dataset, as ASCADv1 contains already feature selected traces.

- **CW-Target Traces**.
  We targeted the first byte (byte index $i = 0$) of the key. To label each side-channel trace (2.5.3), we calculate the cryptographic intermediate value of the S-box function under the ID leakage model. Equation (4.2) defines label y_j for the j-th trace, where p_j and k_j^* represent the j-th plaintext and correct key, respectively.

$$y_j = \text{S} - \text{box}\Big[\, p \oplus k_j^* \,\Big] \tag{4.1}$$

  To select features (2.5.4), we deployed the OPOI method. For this purpose, the SNR was computed with the HW leakage model. Specifically, the original ID-based labels were converted to their HW representations, and the traces were grouped accordingly. For each HW group, the mean of all traces was taken as the signal, while the noise was defined as the difference between each trace and the mean trace (signal). We then computed the variance of both signal and noise to calculate the signal-to-noise ratio (SNR) for each time sample. Based on the computed SNR trace, the top 15 time samples with the highest SNR values were selected as Points of Interest (POIs). These POIs were further visualized to approximate their location within the full power trace to select the optimized window. Appendix A.1 visualizes the SNR trace for HW leakage model, and a sample trace with selected POIs.

  The power traces were then standardized - we fitted the normalization parameters on the profile set and applied the same transformation to the validation and attack sets. The labels for all splits were converted into one-hot encoded vectors.

- **ASCADv1 Traces**.
  We targeted the third byte of the key (byte index $i = 2$), as the first two bytes are not protected. We use the provided labels for each trace under the ID leakage model. Equation (4.2) defines label y_j for the j-th trace, where p_j and k_j^* represent the j-th plaintext and correct key, respectively.

$$y_j = \text{S} - \text{box}\Big[\, p \oplus k_j^* \,\Big] \tag{4.2}$$

  The power traces were standardized as well. We fitted the normalization parameters on the profile set and applied the same transformation to the validation and attack sets. The labels for all splits are converted into one-hot encoded vectors.

4.4.2 Profile Phase Setup

To train our models, we split the original profile set further to a profile and a validation subset. The validation set was used exclusively for model performance validation and selection. Table 4.4 summarizes the data splits.

Dataset	Train Split	Validation Split	Test Split
CW-Target	40'000	10'000	10'000
ASCADv1	40'000	10'000	10'000

Table 4.3: Data split for training, validation and testing.

Both models were trained for a maximum of 100 epochs with batch size B of 768 and learning rate LR of 5×10^{-3}. We used Adam optimizer with weight decay enabled. Table summarizes the model training configurations.

Model validation was performed with the average GE metric. Accuracy and loss were not used for validation, as they are known to be poorly correlated with key-recovery performance [20]. The mean GE measures the average rank of the correct key byte across 100 attack executions. During model training, the validation GE was monitored over 100 epochs, and the model checkpoint that corresponded to the lowest validation GE - beyond which no further improvement was observed - was selected for the final evaluation.

Parameter	Symbol	Value
Batch Size	B	768
Learning Rate	LR	5×10^{-3}

Table 4.4: Model training parameters.

4.4.3 Attack Phase Setup

To obtain the most likely key-byte candidate, we constructed a key guessing vector (2.5.7). To assess the effectiveness of key recovery, we applied three SCA metrics (2.5.7): key rank, guessing entropy (GE), and guessing entropy convergence. The key rank was calculated on the entire test set of 10'000 traces. The key rank represent the position of the correct key within a key score vector. To assess the model's reliability, we performed 100 independent attack simulations. For each simulation, a random subsample of 1'000 traces was drawn from the test set. The Average GE represents the mean rank of the correct key across these 100 iterations. Finally, we analyzed the GE convergence over 1'000 traces. This metric illustrates the rate at which the model's uncertainty decreases as more traces are provided, which detects the minimum trace count required for the correct key to reach the lower rank.

5 Results

This chapter presents the experimental results obtained by applying the proposed Mamba-based model and a baseline multilayer perceptron (MLP) to both unprotected and protected side-channel analysis (SCA) scenarios. Additionally, the chapter shows the model validation process during training, using guessing entropy as the primary evaluation metric.

5.1 Attack on ChipWhisperer Dataset

Table 5.1 summarizes the attack performance comparison of baseline MLP and Mamba models on the unprotected side-channel traces, collected with ChipWhisperer. Both models achieve perfect key recovery with rapid guessing entropy convergence. Figure 5.1 shows the validation metric of mean GE during model training. Figure 5.2 displays the GE convergence of both models together with the lowest GE value.

Model	Key Rank	Median GE	Mean GE	GE Convergence
MLP	0	0.0	0.0	0.0 with 3 traces
Mamba	0	0.0	0.0	0.0 with 3 traces

Table 5.1: Attack performance on CW-Target dataset.

Figure 5.1 presents the validation GE over 100 epochs of training. For both models, the validation GE reached 0.0 from the very first epoch and remained stable. As summarized in Table 5.1 and illustrated in Figure 5.2, both models successfully broke the unprotected dataset, with a key rank of 0 and median and mean GE of 0.0. Notably, the GE converged to 0.0 after only three attack traces out of 1000, demonstrating extremely fast attack convergence.

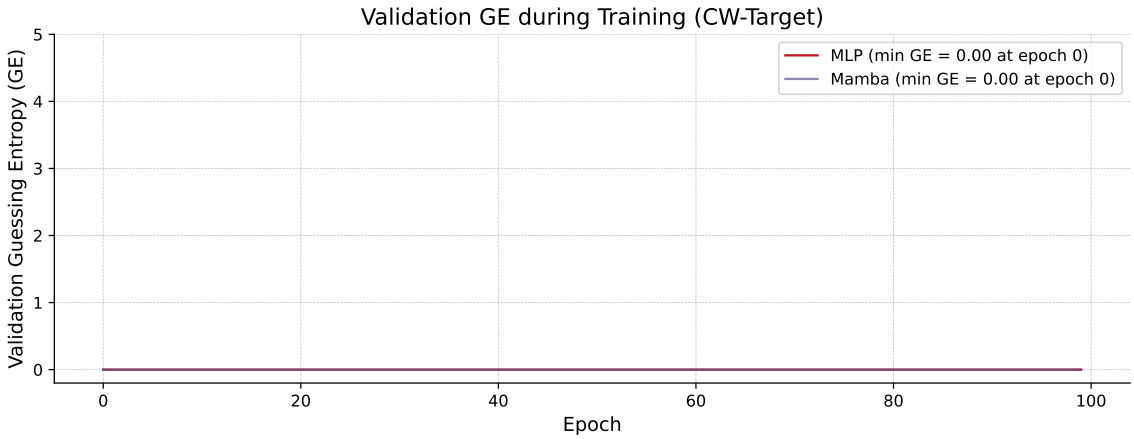

Figure 5.1: Validation GE of Mamba and MLP model.

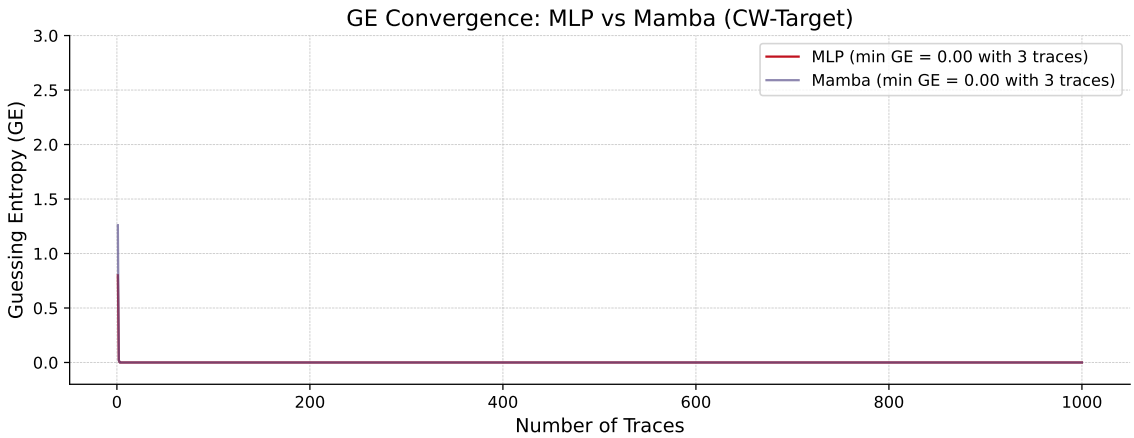

Figure 5.2: GE convergence of Mamba and MLP model.

5.2 Attack on ASCADv1 Dataset

Table 5.2 summarizes the attack performance comparison of baseline MLP and Mamba models on the protected side-channel traces, ASCADv1 dataset with fixed key. Both models break the dataset with successful key recovery. Figure 5.3 shows the validation metric of mean GE during model training. Figure 5.4 displays the GE convergence of both models together with the lowest GE value.

Model	Key Rank	Median GE	Mean GE	GE Convergence
MLP	0	0.0	0.0	0.01 with 983 traces
Mamba	0	0.0	0.04	0.75 with 986 traces

Table 5.2: Attack performance on ASCADv1 dataset.

The baseline MLP model converged to a stable validation GE of 0.0 at epoch 15. The model successfully recovered the secret key with a key rank of 0 and median and mean GE values of 0.0, which reflects perfect key recovery. With 1000 attack traces, the GE converged to 0.01 after 983 traces. The Mamba model converged to a validation GE of 0.92 at epoch 14 and was also able to recover the secret key. However, the attack performance did not match that of the baseline model. The Mamba model achieved a final key rank of 0 with median and mean GE values of 0.0 and 0.04, respectively. When evaluated with 1000 attack traces, the GE converged to 0.75 after 986 traces, which is slightly slower convergence compared to the baseline model.

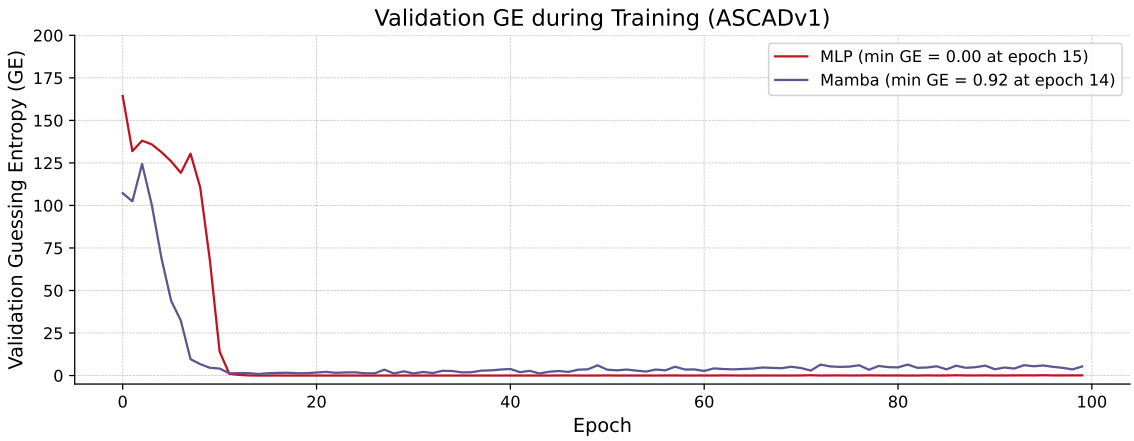

Figure 5.3: Validation GE of Mamba and MLP model.

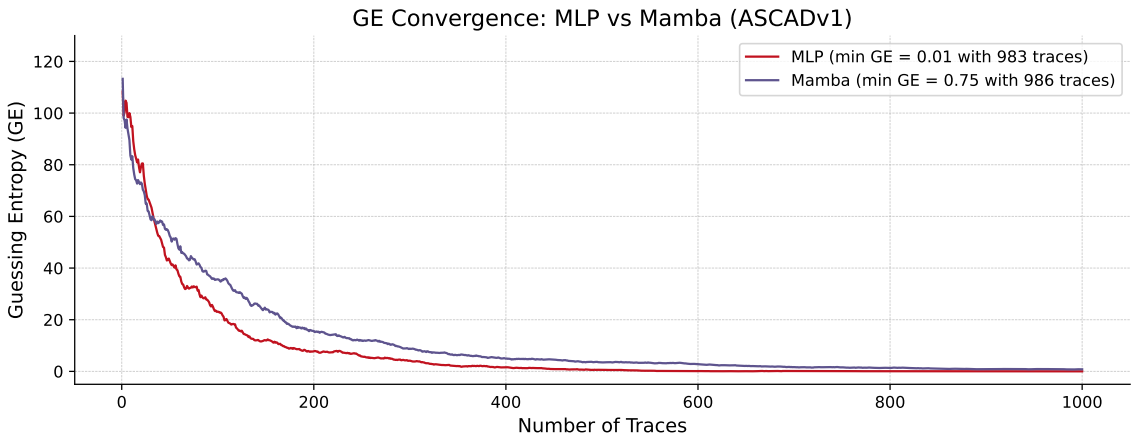

Figure 5.4: GE convergence of baseline MLP and Mamba model.

6 Discussion

This chapter presents and analyzes the outcomes of our experiments. The aim of the discussion is to evaluate whether the Mamba architecture provides practical benefits in terms of attack efficiency and convergence characteristics when compared to a commonly used baseline model. In addition, we outline the limitations of our experiments and suggest potential directions for future research to address them.

6.1 CW-Target Dataset

Both models reached a validation guessing entropy (GE) of 0.0 from the first epoch, and this remained constant throughout the rest of training. In terms of attack performance, both models achieved a key rank of 0 and median and mean GE values of 0.0. GE convergence occurred with as few as three traces.

Overall, both models recover the correct key byte almost immediately, requiring only three traces. This suggests that the dataset is relatively easy to attack, and as a result, both models provide limited insight into potential architectural advantages.

6.2 ASVADv1 Dataset

Both models achieved a validation guessing entropy (GE) close to 0.0 around epoch 15. While Mamba reached a GE of 0.92 at that point, the baseline MLP model plateaued at a GE of zero. From Figure 5.3, we can observe that Mamba trained faster, reaching a low GE value around epoch 10, whereas MLP achieved a similar value approximately five epochs later. During the training of the first 10 epochs, the GE of the MLP model fluctuated more than that of Mamba model, but subsequently it decreased rapidly. Although we selected checkpoints around epoch 15 for attack execution, we continued the training of both models to monitor the behavior of the validation metric over later epochs. For the MLP model, the validation GE remained consistently zero, while for Mamba it fluctuated slightly, with the fluctuations becoming more pronounced toward the final epochs.

The Mamba model achieved a key rank of 0. This means that with the model predictions we were able to compose a vector of key byte candidate scores such that the correct key byte was the first one within the score vector. In real life attack scenario, the adversary would have to *brute-force* only one key byte to recover the correct one. The key rank results are the same with baseline MLP model.

The median an the mean of GE was 0.0 and 0.04 respectively. The median GE reveals that the most common rank of the correct key byte among all key byte candidates was 0.0 when the the attack was executed on a random subsample of 1000 traces over 100 attack runs. The slightly higher mean GE suggests that a few attack runs produced a GE greater than 0.0. In comparison, the baseline MLP achieved a mean GE of 0.0 which reveals that all 100 attack runs consistently ranked the correct key byte first, with no deviation across runs. This observation suggests that the MLP model exhibits a more consistent performance of attack simulations.

Guessing entropy (GE) convergence demonstrates the number of traces required before the correct key byte emerges at the top of the key rank score vector. In our evaluation, GE convergence simulates an attack scenario in which only 1000 traces are available to the adversary. As additional traces are processed, the log-likelihoods accumulate across traces, which eventually allows the correct key byte candidate (signal) to progressively dominate over the rest 255 incorrect candidates. The Mamba model achieves a GE of 0.75 with 986 traces, while the baseline MLP model reaches GE of 0.01 with 983 traces. From Figure 5.4 we can observe

that Mamba model seems to converge faster than MLP model for approximately the first 50 traces but then MLP converges faster reaching a plateau much faster. Even though both models achieve a minimum GE of smaller than 1.0 around trace 985, the GE of MLP model converge faster and more stable.

Overall, both models achieve perfect key recovery, with a key rank of 0 and GE convergence below 1.0 using fewer than 1,000 attack traces. The Mamba model converges slightly slower than the baseline MLP model.

6.3 Future Work

In real-world attack scenarios, side-channel traces often suffer from misalignment, and feature selection may not be feasible if traces are collected without the knowledge of the secret key. Consequently, it would be valuable to evaluate the Mamba model on raw traces without alignment or pre-selected features, as well as with presence of noise. While both models demonstrated similar performance on the current dataset, it would be valuable to investigate how well they can learn from more challenging traces that contain noise, misalignment, and no selected features, making the attack task significantly harder. Although the ASCADv2 dataset is considered more difficult to attack due to stronger countermeasures (affine masking and shuffling), we were unable to run our models on this dataset because of hardware limitations. ASCADv2 traces have 1'400 features, compared to 700 features in ASCADv1, which we used for our study. We could not fit this large input size on our GPU. Additionally, both datasets used - CW-Target and ASCADv1 - contain fixed keys, which possibly contributed to the high performance of the models, as the models can simply memorize the fixed key. The dataset becomes more challenging when keys vary across the profile traces. However, then GE can no longer be used as a validation metric, a limitation noted by previous studies [24].

Beyond dataset difficulty, a more extensive hyperparameter search could further improve model performance. Although we explored the effects of different hyperparameter of the convolution module and Mamba module, we did not perform automated hyperparameter optimization as suggested by prior works [29]. An exhaustive search could potentially offer models with even better performance.

Finally, our experiments were constrained by limited GPU resources (10GB), which prevented us from scaling the models to larger numbers of learnable parameters. As a result, we could not fully explore the effect of model size on attack performance or determine how larger models might improve key recovery. Access to greater computational resources would allow us to explore more complex models.

7 Conclusions

Mamba blocks, specifically the bidirectional Mamba encoder stacked with residual connections, demonstrate that selective state space–based architectures can be effectively applied to the side-channel attack context. The proposed Mamba-based model achieved successful key recovery with a key rank of zero and a fast GE convergence. These results confirm that Mamba-based neural network architectures are able to learn meaningful representations from side-channel traces and exploit temporal dependencies present within power consumption traces.

Despite this, the Mamba model did not outperform the baseline MLP model. In our experiments, the MLP model achieved faster GE convergence, and reached GE close to zero across repeated attack executions. This suggests that, for relatively simple datasets such as ASCADv1 with fixed keys, straightforward MLP architecture may already be sufficient to fully capture the exploitable leakage. In such scenarios, the additional architectural complexity of Mamba-based models offer no measurable attack effectiveness gains.

Nevertheless, the results provide valuable knowledge about the trade-offs between model complexity and attack effectiveness. While MLP models appear to be more robust and consistent, Mamba-based models offer a fundamentally different approach through its state-space formulation. This characteristic may become more advantageous in attack scenarios with side-channel traces that contain longer traces, misalignment, noise, and stronger countermeasures.

In conclusion, although the baseline MLP remains a strong and competitive approach for the datasets considered in this work, Mamba blocks represents an alternative architecture for deep-learning-based side-channel attacks. Future work should focus on evaluating Mamba under more challenging conditions where its ability to model long-range dependencies may offer a clearer advantage.

Bibliography

[1] Amjed Abbas Ahmed and Mohammad Kamrul Hasan. Multi-layer perceptrons and convolutional neural networks based side-channel attacks on AES encryption. In *2023 International Conference on Engineering Technology and Technopreneurship (ICE2T)*, pages 69–73, 2023.

[2] Amjed Abbas Ahmed, Mohammad Kamrul Hasan, Imran Memon, Azana Hafizah Mohd Aman, Shayla Islam, Thippa Reddy Gadekallu, and Sufyan Ali Memon. Secure AI for 6G mobile devices: Deep learning optimization against side-channel attacks. *IEEE Transactions on Consumer Electronics*, 70(1):3951–3959, 2024.

[3] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering*, 10, 06 2020.

[4] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. BRITS: Bidirectional recurrent imputation for time series. *NIPS*, 2018.

[5] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[6] Armando Collado-Villaverde, Pablo Muñoz, and María D. R-Moreno. BRATI: Bidirectional recurrent attention for time-series imputation. 2025.

[7] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. *Journal of Cryptographic Engineering*, 1(2):123–144, 2011.

[8] Sanjit Arunkumar Seshia Edward Ashford Lee. Introduction to embedded systems - a cyber-physical systems approach. 2017.

[9] T. Feng, H. Gao, X. Li, et al. Side-channel attacks on convolutional neural networks based on the hybrid attention mechanism. *Discover Applied Sciences*, 7(390), 2025.

[10] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024.

[11] F. Hameed and H. Alkhzaimi. Deep learning-based profiling side-channel attacks in SPECK cipher. *Scientific Reports*, 15(26149), 2025.

[12] Pengfei He, Ying Zhang, Han Gan, Jianfei Ma, and Hongxin Zhang. Side-channel attacks based on attention mechanism and multi-scale convolutional neural network. *Computers and Electrical Engineering*, 119:109515, 2024.

[13] J.H. Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise: Unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):148–179, 2019.

[14] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[15] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology — CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Santa Barbara, California, USA, August 15–19 1999. Springer.

[16] Zhaobin Li, Chenchong Du, and Xiaoyi Duan. Efficient AES side-channel attacks based on residual Mamba enhanced CNN. *Entropy*, 27(8), 2025.

[17] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):235–274, Jul. 2021.

[18] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 3–26, Cham, 2016. Springer International Publishing.

[19] National Institute of Standards, Technology (NIST), Morris J. Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence E. Bassham, E. Roback, and James Dray Jr. Advanced encryption standard (AES), 2001-11-26 00:11:00 2001.

[20] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):828–861, Aug. 2022.

[21] Stjepan Picek, Annelie Heuser, Alan Jovic, and Lejla Batina. A systematic evaluation of profiling through focused feature selection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12):2802–2815, 2019.

[22] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):209–237, Nov. 2018.

[23] Stjepan Picek, Annelie Heuser, Guilherme Perin, and Sylvain Guilley. Profiled side-channel analysis in the efficient attacker framework. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications*, pages 44–63, Cham, 2022. Springer International Publishing.

[24] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *ACM Comput. Surv.*, 55(11), February 2023.

[25] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and countermeasures for smart cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, pages 200–210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[26] François-Xavier Standaert. Introduction to side-channel attacks. In Ingrid M.R. Verbauwhede, editor, *Secure Integrated Circuits and Systems*, pages 27–42, Boston, MA, 2010. Springer US.

[27] S. Swaminathan, Ł. Chmielewski, G. Perin, and S. Picek. Deep learning-based side-channel analysis against AES inner rounds. In *Applied Cryptography and Network Security Workshops (ACNS 2022)*, volume 13285 of *Lecture Notes in Computer Science*, Cham, 2022. Springer.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[29] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. *IEEE Transactions on Emerging Topics in Computing*, 12(2):546–557, 2024.

[30] Lichao Wu, Léo Weissbart, Marina Krček, Huimin Li, Guilherme Perin, Lejla Batina, and Stjepan Picek. Label correlation in deep learning-based side-channel analysis. *Trans. Info. For. Sec.*, 18:3849–3861, January 2023.

A Appendix

A.1 Feature Selection with HW Leakage Model

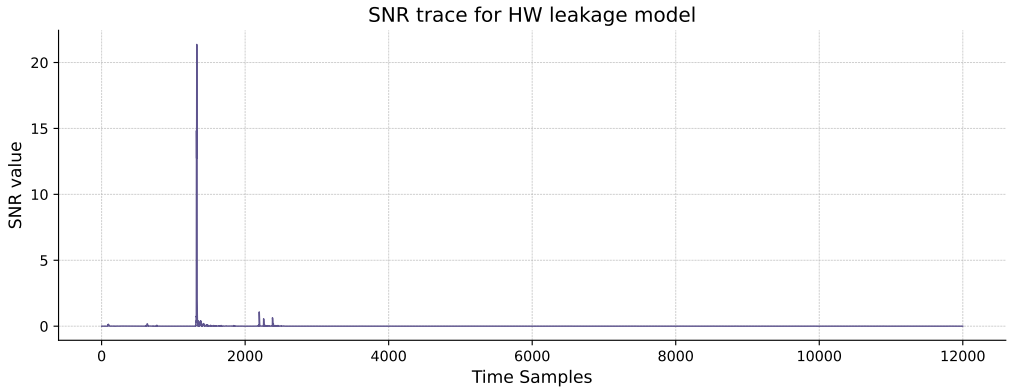

Figure A.1: SNR trace for the Hamming Weight leakage model.

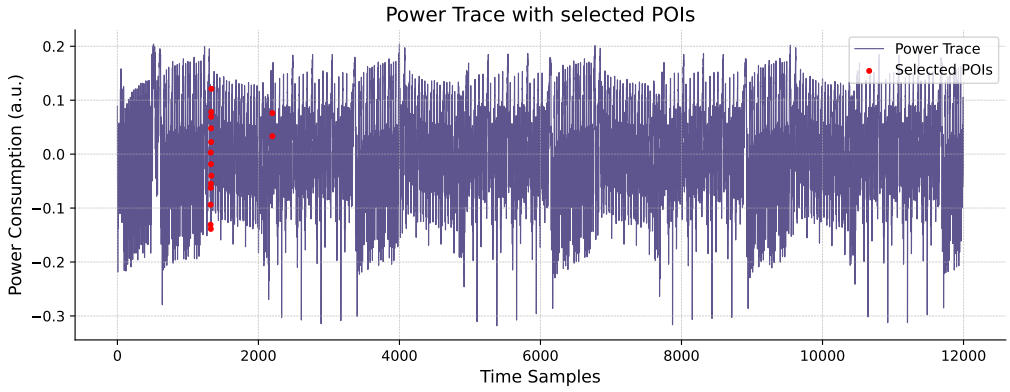

Figure A.2: Raw power trace sample with selected refined points of interest (POIs).

From Figures A.1 and A.2 we can observe that several POIs are selected not only around the main leakage region at approximately time sample 1400, but also around time sample 2200. These secondary selections are consistent with the signal-to-noise ratio (SNR) profile, which exhibits a very small peak near sample 2200. However, this peak is negligible compared to the dominant leakage peak observed around sample 1400. Consequently, we can confidently identify the leakage window centered at sample 1400 as the primary source of information leakage. Occasionally, points outside the main leakage region may be included among the selected POIs, especially when a large number of POIs is selected. In such cases, a noise-dominated sample may be added to the selected POIs along with the informative ones. This may occur, for example, if the leakage corresponding to the target byte is weaker or overlaps with leakage from another key byte.