IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2021, No. 1, pp. 279–304. DOI:10.46586/tches.v2021.i1.279-304

Side-Channel Analysis of the Xilinx Zynq UltraScale+ Encryption Engine

Benjamin Hettwer[1,2], Sebastien Leger[1], Daniel Fennes[2], Stefan Gehrer[3] and Tim Güneysu[2]

[1] Robert Bosch GmbH, Corporate Sector Research, Stuttgart, Germany
{benjamin.hettwer,sebastien.leger}@de.bosch.com
[2] Ruhr University Bochum, Bochum, Germany
{daniel.fennes,tim.gueneysu}@rub.de
[3] Robert Bosch LLC, Pittsburgh, USA
stefan.gehrer@bosch.com

Abstract. The Xilinx Zynq UltraScale+ (ZU+) is a powerful and flexible System-on-Chip (SoC) computing platform for next generation applications such as autonomous driving or industrial Internet-of-Things (IoT) based on 16 nm production technology. The devices are equipped with a secure boot mechanism in order to provide confidentiality, integrity, and authenticity of the configuration files that are loaded during power-up. This includes a dedicated encryption engine which features a protocol-based countermeasure against passive Side-Channel Attacks (SCAs) called key rolling. The mechanism ensures that the same key is used only for a certain number of data blocks that has to be defined by the user. However, a suitable choice for the key rolling parameter depends on the power leakage behavior of the chip and is not published by the manufacturer. To close this gap, this paper presents the first publicly known side-channel analysis of the ZU+ encryption unit. We conduct a black-box reverse engineering of the internal hardware architecture of the encryption engine using Electromagnetic (EM) measurements from a decoupling capacitor of the power supply. Then, we illustrate a sophisticated methodology that involves the first five rounds of an AES encryption to attack the 256-bit secret key. We apply the elaborated attack strategy using several new Deep Learning (DL)-based evaluation methods for cryptographic implementations. Even though we are unable to recover all bytes of the secret key, the experimental results still allow us to provide concrete recommendations for the key rolling parameter under realistic conditions. This eventually helps to configure the secure boot mechanism of the ZU+ and similar devices appropriately.

Keywords: Side-Channel Attacks, Deep Learning, Bitstream Encryption, Key Rolling

1 Introduction

Side-Channel Attacks (SCAs) have been a known threat to embedded systems for more than 20 years [KJJ99]. They provide a modern way of attacking cryptographic implementations of mathematical sound algorithms by exploiting information leaks via, e.g., power consumption or Electromagnetic (EM) emanations. While the discovery of SCAs was a nightmare for the smart card industry in early times, they are nowadays also considered by a broader range of hardware manufacturers due to numerous published attacks against commercial devices [EKM+08, KOP09, LDMPT15, OP11]. In particular, the bitstream decryption engine of several Field Programmable Gate Array (FPGA) device families have been successfully broken in the last years. For example, Moradi et al. performed

Received: 2020-07-15 Accepted: 2020-09-15 Published: 2020-12-03

a black-box analysis of the Xilinx Virtex-4 and Virtex-5 bitstream encryption mechanism [MKP12]. They first reverse engineered the internal architecture of the AES-256 implementation and then mounted several Correlation Power Analysis (CPA) attacks with a 2^{32}-bit key hypothesis to recover the last two round keys. A couple of Graphical Processing Units (GPUs) have been used to speed up the calculations. Later, the same research group showed that the attack complexity can be reduced to 2^8 by following a dedicated measurement procedure (i.e. fixing certain ciphertext bytes to a constant value) [MS16]. As the same bitstream decryption module is integrated in a range of FPGA families from the 5, 6 and 7 series, the presented attack can be applied against a large number of Xilinx devices.

Knowledge of the bitstream encryption key enables an attacker to copy, reverse engineer or manipulate Intellectual Property (IP) which have potentially cost several hundred person-years of development time. Because of that and in light of the aforementioned attacks, Xilinx integrated two protocol-based countermeasures into their current SoC device generation called Zynq UltraScale+ (ZU+) in order to increase the resistance against SCAs. The hardware root of trust secure boot mode uses RSA authentication before decrypting the configuration file to prevent adversaries from chosen-input attacks. Additionally, key rolling limits the number of encryptions that are performed under a given key [ZU_19]. This involves that the configuration image is divided into several chunks, and each chunk is encrypted with a unique secret. Keys for successive data blocks are stored in previous data chunks. There is a trade-off between configuration time and security as fewer encryption operations per key provide higher SCA protection but also increase the bitstream size [ZU_17a]. However, Xilinx does not give a concrete statement about the maximum amount of data that can be encrypted by a single key to be secure against certain kinds of SCAs. Users of the device therefore face the problem to set the key rolling parameter to a value that fits their requirements without knowing any internals of the AES-256 encryption unit (e.g. the leakage behaviour). This poses a potential security issue for systems that include ZU+ devices.

1.1 Contribution

In this work, we assess the side-channel leakage of the encryption unit of the Xilinx ZU+ and provide a recommendation for the key rolling security parameter. We perform a black-box reverse engineering of the hardware architecture of the embedded Advanced Encryption Standard (AES) module by using CPA with known secret. Next, we present an attack procedure that allows to extract the 256-bit key involving the first five AES rounds. Finally, we demonstrate the application of our attack procedure using an enhanced version of the DL-based correlation optimization scheme introduced at CHES 2019 [RQL18]. Although we are not able to recover all bytes of the AES-256 key in our experiments, we can provide concrete numbers about the remaining attack complexity. As a consequence of our results, we are able to suggest a suitable range of parameter values for the key rolling countermeasure that takes security concerns as well as device boot time into account.

1.2 Structure of the Paper

In Section 2 we introduce the preliminaries of our paper. Section 3 covers related work. In Section 4 we explain our measurement setup and present the reverse engineered hardware architecture of the AES encryption engine. In Section 5 we show a generic way to recover all bytes of the 256-bit AES key. In Section 6 we present and discuss our DL-based attacks. Based on the results, Section 7 gives a recommendation for a parametrization of the key rolling countermeasure. Section 8 concludes the paper.

2 Preliminaries

This section covers the basics about FPGA security, SCA countermeasures of the ZU+, and DL-based attacks.

2.1 FPGA Security

FPGAs are powerful and flexible reconfigurable devices, which are mostly based on volatile Static Random-Access Memory (SRAM) technology. The configuration file (i.e. bitstream) must be loaded at each power-up phase of the device. Protection of the bitstream against duplication, manipulation, and/or reverse engineering is very important. FPGA manufacturers provide security mechanisms such as encryption and authentication to ensure confidentiality and authenticity of the IP. In this work we focus on the bitstream encryption mechanism of the ZU+, which is relevant for IP protection.

The bitstream is encrypted with the AES encryption algorithm in a trusted environment and decrypted inside the FPGA at boot time to ensure the plain bitstream is not available outside the FPGA. This mechanism relies on a secret AES key, which is securely stored in the FPGA. The Xilinx ZU+ SoC provides an AES-256 hardware based encryption engine used to decrypt the bitstream at boot time. For more information, we refer the reader to the ZU+ reference manual [$\mathrm{ZU}_1 9$].

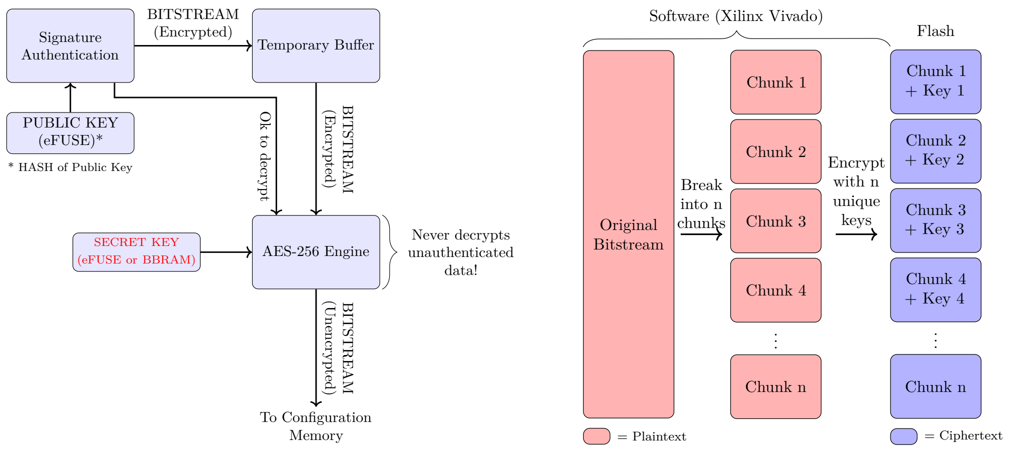
2.2 Side-channel Protections of the Xilinx Zynq UltraScale+

A way to narrow down an adversary's opportunities for analyzing the device's secret key is the usage of *authenticated de-/encryption*. The schematic overview of the authenticated decryption flow of the Xilinx ZU+ platform is given in Figure 1a [$\mathrm{ZU}_1 9$]. This authentication mechanism is based on public key authentication (RSA) and guarantees that the hardware encryption engine only operates on authenticated data and not on any random data provided by the attacker. This mitigates chosen-input SCAs and restricts the capabilities of an adversary to collect arbitrary side-channel information.

Furthermore, the Xilinx ZU+ provides a protocol based SCA countermeasure called *key rolling*. SCAs rely on the measurement of many different encryption operations using the same cryptographic secret (key). These measurements are called traces. Depending on the Signal-to-Noise Ratio (SNR) of the measurements and the leakage model, the minimum number of traces required to extract the key will vary. If the attacker is not able to record this minimum number of traces, she will not be able to extract the secret key. With key rolling, the bitstream is divided into smaller chunks. Each chunk is encrypted with its own key by the Xilinx development tool (i.e. Vivado) and stored in external memory on the device as illustrated in Figure 1b. The initial key is stored on-chip in eFUSE(s) or Battery-Backed Random-Access Memory (BBRAM), while keys for each successive chunk are encrypted (wrapped) in the previous ciphertext chunks. It is crucial to select a chunk size that is smaller than the minimum number of traces for a SCA. In general, fewer AES encryption operations per key offer greater security but increase the bitstream size and therefore configuration time. More details about the key rolling mechanism can be found in the reference manual [$\mathrm{ZU}_1 9$]. All assumptions and models presented within this work are based on the employment of these two countermeasures: bitstream authentication and key rolling.

2.3 AES-GCM

The ZU+ uses the AES-256 algorithm [MVM09] in Galois Counter Mode (GCM) [Dwo07] for bitstream encryption. In GCM mode, a single key is used for encryption and authentication. We focus on the encryption part of the GCM. A 96-bit Initialization Vector (IV) is



(a) (b)

Figure 1: SCA countermeasures of the of the Xilinx ZU+: (a) asymmetric authentication of the configuration data (bitstream), (b) key rolling.

randomly chosen and concatenated with a 32-bit Counter (CTR). This vector is encrypted using AES in Electronic Code Book (ECB) mode. After each encryption, the counter is incremented. The result of the AES-CTR is used as a key stream and added (bitwise XORed) with the plaintext or ciphertext to encrypt or decrypt information. Therefore, although the configuration data is actually decrypted during power-up, the AES engine itself is only used in encryption mode. We perform the side-channel analysis on the CTR part of the algorithm, because the secret key is only used in this part. Note that in the ZU+, the complete AES-GCM including counter incrementation is implemented in hardware [ZU_19].

2.4 Neural Networks for Side-Channel Analysis

In recent years there has been a growing interest in neural networks having several layers of neurons stacked upon each other, which are commonly referred to as Deep Neural Networks (DNNs) [GBC16]. They represent a particular powerful type of Machine Learning (ML) and are the privileged choice for supervised classification tasks. This means, the DNN is fed with training examples from a data set consisting of input data vectors (i.e. features) and associated outcome measurement (i.e. label). The goal is to find a suitable relationship in order to map new inputs from the test set to the correct label. Note that such a setting is similar to profiling and attack phase in classical Template Attacks (TAs) [CRR03]. There have been a large number of publications about DNNs for SCA which make primarily use of two type of ML models: Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) [CDP17, HGG18, HGG20, MPP16, PSB$^+$18, RQL18, Tim19]. Because of space restrictions we refer the reader to [GBC16] for an excellent introduction into DNNs.

3 Related Work

SCAs on older Xilinx devices have been described by Moradi et al. [MKP12, MS16]. For the first time, a CPA with 32-bit key hypothesis has been successfully implemented on GPUs. Our target, the ZU+, has a completely different AES accelerator implementation: hardware enforced GCM instead of CBC mode and four pipelined rounds instead of one round only. Moreover, due to the asymmetric authentication, a chosen data or IV attack is not possible in our attacker model.

Lauren De Meyer describes an attack on AES in CTR mode which has similarities with our work [DM19]: a large part of the state vector is constant when a limited number of traces is available. To retrieve the complete key and the nonce, four AES rounds are involved. However, the targeted implementation is software based, so that almost all intermediate steps leak with a Hamming Weight (HW) behavior. In our case, four pipelined rounds are implemented in hardware and the nonce/IV is known.

Colin O'Flynn et al. describes a non-invasive side-channel measurement method on decoupling capacitors which we use in our work too [OC13]. With this method, the magnetic field generated by small capacitors close to the chip is recorded.

F-secure found a logical vulnerability in the encryption-only boot mode of the ZU+ [F-S19]. As the execution address of the bootloader is not checked, a Return Oriented Programming (ROP) attack may be possible. In our work, we assume that the hardware root of trust is used, so that this kind of attack is not possible. Note that the IV is not authenticated in encryption-only mode. Therefore, a straightforward SCA with chosen initialization vector can be performed (e.g. by using CPA).

Recently, Ender et al. found a logical vulnerability in some 6. and 7. series Xilinx FPGAs, which allows to recover the plain bitstream - but not the key [EMP20]. The attack is based on AES-CBC Mode malleability, non-zeroing of some specific registers and non-authentication of the bitstream before decryption. In our case, AES-GCM is used and the bitstream is authenticated before decryption so that this vulnerability does not apply.

4 Side-Channel Measurements and Leakage Model

In this section, we introduce our measurements setup used to reverse engineer the hardware architecture of the AES-256 engine of the ZU+.

4.1 Assumptions

We are analyzing a security architecture that makes proper use of most security features provided by the ZU+ platform. Our security analysis relies on the following assumptions:

- An attacker cannot use the AES engine of the target device with chosen data (ciphertext, plaintext or IV). This can be enforced by secure boot and RSA authentication with hardware root of trust. Therefore, an attacker can only measure the decryption phase of an authenticated bitstream or other authenticated software items. The (initial) decryption key can be selected from different sources such as BBRAM or eFUSE depending on the authenticated boot image header [$\text{ZU}_1 9$].

- An attacker has only access to the ciphertext data of the bitstream which is stored in external, non-volatile memory. Plaintext remains secret, as the decrypted bitstream is directly used to configure the programmable logic.

- An attacker can record one specific bitstream decryption multiple times by performing a device reset. Thus, several traces related to the same operations can be recorded and averaged in order to increase the SNR.

- Since there is no masking countermeasure in the AES core itself, we do not aim to exploit higher-order leakage but only consider first-order leakage for the analysis.

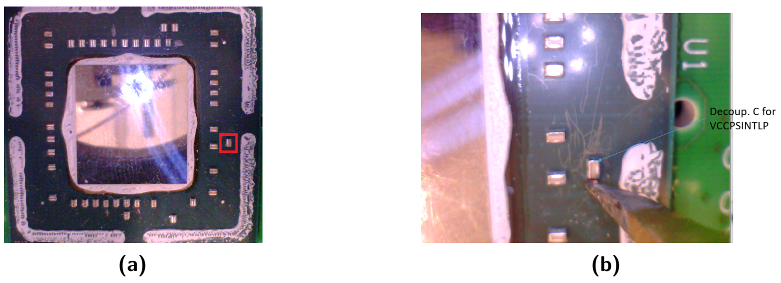


Figure 2: Decapsulated chip of the Xilinx ZU+ (a) and placement of the probe next to the decoupling capacitor (b)

4.2 Measurement Setup

Examples of side-channel leakage vectors are the power consumption or EM of an Integrated Circuit (IC). For our experiments, we performed EM measurement with local probes because it is almost completely non-invasive. Our target platform is the Xilinx ZU+ evaluation board ZCU 102 with 16 nm technology. The device itself is packaged with the flip-chip technique and we only need to remove the metal cap on the chip in order to have access to the silicon die (see Figure 2a). First, we measured the EM emanations on the silicon die itself. However, this measurement technique led to a very poor SNR. We believe this is due to the silicon thickness. Similar to the technique proposed by O'Flynn et al. [OC13], we placed our probe directly over the on-package decoupling capacitor involved in the AES power rail (VCCPSINTLP) as shown in Figure 2b. This technique leads to a much better SNR. Our measurement setup includes a Langer EMV probe ICR HV 500-75 placed on an ICS105 4-axis positioning system and a Picoscope 6404 with anti-aliasing low-pass filter. We have not used an external amplifier besides the one that is integrated in the probe. The sampling rate of the oscilloscope was set to 625 MS/s using a bandwidth of 500 MHz. We use averaging with a factor of 250 to improve the SNR. That means, 250 traces with same inputs are recorded but only the average of the traces is kept for analysis. Moreover, a GPIO is used to synchronize the traces and trigger the oscilloscope. This signal is not available in a real setup but simple techniques such as pattern matching [MOP10] can be applied for realignment. We set the frequency of the AES to about 48 MHz. A plot of an averaged EM trace covering 256 encryptions can be found in Figure 11 in the Appendix. Generally, the quality of the utilized measurement setup can be considered very high and comes with low noise.

4.3 AES Hardware Architecture

We assumed a round-based AES implementation from the timing behavior of the core. We correlated the traces with the input and output values of the AES encryption, which proofed our assumption to be correct (i.e. one encryption takes 14 clock cycles). Then, we tried different leakage models typical for AES hardware implementations using a varying number of pipeline stages and registers (1, 2, 3, 4) in a trial and error manner. We found that a correlation peak is visible for a certain register (e.g. register A in Figure 3) using the output of the same round (e.g. round 1) for encryption: $i \oplus (i+1), (i+1) \oplus (i+2), (i+2) \oplus (i+3)$, but not for: $(i+3) \oplus (i+4)$. The next correlation on the same register occurs for round 5. Thus, we can assume an architecture with four complete (i.e. 128-bit wide) AES rounds implemented in parallel and state registers between the rounds as shown in Figure 3.

At each clock cycle the four state registers are leaking information in a Hamming Distance (HD) way: a register bit leaks information at a certain point in time if this bit is toggling, i.e. the D input and Q output of the register are changing.

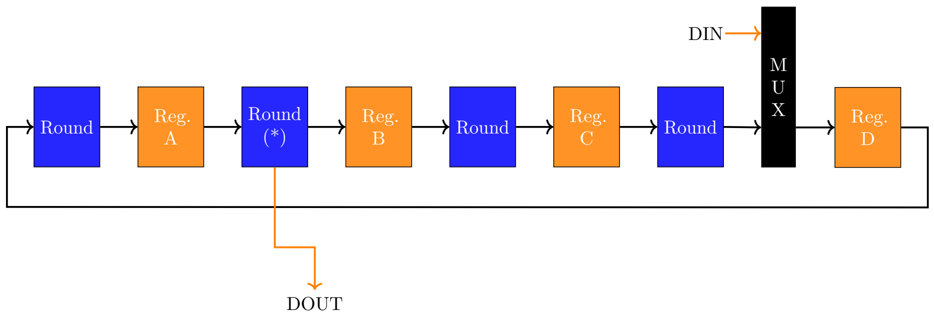


Figure 3: Presumable AES architecture of the analyzed hardware IP. Four rounds are processed in parallel, each result is stored in one of four state registers before passed to the next round.

For each register (Reg.A to Reg.D in Figure 3), the power leakage can be summarized to:

$$l = \mathrm{HW}[\mathrm{ROUNDn}(\mathrm{IV}\|Y_i) \oplus \mathrm{ROUNDn}(\mathrm{IV}\|Y_{i+1})] \tag{1}$$

Where:

- **l**: Leakage
- **ROUNDn**: State of register in AES round number n (0 to 13 in AES-256)
- **HW**: Hamming Weight
- **IV**: Initialization vector for AES-GCM (96 bit)
- $\mathbf{Y_i}$: Counter value of the GCM after i iterations, that is being concatenated with the IV (32 bit)

5 Leakage Model Exploitation

Due to the GCM mode of the AES-256 engine and the leakage model presented earlier is it not possible to extract all 32 key bytes by attacking the first two AES round only. Instead, we discovered a way to extract the complete key involving the first five AES rounds. It will be presented in the following.

5.1 GCM- and Hardware-specific attributes

SCAs against AES are usually performed on the first or last rounds because of the diffusion layer (i.e Mix Columns). Since we only have access to the AES-ECB input (IV and CTR) in our attack scenario, we focus on the first few rounds. Our HD leakage model involves two consecutive block encryptions. In GCM or CTR mode, the only difference between two consecutive AES-ECB input vectors is the counter value. This counter consists of four bytes, the least significant byte changes at each incrementation (CTR_0, see Figure 4). The second byte is only changing every $2^8 = 256$ encryptions (CTR_1); the third byte is changing after $2^{16} = 65536$ encryptions (CTR_2), and the fourth byte is changing every 2^{24} encryptions (CTR_3).

In standard side-channel analysis, it is assumed that the data is randomly distributed which is obviously not the case here: only one byte (least significant byte of counter) out of the 16 input vector bytes is changing at each encryption and is leaking information. This is why only one key byte can be extracted when focusing on the first AES round. In order to extract the remaining key bytes, more rounds have to be considered.

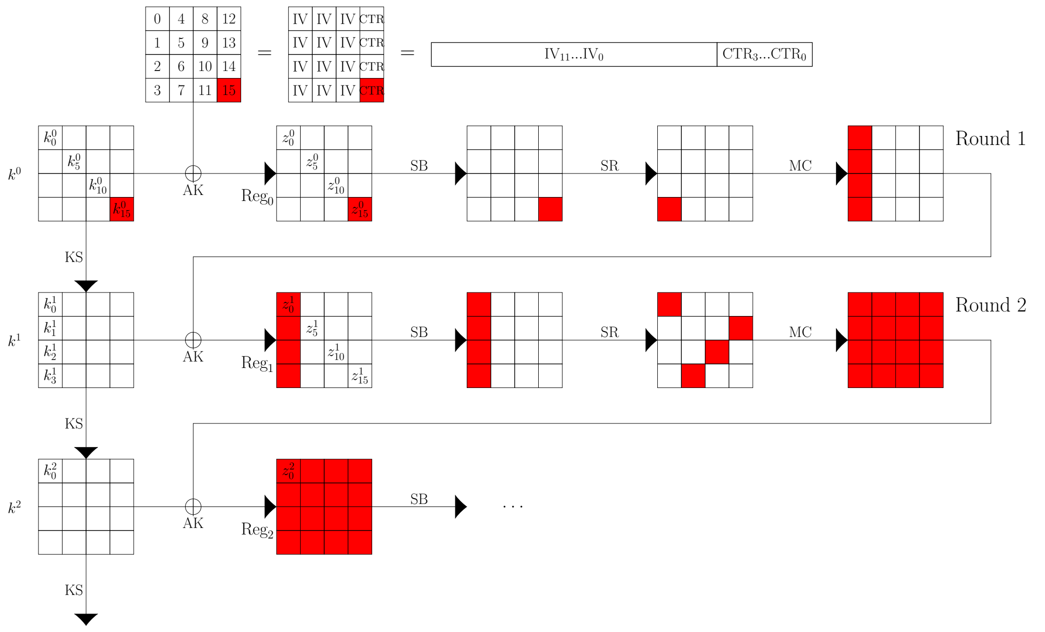


Figure 4: Sketch of AES-ECB with plaintext as used in this AES-GCM implementation. Each round ends with the state register Reg_i, i.e., Round 1 ends after adding subkey k^1 and storing the result in Reg_1. The changes when only CTR_0 increases every encryption are highlighted in red. Abbreviations used: AK = Add Round Key, SB = S-Box (Substitution Box), SR = Shift Row, MC = Mix Columns.

5.2 First Round

The goal in first AES round is to extract the key byte associated with the least significant counter byte (15th byte of first subkey K^0_{15}, highlighted red in the upper left corner of Figure 4). We can define the leakage associated with register Reg_1 in Figure 4 as:

$$l^1 = HW[ROUND1(IV \| Y_i) \oplus ROUND1(IV \| Y_{i+1})] \tag{2}$$

We focus on the first state byte (z^1_0). The same principle applies for the state bytes 5, 10 and 15 modified during the first round.

$$\begin{aligned} l^1_0 =& HW[k^1_0 \oplus 2 \times SB(z^0_0(IV\|Y_i)) \oplus 3 \times SB(z^0_5(IV\|Y_i)) \\ & \oplus 1 \times SB(z^0_{10}(IV\|Y_i)) \oplus 1 \times SB(z^0_{15}(IV\|Y_i)) \\ & \oplus k^1_0 \oplus 2 \times SB(z^0_0(IV\|Y_{i+1})) \oplus 3 \times SB(z^0_5(IV\|Y_{i+1})) \\ & \oplus 1 \times SB(z^0_{10}(IV\|Y_{i+1})) \oplus 1 \times SB(z^0_{15}(IV\|Y_{i+1}))] \end{aligned} \tag{3}$$

Where k^i_n is the subkey i with index n, SB the byte substitution (S-Box) operation, and $\times$ is the Galois Field multiplication used in the mix column operation. This is achieved by a left multiplication with the matrix:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \tag{4}$$

Since the IV values corresponding to the input bytes 0 to 11 are constant, we can reduce the leakage term to:

$$\begin{aligned} l_0^1 &= HW[SB(z_{15}^0(IV\|Y_i)) \oplus SB(z_{15}^0 \oplus (IV\|Y_{i+1}))] \\ &= HW[SB(k_{15}^0 \oplus (IV\|Y_i)) \oplus SB(k_{15}^0 \oplus (IV\|Y_{i+1}))] \end{aligned} \tag{5}$$

k_{15}^0 can be recovered using a SCA with 8-bit hypothesis space.

5.3 Second Round

Goal of the analysis for this round is to calculate some constants related to the subkey bytes $k_i^2, i = 0, \ldots, 15$. With this information the next (third) round can be analyzed. We define the leakage for the second round in the register Reg_2 in Figure 4 to:

$$l^2 = HW[ROUND2(IV\|Y_i) \oplus ROUND2(IV\|Y_{i+1})] \tag{6}$$

Again, we focus on the first state byte (z_0^2). The same principle applies for all other bytes.

$$\begin{aligned} l_0^2 = HW[&k_0^2 \oplus 2 \times SB(z_0^1(IV\|Y_i)) \oplus 3 \times SB(z_5^1(IV\|Y_i)) \\ &\oplus 1 \times SB(z_{10}^1(IV\|Y_i)) \oplus 1 \times SB(z_{15}^1(IV\|Y_i)) \\ &\oplus k_0^2 \oplus 2 \times SB(z_0^1(IV\|Y_{i+1})) \oplus 3 \times SB(z_5^1(IV\|Y_{i+1})) \\ &\oplus 1 \times SB(z_{10}^1(IV\|Y_{i+1})) \oplus 1 \times SB(z_{15}^1(IV\|Y_{i+1}))] \end{aligned} \tag{7}$$

The Least Significant Byte (LSB) of the counter (i.e. byte 15 of the AES input vector) is changing after each encryption; the second counter byte (i.e. byte 14 of the AES input vector) is changing each 256 encryptions (i.e. stays the same for 255 encryptions), and so on that:

• In 255 of 256 cases:	$(IV\|Y_i)_{14} = (IV\|Y_{i+1})_{14}$
• In $(256^2 - 1)/256^2$ cases:	$(IV\|Y_i)_{13} = (IV\|Y_{i+1})_{13}$
• In $(256^3 - 1)/256^3$ cases:	$(IV\|Y_i)_{12} = (IV\|Y_{i+1})_{12}$

Moreover, all IV bytes (0 to 11) are constant. Therefore, we can approximate the leakage to:

$$l_0^2 \approx HW[2 \times SB(z_0^1(IV\|Y_i)) \oplus 2 \times SB(z_0^1(IV\|Y_{i+1}))] \tag{8}$$

$$\begin{aligned} z_0^1(IV\|Y_i) =&k_0^1 \oplus 2 \times SB((IV\|Y_i)_0 \oplus k_0^0) \oplus 3 \times SB((IV\|Y_i)_5 \oplus k_5^0) \\ &\oplus 1 \times SB((IV\|Y_i)_{10} \oplus k_{10}^0) \oplus 1 \times SB((IV\|Y_i)_{15} \oplus k_{15}^0) \end{aligned} \tag{9}$$

We merge all constants (including the key byte) into one constant byte α_0, so that:

$$z_0^1(IV\|Y_i) = \alpha_0 \oplus 1 \times SB((IV\|Y_i)_{15} \oplus k_{15}^0) \tag{10}$$

The leakage is now:

$$\begin{aligned} l_0^2 \approx HW[&2 \times SB(\alpha_0 \oplus 1 \times SB((IV\|Y_i)_{15} \oplus k_{15}^0)) \\ &\oplus 2 \times SB(\alpha_0 \oplus 1 \times SB((IV\|(Y_{i+1})_{15} \oplus k_{15}^0))] \end{aligned} \tag{11}$$

We know k^0_{15} from the first round analysis. The AES input is known too, so that we can extract the constant α_0 (one byte) using an attack with 256 hypotheses. Extending the analysis to all columns of the second round, three further constants α_1 to α_3 can be extracted using the same technique:

$$\begin{aligned}\alpha_0 = k^1_0 &\oplus 2 \times SB((IV\|Y_i)_0 \oplus k^0_0)\\ &\oplus 3 \times SB((IV\|Y_i)_5 \oplus k^0_5)\\ &\oplus 1 \times SB((IV\|Y_i)_{10} \oplus k^0_{10})\end{aligned} \tag{12}$$

can be extracted with byte 0 to byte 3 of the output of round 2.

$$\begin{aligned}\alpha_1 = k^1_1 &\oplus 1 \times SB((IV\|Y_i)_0 \oplus k^0_0)\\ &\oplus 2 \times SB((IV\|Y_i)_5 \oplus k^0_5)\\ &\oplus 3 \times SB((IV\|Y_i)_{10} \oplus k^0_{10})\end{aligned} \tag{13}$$

can be extracted with byte 12 to byte 15 of the output of round 2.

$$\begin{aligned}\alpha_2 = k^1_2 &\oplus 1 \times SB((IV\|Y_i)_0 \oplus k^0_0)\\ &\oplus 1 \times SB((IV\|Y_i)_5 \oplus k^0_5)\\ &\oplus 2 \times SB((IV\|Y_i)_{10} \oplus k^0_{10})\end{aligned} \tag{14}$$

can be extracted with byte 8 to byte 11 of the output of round 2.

$$\begin{aligned}\alpha_3 = k^1_3 &\oplus 3 \times SB((IV\|Y_i)_0 \oplus k^0_0)\\ &\oplus 1 \times SB((IV\|Y_i)_5 \oplus k^0_5)\\ &\oplus 1 \times SB((IV\|Y_i)_{10} \oplus k^0_{10})\end{aligned} \tag{15}$$

can be extracted with byte 4 to byte 7 of the output of round 2.

5.4 Third Round

Goal of the analysis of the third round is to extract 16 constants related to the subkey bytes, which enables to calculate the input vector values of the fourth round for 256 consecutive AES encryptions. The leakage for the third round is (byte 0 only, first column):

$$\begin{aligned}l^3_0 \approx HW[k^3_0 &\oplus 2 \times SB(z^2_0(IV\|Y_i)) \oplus 3 \times SB(z^2_5(IV\|Y_i))\\ &\oplus 1 \times SB(z^2_{10}(IV\|Y_i)) \oplus 1 \times SB(z^2_{15}(IV\|Y_i))\\ &\oplus k^3_0 \oplus 2 \times SB(z^2_0(IV\|Y_{i+1})) \oplus 3 \times SB(z^2_5(IV\|Y_{i+1}))\\ &\oplus 1 \times SB(z^2_{10}(IV\|Y_{i+1})) \oplus 1 \times SB(z^2_{15}(IV\|Y_{i+1}))]\end{aligned} \tag{16}$$

$$\begin{aligned}z^2_0(IV\|Y_i) = k^2_0 &\oplus 2 \times SB(z^1_0(IV\|Y_i)) \oplus 3 \times SB(z^1_5(IV\|Y_i))\\ &\oplus 1 \times SB(z^1_{10}(IV\|Y_i)) \oplus 1 \times SB(z^1_{15}(IV\|Y_i))\end{aligned} \tag{17}$$

The other terms can be calculated similarly. We now take **a set of 256 consecutive traces**, in which only the counter LSB (byte 15) is changing such that $z^1_n(IV\|Y_i)$ is constant for $n = 0...14$. Again, we concatenate the constant terms into γ_0 and rewrite $z^2_0(IV\|Y_i)$ into:

$$\begin{aligned} z_0^2(IV\|Y_i) =& \gamma_0 \oplus 2 \times SB(z_0^1(IV\|Y_i)) \\ =& \gamma_0 \oplus 2 \times SB(\alpha_0 \oplus SB((IV\|Y_i)_{15} \oplus k_{15}^0)) \end{aligned} \tag{18}$$

$$\begin{aligned} \gamma_0 = k_0^2 &\oplus 3 \times SB(z_5^1(IV\|Y_i)) \\ &\oplus 1 \times SB(z_{10}^1(IV\|Y_i)) \\ &\oplus 1 \times SB(z_{15}^1(IV\|Y_i)) \end{aligned} \tag{19}$$

The three other terms of l_0^3 can be rewritten in the same manner with three new constants γ_5, γ_{10} and γ_{15}. As far as the three remaining columns of the third round (i.e. l_4^3 to l_{15}^3) are concerned, 12 other constants γ_n, $n = 1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14$ are created in the same way. Since we know z_{15}^0 from the first round and all α_n, $n = 0...3$ constants from the second round, we can extract the 16 γ_n, $n = 0...15$ constants with four SCAs using a 32-bit hypothesis, one on each column of the third round.

At the end of the third round analysis, we are able to calculate the output state vector of this round for 256 consecutive AES input vectors depending only on the corresponding subkey k_n^3, $n = 0...15$. Here we show it for the first byte as an example:

$$\begin{aligned} z_0^3(IV\|Y_i) = k_0^3 &\oplus 2 \times SB(z_0^2(IV\|Y_i)) \oplus 3 \times SB(z_5^2(IV\|Y_i)) \\ &\oplus 1 \times SB(z_{10}^2(IV\|Y_i)) \oplus 1 \times SB(z_{15}^2(IV\|Y_i)) \end{aligned} \tag{20}$$

Alternatively, in extended form:

$$\begin{aligned} z_0^3(IV\|Y_i) = k_0^3 &\oplus 2 \times SB(\gamma_0 \oplus 2 \times SB(\alpha_0 \oplus 1 \times SB((IV\|Y_i)_{15} \oplus k_{15}^0))) \\ &\oplus 3 \times SB(\gamma_5 \oplus 1 \times SB(\alpha_3 \oplus 2 \times SB((IV\|Y_i)_{15} \oplus k_{15}^0))) \\ &\oplus 1 \times SB(\gamma_{10} \oplus 2 \times SB(\alpha_2 \oplus 3 \times SB((IV\|Y_i)_{15} \oplus k_{15}^0))) \\ &\oplus 1 \times SB(\gamma_{15} \oplus 1 \times SB(\alpha_1 \oplus 1 \times SB((IV\|Y_i)_{15} \oplus k_{15}^0))) \end{aligned} \tag{21}$$

In this way, we know the input vector values of the fourth round for 256 consecutive AES inputs (e.g. for the first 256 AES computations), depending only on the corresponding subkey.

5.5 Fourth / Fifth Round

Knowing the input vectors of the fourth round, four (one for each column) SCAs with 32-bit hypothesis can be performed on this round in order to extract the complete subkey k_n^3, $n = 0...15$. In the same manner, the next subkey k_n^4, $n = 0...15$ can be extracted using the fifth round. Knowing two consecutive subkeys enables the calculation of the AES-256 key.

6 Attacks

In this section, we present the experimental setup used to test our attack model and discuss the results.

6.1 Data Set

Our base data set is composed of 200 000 traces (after averaging) with random keys and IVs. Using more traces (e.g. 300 000 traces or even more) gave no improvements in our experiments. 75 % of the data set has been used for training/profiling, 24 % for validation (i.e. to measure the performance on unseen traces), and 1 % as attack traces. Please note that we use random keys and IVs to create as much toggling entropy as possible for the profiling, and to perform the key recovery with several different key values. Our attack model requires to extract the 256-bit AES key with at most 256 encryptions. Therefore, we acquired traces with the EM emanations of 256 successive AES-GCM encryptions (counter values 0 to 255) according to Section 4.2. Thus, for the rest of the paper one trace corresponds to 256×250 AES operations due to our averaging pre-processing. Each trace contains 15 625 sample points which represents approximately 1200 clock cycles. We checked the traces using CPA and known leakage hypothesis. The correlation coefficients ρ between 50 000 traces and our leakage model (i.e. 32 bits of the state) are shown in Table 1. Please note that because the correlation values vary for different counter values, we performed the correlation for ten different counter pairs per round leakage and calculated the mean correlation. It becomes clear that the leakage of later rounds is more difficult to exploit than the leakage of the first round. For example, the correlation for round three is roughly halved compared to round one. We assume this is because the algorithmic noise is higher due to the inherent parallelism of the investigated AES-GCM implementation, and the fact that more state bytes toggle in round three (and later) than in the first two rounds as illustrated in Figure 4.

Table 1: Mean correlations for the leakages corresponding to Section 5

	Round 1	Round 2	Round 3	Round 4	Round 5
Correlation ρ	0.153	0.117	0.078	0.070	0.073

6.2 Baseline

As a first step, we have tried to recover the AES key using a combination of Linear Discriminant Analysis (LDA) pre-processing and CPA. LDA is an established method to increase the SNR by projecting the traces into a smaller dimension which maximizes the intra-class variance [BGH+15]. For each counter value (i.e. leakage), we selected 50 sample points from the traces. The location of these samples has been determined earlier by correlation with known key. We selected 50 points since a peak in the correlation analysis takes roughly the same amount of samples. As mentioned in the previous paragraph, our traces cover encryptions using a CTR value from zero to 255. The remaining 15 input bytes (IV + three bytes of CTR) remain static. However, due to the pipeline structure of the AES engine with four rounds processed in parallel, only three-quarter of the CTR values produce exploitable leakage starting from $CTR = 3$. This is because the registers are filled with values from round $i + 3$ for every fourth clock cycle, which produces a non-exploitable HD leakage. Thus, there are only $256 \times \frac{3}{4} - 2 = 190$ encryptions that can be effectively used for the attack per round. An individual LDA model has been fit on the reduced (training) traces for each CTR leakage and for each round. From a computational point of view, this might not the best solution but no realignment is needed by following this strategy. Next, the LDA models have been used to compress the attack traces into a single data point per leakage. Finally, ten independent CPA attacks have been performed with the optimized attack set (attack traces have been randomly picked).

We have applied two different power leakage models: The regular HD as described in Section 4.3 and a Linear Regression (LR) model (aka stochastic approach) proposed by Schindler et al. [SLP05]. As evaluation metric we have used the well-known Key Guessing Entropy (KGE). The KGE – also known as key rank – refers to the amount of key guesses that have to be made in order to get a correct key byte while having a fixed amount of attack traces [SMY09]. We show the number of traces to achieve a KGE smaller or equal to one in Table 2. From there, it can be observed that only the attacks in the first round were successful. The attacks against later rounds failed due to the significant lower SNR as discussed in Section 6.1. Since conventional state-of-the art methods are not enough for our target, we decided to change our strategy and move to sophisticated DL attack methods. These are described in the upcoming.

Table 2: Attack results for baseline methods (# traces for $KGE \leq 1$)

	Round 1	Round 2	Round 3	Round 4	Round 5
CPA-LDA (HD model)	170	-	-	-	-
CPA-LDA (LR model)	130	-	-	-	-

6.3 Methodology

We perform our DL-based evaluation based on the Correlation Optimization (CO) scheme introduced by Robyns et al. [RQL18] at CHES 2019. It is shortly introduced in the following before we present two extensions to the original CO scheme.

6.3.1 Correlation Optimization

The basic idea of CO is to teach a DNN to produce an encoding of the input data (i.e. the traces) that maximizes the Pearson correlation with a hypothetical power consumption (i.e. the leakage l in this paper). It can therefore be considered as an extension of classical CPA with an additional profiling/learning phase. Key component of CO is the correlation loss function $\mathcal{L}_{CO}(l, \theta)$ that computes the correlation coefficient between the leakage and the encoding θ produced by the DNN over a batch of D traces. It is defined as:

$$\mathcal{L}_{CO}(l,\theta) = 1 - \frac{cov(l,\theta)}{\sigma_l \sigma_\theta + \epsilon} = 1 - \frac{\sum_{i=1}^{D}[(l_i - \bar{l})(\theta_i - \bar{\theta})]}{\sqrt{\sum_{i=1}^{D}(l_i - \bar{l})^2 \sum_{i=1}^{D}(\theta_i - \bar{\theta})^2} + \epsilon} \tag{22}$$

where ϵ denotes a small number to prevent division by zero. Note that the batch size D has to be larger than in a classification setting in order to receive an appropriate amount of correlation (e.g. 512). Once the training process has been finished, the DNN is used to create optimized encoding of the attack traces to perform a regular CPA.

We chose the CO attack due to several reasons: First, our leakage model requires to extract the secret key within 256 encryptions. This makes the problem a good candidate for a profiled SCA using TAs or DNNs. Second, a 32-bit hypothesis is needed in round three and later which requires evaluating more than four billion key guesses. The CO scheme automatically encodes the samples of a trace into a single value such that it can be directly used for 32-bit CPA running on a GPU. We also experimented with DNNs in a standard classification mode using the categorical cross-entropy loss function and Gaussian TAs as done by many researchers before (see Section 2.4). However, we achieved less promising results because of the large imbalance in the data set. Since we use a 32-bit HD power leakage model as shown in Section 5, some classes are very sparsely represented in the data set, due to the binomial distribution produced by the HW function. Even advanced re-balancing methods such as Synthetic Minority Oversampling Technique with Edited Nearest Neighbor (SMOTE) [PHJ+18] could not improve the classification accuracy as needed. Therefore, we believe that the CO approach is a suitable method to assess the SCA resistance of the ZU+ encryption engine.

6.3.2 Bitwise Correlation Loss

The first extension to the the regular CO scheme is based on bitwise correlations. Instead of applying the HW to the XOR of the values that are stored in consecutive clock cycles in a register, the bit flips are directly used as leakage labels l_{bit}. Consequently, the total loss is calculated by combining the correlations for each bit to:

$$\mathcal{L}_{CO-BIT}(l_{bit}, \theta_{bit}) = \sum_{i=1}^{B} |\mathcal{L}_{CO}(l_{bit}, \theta_{bit})^{(i)}| \tag{23}$$

where B denotes the number of bits in the leakage vector (32 in our case). Note that all leakage bits have a corresponding encoding $\boldsymbol{\theta}_{bit}^{(i)}$ that is produced by the DNN. Therefore, the complexity of the DNN model is slightly increased but we can take advantage of the individual leakage of each flip-flop in the registers similar to a multi-bit CPA [DPRS11].

6.3.3 Weighted-Bit Correlation Loss

The second extension we propose is related to the stochastic approach by Schindler et al. and a bit-dependent leakage model [SLP05]. Although the regular HD model already leads to a very good approximation of the power consumption of a cryptographic hardware implementation, it can be further improved. The output capacitances and thus the power consumption of individual bits is different, due to varying wire lengths between the cells processing and storing the data [MOP10]. In order to account for this effect, some weighting coefficients c_i can be given to each bit. Assuming that R is a B-bit register with an input RD and a registered output RQ, the weighted-bit leakage l_{W-BIT} can be defined as:

$$l_{w-bit} = \eta + \sum_{i=1}^{B} c_i \times (RQ \oplus RD) = \eta + \sum_{i=1}^{B} c_i \times l_{bit}^{(i)} \tag{24}$$

wherein η refers to a non-data dependent noise factor. LR is used in the stochastic model to derive the coefficients (see baseline attack in Section 6.2). In our approach, we let the DNN learn the coefficients. When comparing Equation (24) with the basic structure of a DNN [GBC16], one can notice that l_{W-BIT} can be approximated by a perceptron with linear activation function using l_{bit} as input, the weights $w_i, i = 1, \ldots, B$ as coefficients c_i, and η as bias term. Thus, we have defined a single-neuron MLP that receives the bitwise labels and outputs l_{w-bit}, which is then used in the loss function instead of the HD labels (l in Equation (22)). A schematic overview is given in Figure 5. Such a weighted-bit correlation optimization ($CO-W-BIT$) can be considered as an approach that creates optimized encodings not only for the power traces, but also for the leakage hypothesis in order to enhance the correlation coefficient of a CPA.

6.4 DNN Models

Two DNN models are used for the analysis: an MLP with 2 hidden layers, and a CNN composed of 3 blocks with alternating Convolutional (CONV) and Pooling (POOL) operations preceded by an additional dense layer. The basic architectures of the networks have been inspired by related work [HGG20, KPH+19, RQL18], while fine-tuning of the

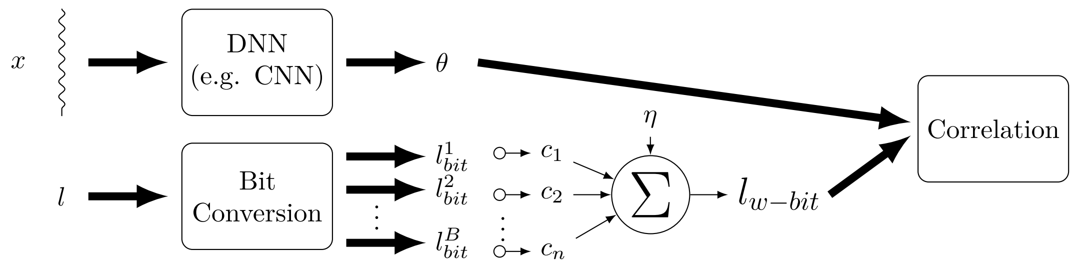


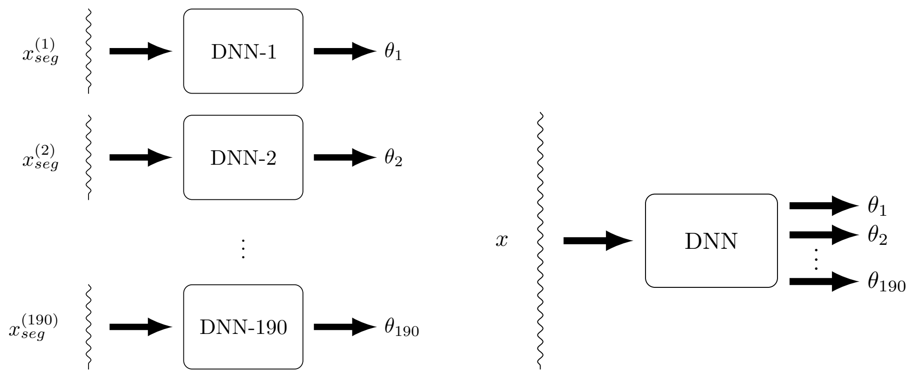
Figure 5: Correlation optimization using a small MLP to approximate the bitwise weighting coefficients for the leakage l, while the power traces x are encoded by a DNN. During training, the correlation loss according to (22) is used to update the weights of both networks in parallel.

hyperparameters has been done with Bayesian optimization using the Tree-structured Parzen Estimator (TPE) algorithm [BBBK11]. The exact model configurations along with hyperparameters search space can be found in the Appendix of the paper. As a preprocessing step for the MLP, we standardized all traces to have zero mean and unit variance. In all experiments, we trained the networks using Adam optimizer and a learning rate of 0.001 for CNNs and 0.0001 for MLPs. The batch size parameter D has been set to a value of 256. We have implemented our attack framework in Python using the open-source DL frameworks *Keras* [C+15] and *TensorFlow* [AAB+15]. Training of the DNN models has been performed on two Nvidia Tesla V100 GPUs.

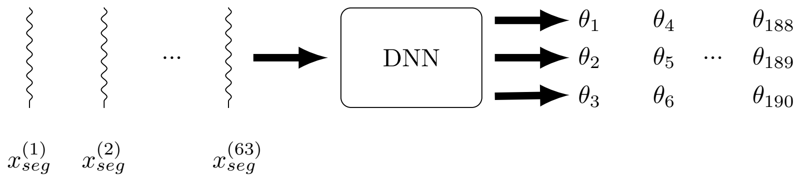
A straightforward approach is to build an individual DNN model for every leaking operation, i.e., training 190 different networks. During key recovery, the same attack trace is given to all models to receive optimized encodings. The advantage of this *model-per-counter* setting as shown in Figure 6a is that shorter traces can be used. Only the sample points corresponding to the counter value is necessary to train one network. An example of such a trace segment is shown in Figure 12 in the Appendix. Using smaller traces reduces the training time significantly. Nevertheless, it can take roughly one to day to train 190 networks on a high-end GPU for a single round attack.

An alternative method is to use a single *output-per-counter model* that produces the encodings for all leaking operations at once as illustrated in Figure 6b. Such a network is more complex in terms of trainable parameters since the complete traces are used as input and the output dimension is increased from one to 190. However, it only has to be trained for a single time. A third method is to train a single DNN model using segmented traces, which outputs the encodings for three leaking operations per input as shown in Figure 6c. This is possible since the leakage of three consecutive counters occurs within a very small period (three clock cycles). The *triple-output-model* is a hybrid of the two aforementioned approaches and combines the advantage of fast training time and reasonable complexity.

We implemented and evaluated all three models using the regular CO loss function as defined in Equaiton (22). For the multi-output models, the correlation loss is calculated for every DNN output individually and eventually summed up (i.e. in Figure 6c, the loss is calculated using three outputs while in Figure 6b, 190 outputs are used.) Furthermore, we tested the bitwise correlation loss (Section 6.3.2) as well as the weighted-bit loss (Section 6.3.3) in combination with the model-per-counter method.



(a) Individual model per counter **(b)** Output per counter model



DNN
θ_1 θ_4 θ_{188}
θ_2 θ_5 θ_{189}
θ_3 θ_6 θ_{190}
$x_{seg}^{(1)}$
$x_{seg}^{(2)}$
$x_{seg}^{(63)}$

(c) Triple output model

Figure 6: Approaches to attack the ZU+ encryption engine. In the *model-per-counter* setting (a), the traces are split up in several segments x_{seg} representing the leakages of a single counter value. For each counter, an individual DNN is created to produce optimized encodings θ. The *output-per-counter model* (b) is fed with complete traces and outputs all encodings in a single pass. Furthermore, the *triple-output-model* (c) receives trace segments corresponding to three counter leakages and outputs the respective encodings.

6.5 Results

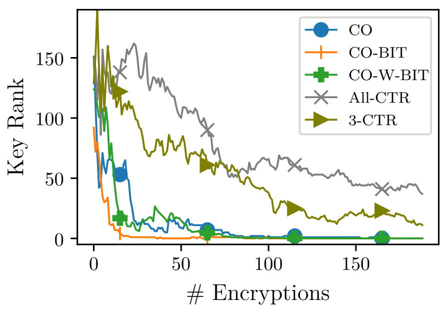
The attack complexity differs between the different AES rounds as shown in Section 5. In the following, we present in detail the methodology and results of each round.

6.5.1 First Round

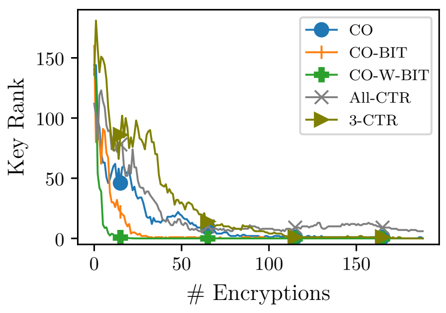
We know from Section 5.2 that we are only able to extract key byte fifteen in the first round. The KGE for the considered models and loss functions are illustrated in Figure 7. Each curve has been calculated by determining the mean KGE over ten randomly chosen traces from the attack set. As stated before, each trace contains 256 AES encryptions but only 190 of them can be used for the attack. Some of our models are still able to recover the fifteenth key byte even under such restricted conditions.

Figure 7 shows that the multi-output models (gray and dark green curves denoted as *ALL-CTR* and *3-CTR*) are generally outperformed by the model-per-counter approaches. However, the difference is smaller with the CNN than with the MLP. This is because CNNs are more effective to extract features from larger input dimensions than MLPs. Comparing the different loss functions, it becomes clear that the bitwise-correlation loss (CO-BIT) and the weighted-bit correlation (CO-W-BIT) obtain the best KGE among all attacks and successfully reveal the correct key byte with less than 50 encryptions on the CNN model. Please note that these results are substantially better than the baseline attacks presented in Section 6.2.

The training times for the two model types are given in Table 3. For the triple-output model we divided 5000 of our profiling traces – each containing 15 625 sample points – into



(a) MLP



(b) CNN

Figure 7: Attack results for the first round

63 overlapping segments with 300 sample points. Each segment contains the leakage for three counters. Therefore, the model was trained with $5000 \times 63 = 315\,000$ examples. The MLPs require less time to iterate completely over the training data set (i.e. an epoch), as they contain less trainable parameters than the CNNs.

For some approaches, however, the CNN models converge faster than the MLP. We stopped the training process after the validation loss has not decreased for more than 20 epochs.[1] This led to varying training times per model. Comparing the timing overhead for the different loss functions, one can notice that the CO-BIT approach takes longest as a correlation is calculated for each leakage bit individually. The multi-output models are very efficient in terms of training time since they only require training of a single model. However, their attack performance – especially with the MLP – is not as good as using an individual model per counter.

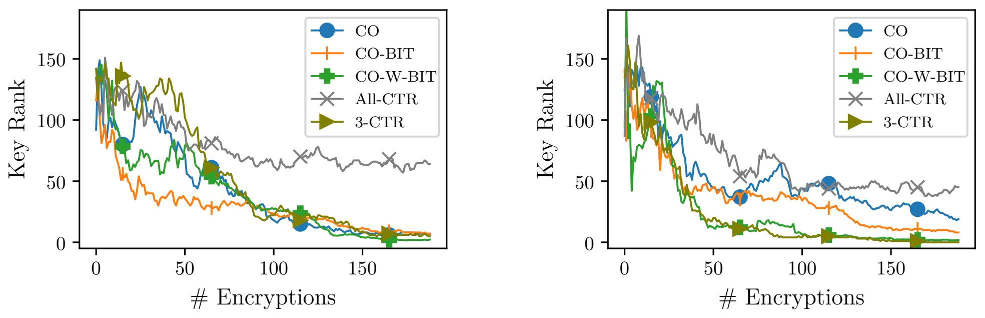
Table 3: Training times comparison

	CO	CO-BIT	CO-W-BIT	All-CTR	3-CTR
Number of training traces	200 000	200 000	200 000	200 000	315 000
Input layer size	200	200	200	15625	300
Time per epoch (MLP/CNN)	3s/9s	10s/15s	5s/11s	55s/86s	42s/80s
Total training time (MLP/CNN)	7.3h/18.5h	29.3h/26h	14.5h/20.6h	3h/1.5h	0.5h/1.2h

6.5.2 Second Round

The aim for the second round attack is to extract four 8-bit constants $(\alpha_0, \ldots, \alpha_3)$ that are needed to calculate the leakage of round three. The attack complexity is the same as in the first round. However, the algorithmic noise is higher as discussed earlier in the paper. The increased noise level leads to a lower correlation and thus to a worse KGE compared to the attacks in the first round. Nevertheless, several techniques are able to extract the alphas using 190 encryptions with a KGE smaller than two as shown in Figure 8. The CNN using the weighted-bit loss function and the triple-output CNN perform best among the attacks, followed by the CO-Bit models. This demonstrates the benefit of our proposed extensions of the classical CO approach. The multi-output networks that predict the leakage of all counter values only reach a mean key rank of 50 with the CNNs and even higher with the MLP. For this reason, we do not consider this approach for the attacks against later AES rounds.

[1] We have not used learning rate decay since the ADAM optimizer intrinsically handles learning rate optimization [KB14]



(a) MLP **(b)** CNN

Figure 8: Attack results for the second round

6.5.3 Third Round

In the third round, 16 gamma constants γ_n, $n = 0...15$ related to the third round key have to be extracted. This is done by four attacks, each using a 32-bit hypothesis due to the mix columns operation. The complexity to recover the gamma constants is therefore much higher than recovering the alphas in the round before, i.e., the range of possible values is increased from 256 to more than four billion candidates.

In order to parallelize the calculation of the correlation between the encoded traces and the hypothetical leakage guesses, we ran the attacks on GPU using the Nvidia CUDA framework [Coo12]. By using CUDA, the recovery phase was reduced from several weeks to roughly 6 h. However, since the SNR is even smaller than in the second round, none of the attack methods have been able to recover the gammas using 190 encryptions. The remaining KGE has been in a range of a few thousand and several million.

During the training phase, we noticed a mean validation loss of 0.835 for the CO-W-BIT approach. The mean correlation coefficient can thus be calculated to: $1 - 0.835 = 0.165$. Although the correlation is more than twice as high as with a standard correlation attack (i.e. 0.078 according to Section 6.1), it is not sufficient for a recovery of the gammas with 190 encryptions. According to the rule of thumb given by Mangard et al. [MOP10], the number of required traces for a successful attack n_A can be roughly estimated to:

$$n_A \approx \frac{28}{\rho^2}. \tag{25}$$

, if $|\rho \leq 0.2|$. We thus believe that at least 1000 encryptions are needed in our attack setup to recover the gammas with sufficient certainty.

6.5.4 Fourth/Fifth Round

The attacks in the fourth and fifth round directly target the corresponding subkeys. As in the round before, four 32-bit attacks are necessary to extract all 16 subkey bytes of a round. We were again not able to perform a successful key recovery within 256 consecutive encryptions (with 190 effective leakages). However, if an attacker would be able to recover the gamma constants in round three with a high probability, more encryptions than only 256 can be used for the attack against round four and five. This is because the round keys are static over all encryptions under the same AES-256 key, while the gammas from round three (and also the alphas from round two) are only stable for 256 consecutive encryptions. Therefore, an attacker can just repeat the attacks against the second and third round to extract multiple sets of alphas and gammas (e.g. ten sets) and use them to have more

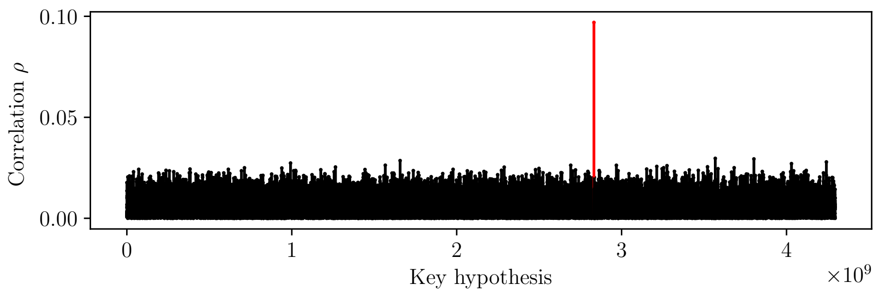


Figure 9: Result for the CNN attack against one column of the fourth round key using 19 000 encryptions. The correct subkey hypothesis is shown in red.

traces in round four. In order to demonstrate this, we acquired a smaller second data set of 2000 traces (i.e. covering $2000 \times 256 = 51\,200$ encryptions) using a static AES key. We trained the CNN with bit-dependent leakage model (CO-W-BIT) on the training data set composed of 200 000 traces with random keys. Then, we used this model to perform ten independent 32-bits attacks against one column of the fourth round key, each time using 100 randomly chosen traces (i.e. having 19 000 effective encryptions) from the data set with static key. The absolute correlation values over a subset of the $2^{32} = 4\,294\,967\,296$ key guesses for a single attack are illustrated in Figure 9. The correlation plots for the other nine attacks look very similar. In general, the maximum correlation for the correct key hypothesis has been around 0.1. This means that approximately 3000 encryptions are needed to extract the fourth round key according to Equation (25). The same technique has been applied to extract the fifth round key, producing similar results. The MLP and the other correlation techniques (CO, CO-BIT, 3-CTR) produced a slightly worse maximal correlation coefficient. We omit the corresponding plots due to space limitations.

7 Recommendation for Key Rolling Parameter

The actual value of the key rolling parameter (i.e. how many bitstream blocks are encrypted under the same key) has a major impact on the size of the configuration image and the boot time of the ZU+. Figure 10 shows the scaling of the bitstream size and boot time with varying key rolling parameters. An unencrypted bitstream of the ZU9EG/EC device series has a size of roughly 26.5 MB. Selecting a key rolling value of 100 or higher increases the encrypted bitstream size by not more than 4 %, which means that approximately 1.7 million AES operations are performed during boot-up. A value of eight doubles the bitstream size. The boot time for different configuration sizes can be estimated using an Excel macro provided by Xilinx [ZU$_1$7*b*]. The corresponding values for a ZU9EG/EC device using QSPI dual boot with enabled RSA authentication and bitstream encryption are also plotted in Figure 10. From there, one can notice that the boot time scales linear with the configuration size, i.e., a key rolling value of five increases the boot-up time by more than 400 %, while values larger than 40 impact the boot-up the by less than 10 %.

We know from the previous section that a complete extraction of the key has not been successful within 256 encryptions. Thus, a straightforward choice would be to set the key rolling parameter to a value of around 200. However, the fifteenth key byte k^0_{15} could be recovered with less than 50 encryptions in the attacks on the first round. We also demonstrated that extraction of the alpha constants in the second round is possible with less than 190 encryptions. By extracting four different sets of alphas (i.e. 16 constants), an equation system with 15 variables can be solved. This allows to extract information

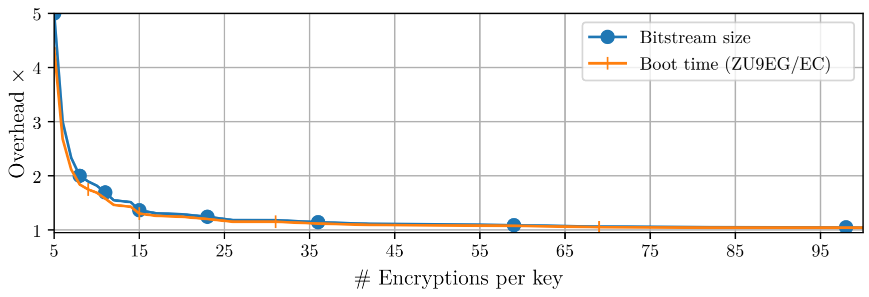


Figure 10: Normalized bitstream size and boot time for different key rolling values. The numbers have been calculated using the scripts provided at [ZU$_1$7*a*] and [ZU$_1$7*b*].

about seven key bytes $(k_0^0, k_5^0, k_{10}^0, k_0^1, k_1^1, k_2^1, k_3^1)$.[2] In round three, we have not been able to recover the gamma constants using traces that contain 190 encryptions. However, it might be possible with an advanced version of the CO scheme or another DL-based attack. Approximately 3000 traces were necessary to extract the forth and the fifth round key.

Please note that we have used a single ZU+ device for profiling and attack in our setup. Our analysis can thus be considered as a worst-case scenario since in an actual attack, the profiling has to be done on another device with disabled secure boot. This can lead to portability issues when the leakage behaviour of the profiling and target device differs too much. There are several reasons that can cause differences in the power consumption such as aging, slight differences in the resistance and/or capacitance of a circuit due to the manufacturing process, etc. [RBA20]. However, there have been proposed a number of techniques in literature that can deal with portability problems to lower the effect of diverging profiling and attack devices (e.g. [BCH$^+$20, CK14, ZSX$^+$20]). Because of that, we believe that our results still give a realistic view of an adversary's capabilities.

We recommend a key rolling parameter between 20 and 30 after considering all aspects mentioned above and given our assumptions described in Section 4.1. On the one hand, this range gives a considerable security margin against current and future attacks, as we required almost twice as many traces to recover any information about the secret key. On the other hand, it introduces a practically overhead in terms of configuration size $(20-30\,\%)$ and boot time $(15-25\,\%)$. A key rolling parameter range of 20 to 30 represents therefore a reasonable trade-off between security and usability.

8 Conclusion

In this paper we performed a black-box side-channel analysis of the Xilinx ZU+ AES-256 engine, which is used to decrypt the configuration image (bitstream) when the device boots. While the AES core does not have any SCA countermeasures, the ZU+ is equipped with a key rolling mechanism which enforces a limit on the number of data blocks that are encrypted and decrypted under the same key. Xilinx does provide only limited information on how to choose a secure key rolling parameter. The goal of our analysis was to find suitable parameters by attacking the AES engine using state-of-the-art approaches that require a minimal amount of attack traces.

We found that a straightforward extraction of the complete 256-bit key is not possible because of the pipelining structure of the AES engine and the asymmetric authentication of the configuration data. Instead, an adversary needs to mount an attack against the first five AES rounds with only 256 consecutive encryptions. We presented several DNN-based

[2]Please note four bitstreams encrypted under the same key and with different IVs have to be available in order to recover those values

methods to exploit the found leakage model. Although none of them has been able to recover the complete AES-256 key, information about eight key bytes can be extracted by our attacks. However, we believe that future works can manage to break the full AES-256 key within 256 encryptions. A possible way might be to combine the leakage of several rounds to improve the SNR, or to apply a SAT solver that derives the remaining key bytes using partial information extracted from the trace set as proposed recently by Gohr et al. [GJS20]. In order to be protected against state-of-the-art and upcoming attacks, we suggest to set the key rolling parameter to a value between 20 and 30.

Acknowledgements

The authors would like to thank the anonymous reviewers as well as the shepherd for their valuable comments. This work was supported in parts by the German Federal Ministry of Education and Research (BMBF) under grant agreement number 16KIS0606K (SecRec).

References

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.

[BBBK11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-Parameter Optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc.

[BCH+20] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. Mind the portability: A warriors guide through realistic profiled side-channel analysis. 2020. https://www.ndss-symposium.org/wp-content/uploads/2020/02/24390-paper.pdf.

[BGH+15] Nicolas Bruneau, Sylvain Guilley, Annelie Heuser, Damien Marion, and Olivier Rioul. Less is more. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, pages 22–41, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[C+15] François Chollet et al. Keras. https://keras.io, 2015.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. *Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures*, pages 45–68. Springer International Publishing, Cham, 2017.

[CK14] Omar Choudary and Markus G. Kuhn. Template attacks on different devices. In Emmanuel Prouff, editor, *Constructive Side-Channel Analysis and Secure Design*, pages 179–198, Cham, 2014. Springer International Publishing.

[Coo12] Shane Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

[CRR03] 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.

[DM19] Lauren De Meyer. Recovering the CTR-DRBG state in 256 traces. 2020:37–65, Nov. 2019.

[DPRS11] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate Side Channel Attacks and Leakage Modeling. *Journal of Cryptographic Engineering*, 1:123–144, 2011.

[Dwo07] Morris Dworkins. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC - NIST Special Publication 800-38D. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=51288, 2007.

[EKM+08] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoq Code Hopping Scheme. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, pages 203–220, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[EMP20] Maik Ender, Amir Moradi, and Christof Paar. The Unpatchable Silicon: A Full Break of the Bitstream Encryption of Xilinx 7-Series FPGAs. In *29th USENIX Security Symposium (USENIX Security 20)*, Boston, MA, August 2020. USENIX Association.

[F-S19] Security advisory: Xilinx ZU+ Encrypt Only Secure Boot bypass. https://github.com/f-secure-foundry/advisories/blob/master/\Security_Advisory-Ref_FSC-HWSEC-VR2019-0001-Xilinx_ZU%2B-Encrypt_Only_Secure_Boot_bypass.txt, 2019.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[GJS20] Aron Gohr, Sven Jacob, and Werner Schindler. Efficient Solutions of the CHES 2018 AES Challenge Using Deep Residual Neural Networks and Knowledge Distillation on Adversarial Examples. Cryptology ePrint Archive, Report 2020/165, 2020. https://eprint.iacr.org/2020/165.

[HGG18] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Profiled Power Analysis Attacks Using Convolutional Neural Networks with Domain Knowledge. In *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, pages 479–498, 2018.

[HGG20] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Deep Neural Network Attribution Methods for Leakage Analysis and Symmetric Key Recovery. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography – SAC 2019*, pages 645–666, Cham, 2020. Springer International Publishing.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, Dec 2014.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. *Differential Power Analysis*, pages 388–397. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[KOP09] Timo Kasper, David Oswald, and Christof Paar. EM Side-Channel Attacks on Commercial Contactless Smartcards Using Low-Cost Equipment. In Heung Youl Youm and Moti Yung, editors, *Information Security Applications*, pages 79–93, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[KPH$^+$19] Jaehun 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, May 2019.

[LDMPT15] J. Longo, E. De Mulder, D. Page, and M. Tunstall. SoC It to EM: ElectroMagnetic Side-Channel Attacks on a Complex System-on-Chip. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, pages 620–640, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[MKP12] Amir Moradi, Markus Kasper, and Christof Paar. Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures. In Orr Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, pages 1–18, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MOP10] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards.* Springer Publishing Company, Incorporated, 1st edition, 2010.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. *Breaking Cryptographic Implementations Using Deep Learning Techniques*, pages 3–26. Springer International Publishing, Cham, 2016.

[MS16] Amir Moradi and Tobias Schneider. Improved Side-Channel Analysis Attacks on Xilinx Bitstream Encryption of 5, 6, and 7 Series. In François-Xavier Standaert and Elisabeth Oswald, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 71–87, Cham, 2016. Springer International Publishing.

[MVM09] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Advanced Encryption Standard.* Alpha Press, 2009.

[OC13] Colin O'Flynn and Zhizhang Chen. A Case Study of Side-Channel Analysis Using Decoupling Capacitor Power Measurement with the OpenADC. In Joaquin Garcia-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, Ali Miri, and Nadia Tawbi, editors, *Foundations and Practice of Security*, pages 341–356, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[OP11] David Oswald and Christof Paar. Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, pages 207–222, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[PHJ$^+$18] 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.

[PSB$^+$18] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cecile Dumas. Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database. Cryptology ePrint Archive, Report 2018/053, 2018. https://eprint.iacr.org/2018/053.

[RBA20] Unai Rioja, Lejla Batina, and Igor Armendariz. When similarities among devices are taken for granted: Another look at portability. In Abderrahmane Nitaj and Amr Youssef, editors, *Progress in Cryptology - AFRICACRYPT 2020*, pages 337–357, Cham, 2020. Springer International Publishing.

[RQL18] Pieter Robyns, Peter Quax, and Wim Lamotte. Improving CEMA using Correlation Optimization. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):1–24, Nov. 2018.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 30–46, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[SMY09] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 443–461, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Tim19] Benjamin Timon. Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis. 2019:107–131, Feb. 2019.

[ZSX$^+$20] F. Zhang, B. Shao, G. Xu, B. Yang, Z. Yang, Z. Qin, and K. Ren. From homogeneous to heterogeneous: Leveraging deep learning based power analysis across devices. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[ZU$_1$7*a*] AR 65528 - UltraScale Encryption - How do I estimate my bitstream size when using the Rolling Keys feature? https://www.xilinx.com/support/answers/65528.html, 2017.

[ZU$_1$7*b*] AR 67475 - Zynq UltraScale+ MPSoC Boot Times Estimation. https://www.xilinx.com/support/answers/67475.html, 2017.

[ZU$_1$9] Zynq UltraScale+ Device Technical Reference Manual UG1085 (v2.1). https://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf, 2019.

A Network Parameters

Table 4: Network configuration of CNN

Layer Type	Hyperparameters
Trace Input	-
Batch Normalization	-
Convolution 1D	filters=8, filter length=3, activation=SeLU
Batch Normalization	-
Max-Pooling	pool length=2, strides=2
Dropout	$P_{Drop} = 0.2$
Convolution 1D	filters=16, filter length=2, activation=SeLU
Batch Normalization	-
Max-Pooling	pool length=2, strides=2
Dropout	$P_{Drop} = 0.2$
Convolution 1D	filters=32, filter length=3, activation=SeLU
Batch Normalization	-
Max-Pooling	pool length=2, strides=2
Dropout	$P_{Drop} = 0.2$
Flatten	-
Dense	neurons=20, activation=SeLU
Batch Normalization	-
Dropout	$P_{Drop} = 0.2$

Table 5: Network configuration of MLP

Layer Type	Hyperparameters
Trace Input	-
Dense	neurons=10, activation=SeLU
Dropout	$P_{Drop} = 0.4$
Dense	neurons=15, activation=SeLU
Dropout	$P_{Drop} = 0.4$

Fine-tuning of the DNNs involved the number of neurons per fully-connected layer $[1, 100]$, dropout rate $[0.1, 0.9]$, choice of optimizer (*adam, rmsprop, sgd*), learning rate, activation function (*SeLU, ReLU, tanh, sigmoid*). Furthermore, we tested different batch size values (128, 256, 512, 768, 1024) for the training and got the smallest loss using a value of 256.

B Trace

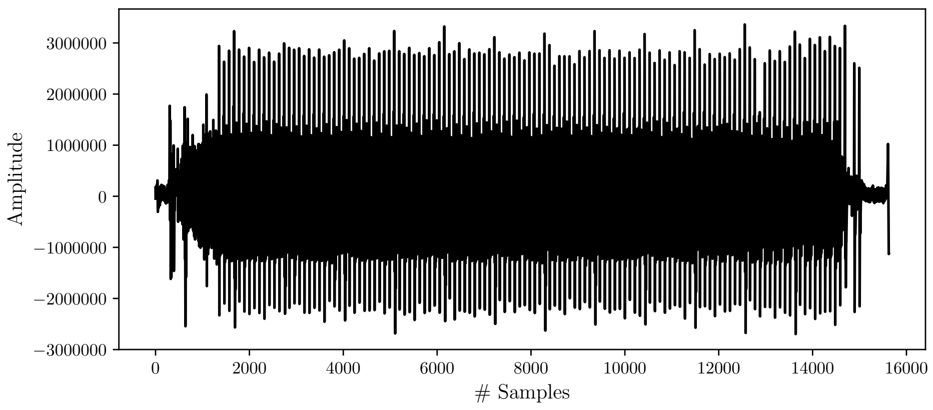


Figure 11: Plot of averaged EM trace covering 256 consecutive AES encryptions. Complete traces are used for the output-per-counter model illustrated in Figure 6b.

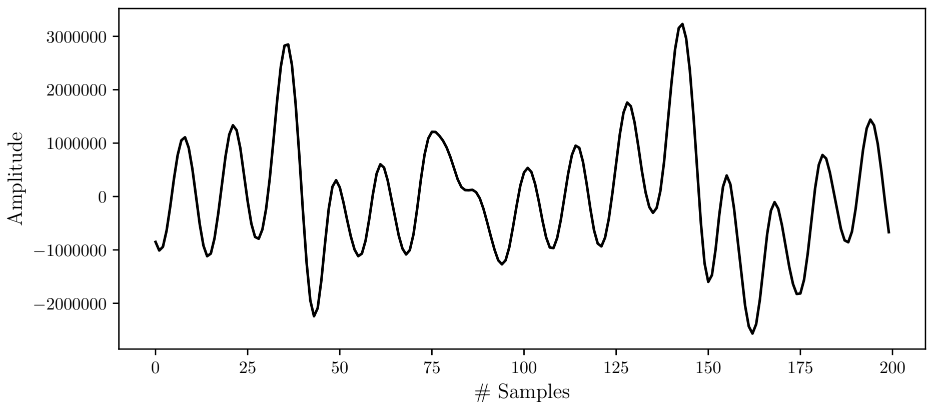


Figure 12: Segment of EM trace as used for the model-per-ctr approach described in Figure 6a.