Dynamic Logic Reconfiguration Based Side-Channel Protection of AES and Serpent

Petr Socha*, Jan Brejník*, Stanislav Jeřábek*, Martin Novotný*, Nele Mentens†
*Czech Technical University in Prague, Faculty of Information Technology, Czech Republic
{sochapet,brejnjan,jerabst1,novotnym}@fit.cvut.cz
†KU Leuven, ES&S and imec-COSIC/ESAT, Belgium
nele.mentens@kuleuven.be

***Abstract*—Dynamic logic reconfiguration is a concept which allows for efficient on-the-fly modifications of combinational circuit behaviour in both ASIC and FPGA devices. The reconfiguration of Boolean functions is achieved by modification of their generators (e.g. shift register-based look-up tables) and it can be controlled from within the chip, without the necessity of any external intervention. This hardware polymorphism can be utilized for the implementation of side-channel attack countermeasures, as demonstrated by Sasdrich et al. for the lightweight cipher PRESENT.**

In this work we adopt these countermeasures to two of the AES finalists, namely Rijndael and Serpent. Just like PRESENT, both Rijndael and Serpent are block ciphers based on a substitution-permutation network. We describe the countermeasures and adjustments necessary to protect these ciphers using the resources available in modern Xilinx FPGAs. We describe our VHDL implementations and evaluate the side-channel leakage and effectiveness of different countermeasure combinations using a methodology based on Welch's t-test.

We did not detect any significant leakage from the fully protected versions of our implementations. We show that the countermeasures proposed by Sasdrich et al. are, with some modifications compared to the protected PRESENT implementation, successfully applicable to AES and Serpent.

***Index Terms*—Internet of Things, Embedded Security, Cryptography, Side-Channel Analysis, Dynamic Reconfiguration**

I. Introduction

The use of computers and various embedded systems has become our daily routine in the past years. In the upcoming Internet-of-Things (IoT) era, smart cities and smart homes are expected to bring even more embedded devices into our everyday lives. The presence of such smart devices, including personal assistants, cars and many more, makes our private lives more vulnerable than ever before. In order to protect sensitive information, various authentication, authorization, and encryption schemes need to be employed. Even though these algorithms may be considered secure, their implementations may still be vulnerable to side-channel attacks. These attacks exploit the fact that sensitive information may leak through side channels, such as the power consumption of the device [1], [2] or its electromagnetic radiation [3]. Given the typical deployment of IoT devices, where the attacker may easily gain physical access and tamper with the device, these attacks pose a serious threat.

To prevent side-channel attacks, many different countermeasures have been proposed. Masking is a popular approach based on randomizing intermediate cipher values by introducing a random mask [4], [5], making it difficult for an attacker to predict the processed values. Another approach, called hiding, tries to hide the information leakage, e.g. through the use of dual-rail logic [6]. Dynamic reconfiguration has been proposed as another hiding countermeasure to achieve side-channel resistance [7]. A combination of countermeasures implemented using dynamic logic reconfiguration is proposed in [8] and evaluated on the lightweight block cipher PRESENT [9].

In this paper, we extend the work presented in [8] by using dynamic logic reconfiguration to secure two of the Advanced Encryption Standard (AES) competition finalists, Rijndael [10] (winner of the competition, nowadays therefore known as the AES) and Serpent [11]. We describe our implementations and the non-straightforward way in which we tailored the countermeasures in [8] to AES and Serpent. We evaluate the side-channel leakage and the effectiveness of different countermeasure combinations.

II. Theoretical Background

In this work, we intend to secure AES and Serpent using the approach described in [8]. In the following subsections, we first describe both AES/Rijndael and Serpent. Then we explain the concept of dynamic logic reconfiguration on FPGA, and finally we describe the implemented and evaluated countermeasures.

A. AES Finalists: Rijndael and Serpent

Both ciphers share common features [12]. They are iterated substitution-permutation networks (SPN) with a block size of 128 bits and possible key sizes of 128, 192 or 256 bits. The plaintext (i.e. the data to be encrypted) is transformed into a ciphertext by iteratively applying a number of operations. Each iteration is called a round. Both ciphers also describe a method for expanding the secret key into a number of subkeys which are used as an input to each round.

1) AES/Rijndael: Rijndael [10] consists of 10, 12 or 14 rounds (depending on the key length). First, the secret key is XORed with the plaintext. After that, a number of round transformations is performed. Each round consists of four layers: a non-linear substitution layer (SubBytes, i.e. 16 parallel applications of an 8-bit substitution box or S-box), two

linear mixing layers (ShiftRows and MixColumns) and a XOR with the round subkey (AddRoundKey). In the last round, the MixColumns transformation is omitted.

2) Serpent: Serpent [11] consists of 32 rounds. First, an initial permutation is applied and then the round transformations take place. Each round consists of three layers: a XOR with the round subkey, a non-linear substitution layer (i.e. 32 parallel applications of one of the eight specified 4-bit S-boxes, which are different in the consecutive rounds), and a linear transformation. In the last round, a second XOR takes place instead of the linear transformation. In the end, the final permutation is applied.

B. Dynamic Logic Reconfiguration

In FPGAs, combinational circuits are typically implemented using Look-Up Tables (LUTs), i.e. configurable primitives which store truth tables of k-input Boolean functions $f : \mathbb{B}^k \rightarrow \mathbb{B}$. Dynamic logic reconfiguration allows for the run-time alteration of the circuit behaviour by modifying the content of specific look-up tables, while leaving the routing intact. The reconfiguration of LUTs is done from within the chip itself and can be achieved e.g. by using a shift register (allowing for serial programming) and a cascade of addressing multiplexers. In Xilinx FPGAs [13], this functionality is provided by k-input Configurable Look-Up Tables (CFGLUTs) with a serial configuration input and output (allowing to connect CFGLUTs in separately configurable chains). In Xilinx Spartan-6 FPGAs, 5-input CFGLUTs are available.

In order to implement dynamically reconfigurable Boolean functions $f : \mathbb{B}^n \rightarrow \mathbb{B}$, where $n > k$, multiple k-input CFGLUTs are required in combination with addressing multiplexers (using Boole's expansion, also referred to as the Shannon expansion [14]). Specifically, to implement an n-input function using k-input CFGLUTs and 2-to-1 multiplexers, we need 2^{n-k} CFGLUTs and $2^{n-k} - 1$ multiplexers.

Multiple-output Boolean functions $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ can be trivially implemented as m single-output Boolean functions $f_i : \mathbb{B}^n \rightarrow \mathbb{B}$.

C. Countermeasures

To protect AES and Serpent, we have implemented countermeasures that were proposed (and evaluated on PRESENT) by Sasdrich et al. in [8]. In this subsection, we briefly describe these countermeasures.

1) S-box Decomposition: Since information leakage often occurs based on changing values in registers, and since the output of the non-linear substitution layer is a frequent target of side-channel attacks, the S-box decomposition countermeasure is based on avoiding the storage of the S-box outputs into such registers. This is done by decomposing the S-box into two bijections R_1, R_2, where

$$\text{S-box}(x) = R_2(R_1(x)), \tag{1}$$

and placing the register in between the two bijections. The number of possible n-bit bijections for R_1 is equal to $(2^n)!$. For each option, a bijection R_2 can be found such that Eq. (1) holds.

Thanks to dynamic logic reconfiguration, different bijections R_1, R_2 can easily be used for every encryption. Starting with R_1 being an identity and R_2 being the actual S-box, the bijections for the next encryption are computed by randomly selecting two pairs of elements in the R_1 mapping, swapping them, and recomputing R_2 accordingly.

2) Boolean Masking: In order to randomize intermediate values, a random mask is added (XORed) to the data prior to encryption, and subtracted (i.e. once again XORed) after the encryption. For the cipher to produce valid results working with masked data, various alterations must be done.

Boolean masking can be combined with the previously mentioned bijective S-box decomposition and can once again take advantage of dynamic logic reconfiguration. Two different random masks m_1, m_2 are used for every encryption: mask m_1 is used outside the decomposed S-box, and mask m_2 is used inside of it. If the substitution layer would be the only layer in the round, the previously mentioned bijections R_1, R_2 would get adjusted as follows:

$$R'_1(x) = R_1(x \oplus m_1) \oplus m_2, \tag{2}$$

$$R'_2(x) = R_2(x \oplus m_2) \oplus m_1. \tag{3}$$

The function R'_1 first subtracts/removes mask m_1, then performs the R_1 bijection mapping, and finally masks this value using m_2. The output of this function is stored in the register. Analogically, the function R'_2 subtracts the mask m_2, does the R_2 mapping and masks the result using m_1. This way, the same CFGLUTs can be used for both the S-box decomposition and the masking, saving both area and reconfiguration time.

However, to deal with the linear transformation layers, further alterations to the R'_1, R'_2 bijections need to be done. We can exploit one of these two facts:

$$f(x) = f(x \oplus f^{-1}(m)) \oplus m, \tag{4}$$

$$f(x) = f(x \oplus m) \oplus f(m), \tag{5}$$

which both hold when $f(x)$ is a linear mapping. These give us two different and fairly straightforward approaches to take linear transformations $f(\cdot)$ into account.

One option is to alter R'_2 function in terms of Eq. (4) so that m_1 processed by the inverse transformation is used to mask the data, allowing to subtract m_1 in R'_1:

$$R'_1(x) = R_1(x \oplus m_1) \oplus m_2, \tag{6}$$

$$R'_2(x) = R_2(x \oplus m_2) \oplus f^{-1}(m_1). \tag{7}$$

The second option is to use m_1 for masking in R'_2, and to alter R'_1 according to Eq. (5), so that m_1 processed by the linear transformation gets subtracted:

$$R'_1(x) = R_1(x \oplus f(m_1)) \oplus m_2, \tag{8}$$

$$R'_2(x) = R_2(x \oplus m_2) \oplus m_1. \tag{9}$$

Notice that further alterations may be required for the first and the last round, depending on the selected approach.

The last obstacle is the subkey XOR layer, which can be considered an affine transformation. Suppose we have a vector x, which gets XORed with the subkey: $x \oplus k$. Suppose we process masked data the same way: $(x \oplus m) \oplus k$, then by subtracting the mask m with no alterations we have:

$$((x \oplus m) \oplus k) \oplus m) = x \oplus k. \tag{10}$$

Therefore, no further alterations need to be done to take the XOR layer into account.

3) Register Precharge: Because the same masks are used for the whole encryption (i.e. for every round), the leakage occurs in the register, since

$$\mathrm{HD}(x \oplus m, y \oplus m) = \mathrm{HD}(x, y), \tag{11}$$

where $\mathrm{HD}(x, y)$ denotes the Hamming distance between x and y. To avoid this leakage, the register is duplicated and the processed data are interleaved with random data. This technique avoids leakage, however, it reduces the throughput of the circuit when it is implemented using an architecture that is not fully unrolled.

III. Secure Cipher Design

In this section, we examine the specifics of both AES/Rijndael and Serpent and we propose a manner in which these ciphers can be secured against side-channel attacks using the countermeasures explained in Section II-C.

In order for our implementations to fit into a Xilinx Spartan-6 FPGA device, we take into account that CFGLUTs with at most 5 input bits are available. When a platform with smaller CFGLUTs is available, the dynamic logic reconfiguration method can be implemented using the approach described in Section II-B.

A. AES/Rijndael

Rijndael employs an 8×8 S-box, which can be considered as a function $\text{S-box}_{\text{Rijndael}} : \mathbb{B}^8 \rightarrow \mathbb{B}^8$. Therefore, to implement the Rijndael S-box using reconfigurable logic, $8 \cdot 2^{8-5} = 64$ (5-input) CFGLUTs and $8 \cdot (2^{8-5} - 1) = 56$ (2-to-1) multiplexers are necessary. Moreover, the S-box decomposition countermeasure suggests the S-box to be split into two bijections $R_1, R_2 : \mathbb{B}^8 \rightarrow \mathbb{B}^8$, which doubles the amount of CFGLUTs and multiplexers in the secured version. Since the Rijndael algorithm applies 16 S-boxes in parallel, this brings the total count up to 2048 (5-input) CFGLUTs and 1792 (2-to-1) multiplexers.

The decomposition into two bijections is done in a similar fashion as described in Section II-C, with the round register being placed in between the two bijections. For the AES algorithm, we have decided to swap 8 pairs of elements in the R_1 bijection after every encryption (in contrast to the PRESENT 4-bit S-box decomposition in [8], where only two pairs get swapped).

To implement the Boolean masking countermeasure as described in Section II-C, bijections R_1', R_2' (i.e. the decomposed S-box combined with masking) must be altered. We choose the option where R_2' adds the mask m_1 and R_1' subtracts m_1 processed by the linear transformations (see Eq. (8)):

$$R_1'(x) = R_1(x \oplus \mathrm{MixColumns}(\mathrm{ShiftRows}(m_1))) \oplus m_2, \tag{12}$$

$$R_2'(x) = R_2(x \oplus m_2) \oplus m_1. \tag{13}$$

Note that the data are masked by m_1 in the second bijection R_2 and that this mask is subtracted in the following round. Therefore prior to the first round, the input data must be masked properly. Also, the last round of Rijndael omits the $\mathrm{MixColumns}$ operation. Therefore, in the last round, only $\mathrm{ShiftRows}(m_1)$ must be subtracted in R_1', or additional unmasking of the output must be done (which is our choice).

The implementation of the register precharge requires the register to be duplicated and the controller to be adjusted appropriately, such that the processed data are interleaved with random data.

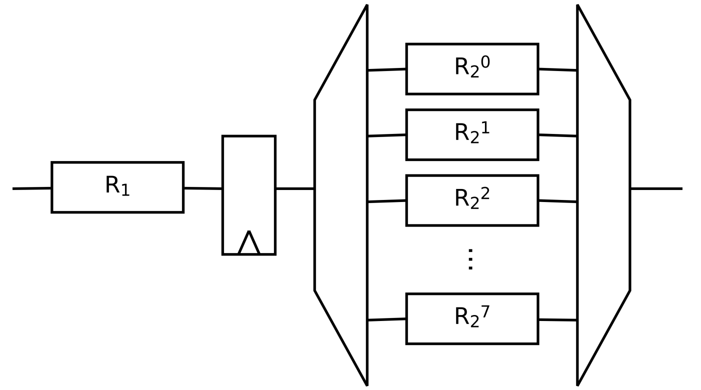


Figure 1: Serpent S-box Decomposition

B. Serpent

Unlike Rijndael or PRESENT, Serpent defines eight different 4×4 S-boxes. Each S-box is used in a different round. One way to implement the S-box decomposition is to decompose each of these S-boxes into two bijections, resulting in 16 bijections in total. We have decided for an approach where the first bijection R_1 is shared among all S-boxes, while the other 8 bijections R_2^i, $i \in \{0, .., 7\}$, implement the eight S-boxes, with the correct output being selected by a multiplexer. The eight decomposed Serpent S-boxes are depicted in Figure 1. Notice the **demultiplexer**, which selects the right R_2^i bijection, while the other bijections are fed with zeroes. This demultiplexer is necessary to prevent glitches that lead to information leakage. Since the Serpent S-boxes realize the functions $\text{S-box}^{\text{i}}_{\text{Serpent}} : \mathbb{B}^4 \rightarrow \mathbb{B}^4$, only four CFGLUTs are necessary to implement the bijection. Given the selected architecture, $4 + 8 \cdot 4 = 36$ CFGLUTs are required to decompose all eight S-boxes. Since the S-box is applied 32 times in parallel, this results in 1152 CFGLUTs in total.

Boolean masking is implemented similarly to the Rijndael algorithm, with m_1, processed by the linear transformation,

being subtracted in the R_1' bijection (see Eq. (8)). Suppose the Serpent linear transformation is $L_{Serpent}$, then:

$$R_1'(x) = R_1(x \oplus \mathrm{L}_{\mathrm{Serpent}}(m_1)) \oplus m_2, \tag{14}$$

$$R_2'(x) = R_2(x \oplus m_2) \oplus m_1. \tag{15}$$

Regarding the first round, similarly to the Rijndael approach, appropriate initial masking of the input data must be performed first. Also, there is no linear transformation in the last round, therefore, either the unprocessed mask m_1 gets subtracted in R_1', or final unmasking must be performed.

Register precharge is once again implemented simply by duplicating the round register and altering the controller appropriately to interleave the processed data with random data.

C. Reconfiguration Controller

For every encryption, new bijections are generated (as described in Section II-C), as well as new masks m_1, m_2. This requires the CFGLUTs configurations to be computed and loaded prior to every encryption. The reconfiguration of all CFGLUTs can be done using different levels of parallelism (the CFGLUTs "programming" I/O can be variously chained, given its shift register nature).

IV. Side-Channel Leakage Evaluation

In this section, we present our experimental set-up and a leakage methodology used to evaluate all combinations of previously described countermeasures.

A. Set-up and Methodology

We choose the Sakura-G board [15] with a Xilinx Spartan-6 FPGA as our evaluation platform. AES/Rijndael and Serpent VHDL implementations with a 128-bit key are evaluated. The power traces are measured using a PicoScope 6406D oscilloscope.

Leakage is evaluated using the non-specific univariate first-order Welch's t-test as described in [16]. This evaluation method consists of two phases. In the active phase, power traces are collected, each trace measured while encrypting either a random or a (preselected) constant plaintext. In the analytical phase of the evaluation, Welch's t-test statistic is computed in each sampling point, examining the null hypothesis of equal population means (where one population consists of random plaintext measurements and the other population consists of constant plaintext measurements). For every evaluation, 1 million power traces are captured.

The necessary random data (random pairs to be swapped in the bijection, random masks, register precharge with random values) are generated externally and sent to the cryptographic device alongside the plaintext. This approach allows us to easily enable or disable specific countermeasures.

B. Results

We evaluate every possible combination of the proposed countermeasures:

(a) Unprotected
(b) Register Precharge
(c) Masking
(d) Masking + Register Precharge
(e) S-box Decomposition
(f) S-box Decomposition + Register Precharge
(g) S-box Decomposition + Masking
(h) S-box Decomposition + Masking + Register Precharge

For every implementation, 1 million power traces are measured and processed using a non-specific first-order t-test, as described earlier. Figure 2 depicts the t-values during the AES encryption and Figure 3 depicts the t-values during the Serpent encryption. The sensitive information leakage is the most prominent for the unprotected versions, as expected.

It is also visible that different countermeasures and their combinations have various influence on the significance of the detected leakage. Figures 2c and 3c show that a countermeasure based on masking only protects the first round of the cipher, while, starting from the second round, the leakage is comparable to the unprotected version (cf. Figures 2a and 3a). Figures 2d and 3d suggest that masking becomes more effective in combination with register precharge (which is expected, as discussed in Section II-C).

Figures 2h and 3h show results with all three countermeasures combined. As can be seen, no significant first-order leakage is detected when evaluating these fully protected implementations. However, the used TVLA methodology is merely a first step in the evaluation of a side-channel security of the implementations and the results do not provide any guarantee of a security level [17]. This is not only because of a high risk of both false positives and false negatives, but also because only univariate statistics is considered in our methodology.

V. Conclusion

In this paper, we describe and evaluate side-channel attack protected AES and Serpent implementations, which are based on an approach demonstrated by Sasdrich et al. [8] for the PRESENT cipher. These implementations utilize dynamic logic reconfiguration, which can easily be deployed in both FPGA and ASIC designs. We describe a method by means of which a generic substitution-permutation network can be protected against side-channel attacks, and we tailor the approach to a Xilinx Spartan-6 FPGA for the protection of both AES and Serpent.

We demonstrate the effectiveness of the implemented countermeasures by evaluating the side-channel leakage using Welch's t-test, with different combinations of countermeasures in place. We did not detect any significant leakage from the protected versions of both AES and Serpent encryption implementations.

Acknowledgement

This work was partially funded by the CELSA project "DRASTIC: Dynamically Reconfigurable Architectures for Side-channel analysis protection of Cryptographic implementations" (CELSA/17/033); and CTU grant No. SGS17/213/OHK3/3T/18.

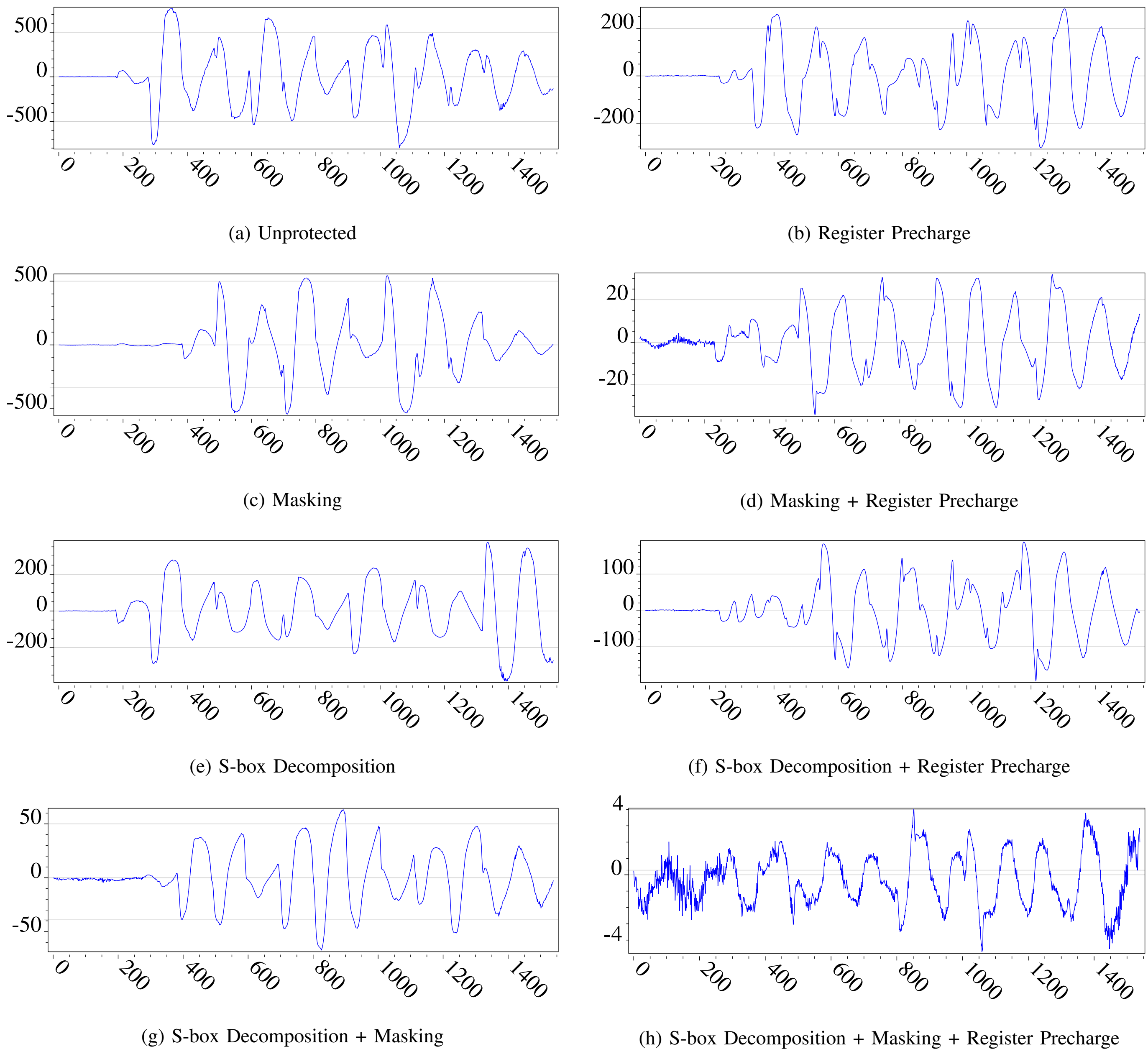


Figure 2: Results of the AES/Rijndael t-test, where the t-value is shown on the vertical axis and the time samples during encryption are shown on the horizontal axis

References

[1] P. Kocher, J. Jaffe, and B. Jun, *Differential Power Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[2] E. Brier, C. Clavier, and F. Olivier, “Correlation power analysis with a leakage model,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 16–29.

[3] J.-J. Quisquater and D. Samyde, “Electromagnetic analysis (ema): Measures and counter-measures for smart cards,” in *Smart Card Programming and Security*. Springer, 2001, pp. 200–210.

[4] G. Piret and F.-X. Standaert, “Security analysis of higher-order boolean masking schemes for block ciphers (with conditions of perfect masking),” *IET Information Security*, vol. 2, no. 1, pp. 1–11, 2008.

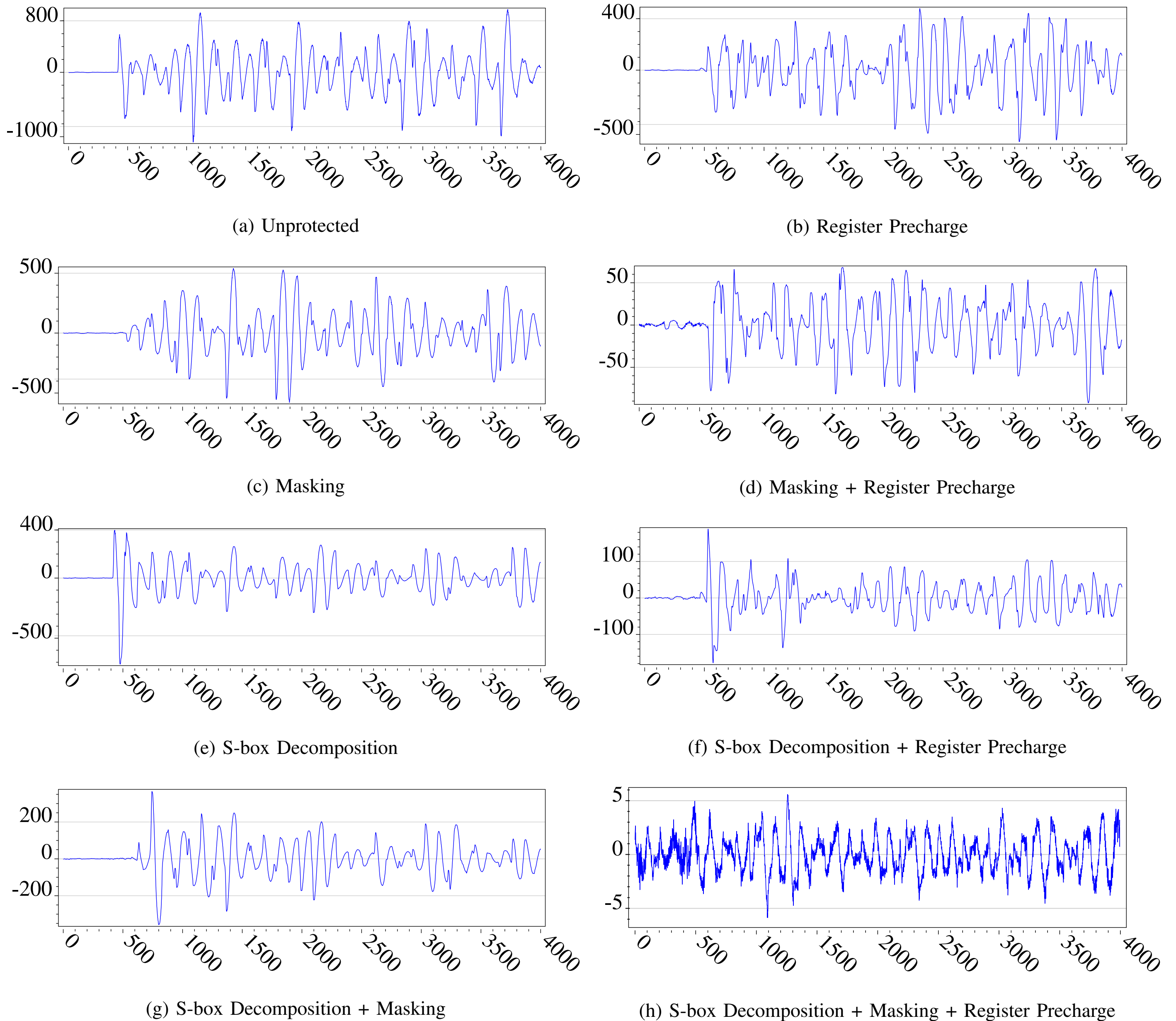
[5] S. Nikova, C. Rechberger, and V. Rijmen, “Threshold implementations against side-channel attacks and glitches,” in *International conference on information and communications security*. Springer, 2006, pp. 529–545.

[6] D. Sokolov, J. Murphy, A. Bystrov, and A. Yakovlev, “Design and analysis of dual-rail circuits for security applications,” *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 449–460, 2005.

[7] N. Mentens, B. Gierlichs, and I. Verbauwhede, “Power and fault analysis resistance in hardware through dynamic reconfiguration,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2008, pp. 346–362.

[8] P. Sasdrich, A. Moradi, O. Mischke, and T. Güneysu, “Achieving side-channel protection with dynamic logic reconfiguration on modern fpgas,” in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2015, pp. 130–136.

[9] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, “Present: An ultra-lightweight



(a) Unprotected

(b) Register Precharge

(c) Masking

(d) Masking + Register Precharge

(e) S-box Decomposition

(f) S-box Decomposition + Register Precharge

(g) S-box Decomposition + Masking

(h) S-box Decomposition + Masking + Register Precharge

Figure 3: Results of the Serpent t-test, where the t-value is shown on the vertical axis and the time samples during encryption are shown on the horizontal axis

block cipher," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2007, pp. 450–466.

[10] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.

[11] E. Biham, R. Anderson, and L. Knudsen, "Serpent: A new block cipher proposal," in *International Workshop on Fast Software Encryption*. Springer, 1998, pp. 222–238.

[12] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback, "Report on the development of the advanced encryption standard (aes)," *Journal of Research of the National Institute of Standards and Technology*, vol. 106, no. 3, p. 511, 2001.

[13] Xilinx, "Spartan-6 libraries guide for hdl designs." [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/spartan6_hdl.pdf

[14] F. M. Brown, *Boolean reasoning: the logic of Boolean equations*. Springer Science & Business Media, 2012.

[15] H. Guntur, J. Ishii, and A. Satoh, "Side-channel attack user reference architecture board sakura-g," in *Consumer Electronics (GCCE), 2014 IEEE 3rd Global Conference on*. IEEE, 2014, pp. 271–274.

[16] T. Schneider and A. Moradi, "Leakage assessment methodology," *Journal of Cryptographic Engineering*, vol. 6, no. 2, pp. 85–99, 2016.

[17] F.-X. Standaert, "How (not) to use welch's t-test in side-channel security evaluations," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2018, pp. 65–79.