Building a Side-Channel Attack Scheme on SipHash FPGA Implementation

Vít Mašek, Vojtěch Miškovský, Matúš Olekšák
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
{masekvit,miskovoj,oleksmat}@fit.cvut.cz

***Abstract*—This paper introduces a novel side-channel attack scheme targeting FPGA implementations of the SipHash cipher, a cryptographic hash function commonly used for message authentication. At the time of writing, we were unaware of any side-channel attack on hardware implementation of the SipHash. A leakage function was built that is able to compute part of the internal state after 1 round based on just a few bits of the key. Our approach is based on progressively eliminating incorrect subkeys in iterations, while adding new bits of the key we attack, rather than trying to extract the subkey directly. The total amount of key hypotheses does not exceed a limit when it becomes unfeasible to compute. In the end, all 128 bits of the SipHash key are retrieved solely from the Hamming distance of the initial state and the state after first round.**

***Index Terms*—ARX-based cryptography, SipHash, Side-Channel Attacks, FPGA**

I. Introduction

Side-channel attacks (SCAs) are a class of intrusive techniques that exploit the physical implementations of cryptographic systems, rather than flaws in the algorithms themselves. These attacks leverage information collected from the physical environment of the cryptographic device, including timing information [1], power consumption [2], or electromagnetic leaks [3]. Initially recognized in the late 1990s, SCAs have since emerged as a significant threat to the security of cryptographic modules.

The SipHash cipher, introduced by Aumasson and Bernstein in 2012 [4] is an ARX-based cryptographic hash function optimized for speed and aimed at safeguarding against hash-flooding DoS attacks. It is widely appreciated for its simplicity and efficiency in generating 64-bit Message Authentication Codes (MACs) from a variable-length message and a 128-bit secret key.

II. Background and Related Work

A. SipHash

SipHash is a pseudo-random function used for 64-bit MAC generation with 128-bit key. Its core is an ARX function SipRound, see Figure 1. Its 256-bit state is divided into four 64-bit words – $v_{0..3}$. First, the key is XORed to the initial state. The message is then embedded into the state by 64-bit blocks. After a message block is XORed to the state, the state is updated by the SipRound function, as shown in Figure 2. When the full message is processed, final rounds of SipRound are performed and the 4 state variables are XORed together making the final 64-bit result.

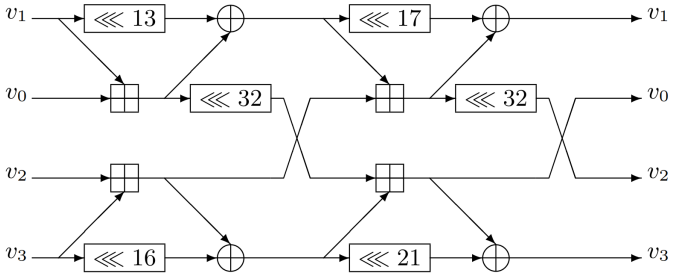

Fig. 1. SipRound Scheme

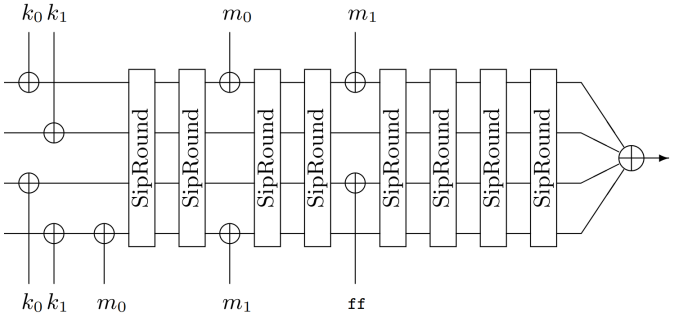

Fig. 2. SipHash-2,4 Scheme

B. Side-Channel Attacks Targeting ARX-Based Functions

The results of [5] indicate "potential intrinsic resilience" of ARX-based functions against side channel attacks. However, authors of [6] found a leakage in each core operation (addition, rotation, and xor) and calls for a re-evaluation of security assumptions.

While successful side-channel attack on software implementation of SipHash was presented in [7], FPGA implementations seem to be still untouched. The simplicity of SipRound function allows to implement it in only one clock cycle, making it basically impossible to target any intermediate value within the SipRound function itself, as the attack presented in [7] does.

Due to the big difference between CPU and FPGA implementation of SipHash, the leakage function presented in [7]

cannot be directly reused for FPGA implementation. This paper aims to address this obstacle, and introduces a novel approach to target the result of SipRound, rather then its intermediate values, making it suitable for attacking FPGA implementations.

III. Building the Attack

This section describes the process which leads to successfully building a side-channel attack scheme on SipHash, that can be used for FPGA implementations. The attack was build, debugged and tested using model implementation of SipHash in Python for simplicity. We observe the intermediate values and model the power consumption. With this approach, we are able to better understand and debug issues that came along the process.

We tested the model against our real FPGA implementation of SipHash using ChipWhisperer CW308 UFO Board with Spartan-6 target. It was experimentally verified that the real and modeled power consumption has high corelation, giving us a confidence in using this approach, rather than straight trying to attack a real device.

A. Notation

- K – SipHash key
- P – Input to the very first SipRound
- S – Output of the very first SipRound
- S_i – Value of i-th bit of S
- I – Sorted list of bit indexes of K
- I_i – Sorted list of all bits of K, that has direct influence on the value of S_i
- K_I – Set of subkeys using bits of from I
- k_I – Subkey from K_I

B. Analyzing the SipRound

For the initial attack, we need to find a bit i of S (S_i), which value is determined by only small number of bits of the secret key K. If we find such S_i, we can mount a classic DPA using this bit and retrieve the corresponding bits of K. We run analysis to get the list I_i of all key bits needed to compute the value of S_i. Figure 3 shows the size of I_i for all 256 bits i of the state.

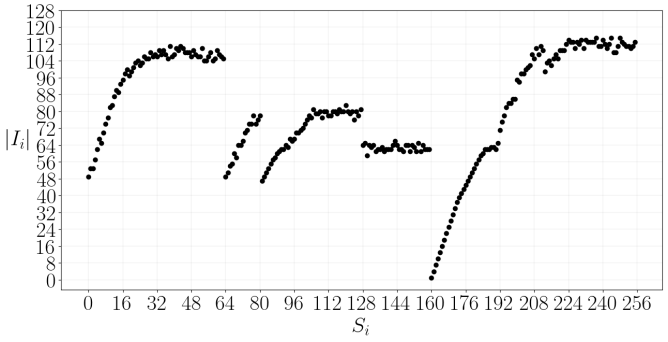

Fig. 3. Number of key bits influencing the value of bit S_i

For i starting from 160, only few key bits are needed to compute the given bit of the internal state after the first SipRound. These bits are a good candidates to mount a DPA attack.

C. Leakage Function and its Success Rates

We define a leakage function

$$L(k_{I_i}, m_0, i) = SipRound(IV \oplus k_{I_i} \oplus m_0)_i$$

where k_{I_i} is a key hypothesis on I_i key bits, and also a measurement function

$$O(K, m_0, i) = SipRound(IV \oplus K \oplus m_0)_i$$

where K is the correct key. Then we define a success rate function for given key hypothesis as

$$R(k_{I_i}) = \frac{|M_{eq}|}{|M|}$$

where M is a set of random messages, and

$$M_{eq} = \{m_0 \in M | O(K, m_0, i) == L(k_{I_i}, m_0, i)\}$$

For the correct key hypothesis, the mentioned equality will hold for all $m_0 \in M$, and $R = 1$. For incorrect hypothesis, R should converge to 0.5.

We try bit $i = 161$, where $|I_{161}| = 4$, so 2^4 key hypotheses. We compute the R function for each hypothesis, using $|M| = 40.000$, and get results shown in Figure 4.

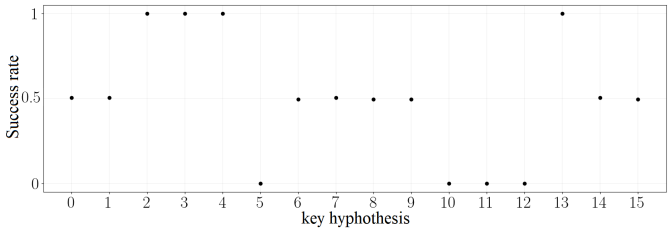

Fig. 4. Success rates of each $k_{I_{161}}$ using bit S_{161}

Using the leakage function and the success rate model, we are able to separate the key hypotheses into three groups:

- $R = 1$
- $R \approx 0.5$
- $R = 0$

Mounting DPA targeting the value of S_{161} would eliminate the keys for which $R \notin \{0, 1\}$, which is exactly half. We note these keys as $\widehat{K}_{I_i}$. If done for $i = 162$, the elimination is 75%, and for $i = 163$ it is 87.5 %.

The fact that multiple key hypotheses also pass this test and have $R = 1$ is not surprising. Actually, it holds with the findings of [8]. Generalized ARX-Box has the property, that for each key k, there is a key $k' \neq k$, always leading to the same result. Furthermore, we found out that this property may be extended to additional keys k' which always lead to opposite result, when considering only single bit. This is the group where $R = 0$.

D. Iterative Attack

Using single bit of the SipHash state, we are able to eliminate some keys, however, the size of I_i for bits from $i = 161$ grows quickly, and it becomes unfeasible. The idea is to use the knowledge of $\widehat{K}_{I_i}$ got by targeting bit i to reduce the number of key hypotheses K_{I_j} for different state bit j upfront.

So we first target the state bit 161, and get $\widehat{K}_{I_{161}}$. Then, we move to bit 162, and extend the set $\widehat{K}_{I_{161}}$ to $K_{I_{162}}$, by only enumerating bits $b \in I_{162} \setminus I_{161}$. Because $|I_{162} \setminus I_{161}| = 3$, for each subkey from $\widehat{K}_{I_{161}}$ we add 2^3 keys to the new set $K_{I_{162}}$. And because $|\widehat{K}_{I_{161}}| = 2^3$, the size of the final set $K_{I_{162}}$ of key hypotheses is $2^3 \times 2^3 = 2^6$.

Instead of targeting the state bit 162 directly, we used the knowledge got by targeting state bit 161, and all together we need to try

$$2^{|I_{161}|} + 2^{|I_{162} \setminus I_{161}|} = 2^4 + 2^6 < 2^{|I_{162}|} = 2^7$$

With each iteration, moving from bit i, we find an unused bit j such that $|I_j \setminus I_i|$ is the smallest, and then the key bits used in the next iteration is $I = I_j \cup I_i$. We iterate until $|I| = 128$ (full key) and $|\widehat{K}_I|$ is in brute-force range.

Figure 5 plots the size of K_I with respect to the size of I throughout the iterations. The number of key hypotheses reaches a maximum of $2^{27.3}$ in iteration 36. From then, is starts shrinking again. In iteration 66 the size of I hits the full key size. After 88 iterations and total of 2^{30} tried key hypotheses, we are left with 16 last possible keys.

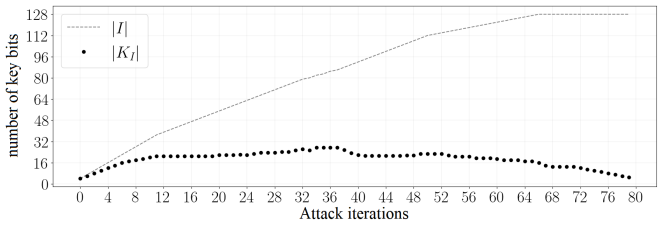

Fig. 5. Iterative Attack Progress – Success Rate Model

E. Full State Model

So far, we used the simplified success rate model, where we were comparing the value of our leakage function L and measurement function O. We ignored the rest of the SipHash state, considering it as a random noise.

Now, we start to model the power consumption as a Hamming distance between

$$P = (IV \oplus K), \quad \text{and} \quad S = SipRound(P \oplus m_0)$$

From our experiments on real FPGA implementation, using ChipWhisperer platform and Spartan6 FPGA target, this model of power consumption reaches a correlation of 0.6 with the real measurements.

Even if our leakage function computes only the value of S_i, because the value of P_i is constant, we can simply consider it as 0. If the real value of P_i is 1, in the success rate model, this would just result in $R = 0$ for the correct key. For the key hypothesis that would have $R \in \{0, 1\}$ in the success rate model, the DPA partitioning and the absolute difference of means (DoM) will be equal. This is why we can avoid computing the actual value of P_i.

Bad Bits: We quickly find, that the attack fails after a few iterations. This is caused by so-called "bad bits". The Hamming distances on some pairs of state bits is perfectly correlated. Sometimes positively, sometimes negatively. This results in following difference of means in the Hamming distance on the full state, when we partition the modeled traces using given bit of state for correct key, see Figure 6

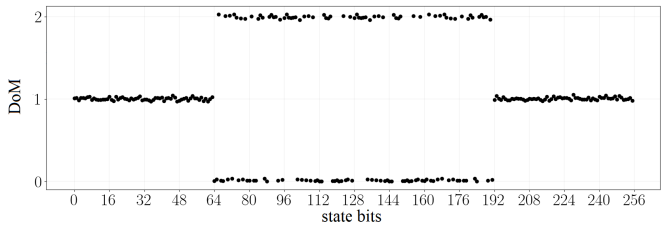

Fig. 6. Correct Absolute Difference of Means per bit i

In general, we would expect the DoM to converge to 1 for all bits. But if a pair of bits (i, j) is negatively correlated:

$$\forall m_0 \in M : (P_i \oplus S_i) \neq (P_j \oplus S_j)$$

they mask each other, and the DoM converge to 0. When a pair (i, j) of state bits is positively correlated in the Hamming distance, their combined Hamming distance is aether 0, or 2. Whereas if they are negatively correlated, their combined Hamming distance is always 1, therefore we cannot distinguish between the cases when HD on bit i is 1, or 0. The issue is, that how these bits are correlated (positively or negatively) depends on the value of the correct key. That is, obviously, unknown. So we have to be able to detect these negatively correlated state bits during the attack.

Understanding the cause of this behavior will require a comprehensive analysis of the SipRound function and structure, which is beyond the scope of this paper. However, because it is key dependent, it is another possible leakage of information about the secret key.

Fixing the Full State Model: We solve this by performing a Welch's t-test [9] on a randomly selected key from $\widehat{K}_I$ after each iteration. We put null hypothesis H_0, that the means of the two partitions made in DPA attack have equal means. If we do not reject the null hypothesis, the DoM is not statistically significant, and we scrape the results of given iteration and continue with next bit without any update. Figure 7 shows the t-statistic for the correct key. The structure is perfectly aligned with the DoM structure. The DoM could be entirely replaced by the Welch's t-test, however, it is more computationally extensive than DoM, so we do it only once at the end of each iteration.

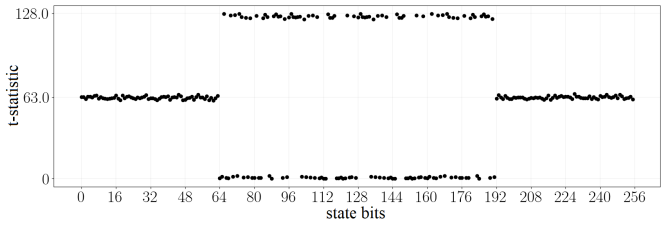

Fig. 7. Correct t-statistic per bit i

Figure 8 plots the progress of the attack algorithm. The maximum size of K_I reaches $2^{28.7}$ in iteration 31 and does not go below 2^{26} until iteration 50. The total number of key hypotheses that the attack must go through is $2^{31.7}$. The size of M (number of modeled traces) used was 8.000, and the critical value of the t-test was set to 5.

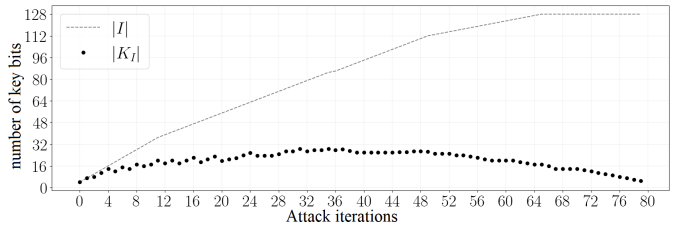

Fig. 8. Iterative Attack Progress – Full State Model

Like this, we were able to successfully retrieve the correct SipHash key solely from the Hamming distance between the initial state after key initialization P and the result S of the very first SipRound. We used a single-bit DPA attack performed in iterations, gradually increasing the number of key bits used for the attack, while keeping the total number of key hypothesizes and modeled traces in a feasible range.

IV. Preliminary Results on a Real Device

Before replacing the modeled power consumption with the real measurements, we first look at the structure of the t-statistic for each state bit, the same way as in Figures 7. Single power trace of the computation is shown in Figure 9.

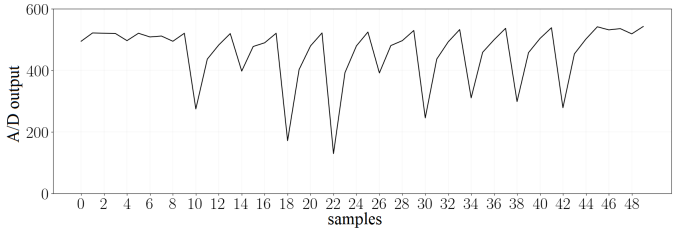

Fig. 9. Single Power Trace

We clearly see the initialization at sample 10, and the first state update at sample 18. Figure 10 shows the absolute correlations of our model and the real measurements at each sample. The number of traces used was 1.000.000.

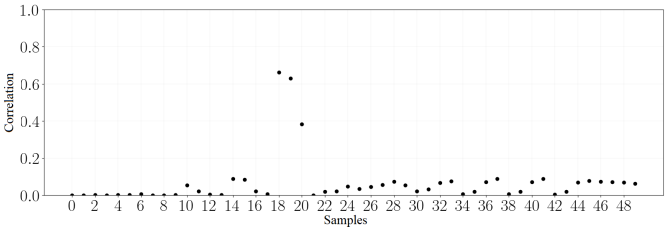

Fig. 10. Model vs. Real Measurement Correlation

The results show a high correlation at sample 18, reaching 0.66. To avoid ghost peaks, and to speed-up the initial evaluation, we focus on the sample 18. The t-statistics are shown in Figure 11. We would expect to see similar separation as in Figure 7, but unfortunately, that is not the case.

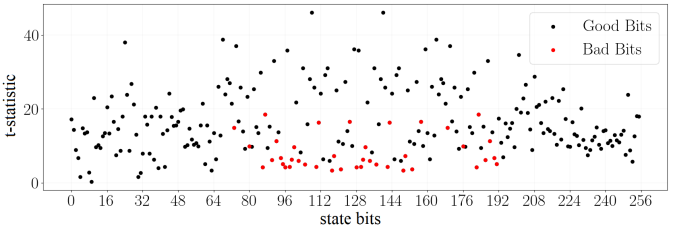

Fig. 11. Correct t-statistic per bit i for real measurements

So we increase the critical value for the t-test to ensure that all "bad bits" are correctly detected, at the cost of false identification of a good bit as a bad one. The progress of the attack, using 100.000 traces is shown in Figure 12.

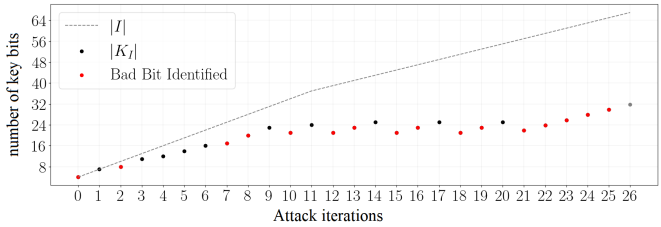

Fig. 12. Progress of the attack on real device

We have stopped the attack at iteration 27, when the size of K_I reached $2^{31.87}$. At this point, the 27th iteration would take 40 hours, and if not successful, the 28th would take 161 hours. So clearly, there are still some adjustments and improvements needed in order to make the attack feasible on real device.

The size of $\widehat{K_I}$ at the end of the last successful iteration (20th) was 2^{20} with $|I| = 2^{55}$. So 2^{35} keys were eliminated with a total of 2^{27} key hypotheses needed.

V. Future Work

The presented attack is at this moment successful only on modeled power consumption. Preliminary results on real device show a potential for the attack to succeed, but a clear identification of the "bad bits" becomes problematic with the real measurements, and will require further research.

The next step is to improve the quality of the power measurement, and investigate more sophisticated attack methods like Mutual Information and introduce a multi-bit leakage function. Also, as the structure of the "bad bits" is key dependent, it also leak some information. Therefore, we should analyze what key bits make particular bits "bad" and how these key bits must look like in such a case. We could then use this knowledge to eliminate some keys even if the target bit is "bad".

VI. Conclusion

In this paper, we demonstrate a practical side-channel attack targeting an FPGA implementation of SipHash. Starting with a simplified model, we gradually build up the complexity of the attack by introducing more realistic leakage models. By analyzing the SipRound function, we found a weakness in the state bits starting from bit $i = 160$. We then used this weakness to gradually enlarge the number of key bits used for the DPA attack in iterations, without ever exceeding the feasible amount of key hypothesizes the attack has to iterate through. Along the way, we refined our strategy to handle challenges such as incorrect keys leading to the same results as the correct one, or the presence of "bad bits".

The computational complexity is not negligible, but still in feasible range. The number of traces needed to successfully retrieve the secret key is about 10 times higher than the attack on software implementation in [7] needs, as well as the total number of key hypotheses. However, comparison of attack on software vs. hardware implementation is tricky, because of the nature of the SipRound computation (CPU vs. one-cycle combo-logic). Since there is no attempt to attack FPGA implementation of SipHash known to us at the time of writing, comparison with [7] is all we can do for now.

Although the presented attack is at this moment successful only on modeled power consumption, it demonstrates that even ARX-based designs like SipHash are not inherently immune to side-channel attacks. The results highlight the importance of thorough leakage assessment and careful implementation practices, even for algorithms that are often assumed to be inherently resistant.

Acknowledgment

This work was supported by the Czech Technical University (CTU) grant No. SGS23/208/OHK3/3T/18, by the Student Summer Research Program 2024 of FIT CTU in Prague, and the grant VJ02010010 of the Ministry of the Interior of the Czech Republic, "Tools for AI-enhanced Security Verification of Cryptographic Devices" in the program Impakt1 (2022-2025).

References

[1] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Advances in Cryptology — CRYPTO '96*. Ed. by Neal Koblitz. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 104–113.

[2] Paul Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". In: *Advances in Cryptology — CRYPTO' 99*. Ed. by Michael Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[3] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. "Electromagnetic Analysis: Concrete Results". In: *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*. CHES '01. Berlin, Heidelberg: Springer-Verlag, 2001, pp. 251–261. ISBN: 3540425217.

[4] Jean-Philippe Aumasson and Daniel J Bernstein. "SipHash: a fast short-input PRF". In: (2012), pp. 489–508.

[5] Alex Biryukov, Daniel Dinu, and Johann Großschädl. "Correlation Power Analysis of Lightweight Block Ciphers: From Theory to Practice". In: *Applied Cryptography and Network Security*. Ed. by Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider. Cham: Springer International Publishing, 2016, pp. 537–557. ISBN: 978-3-319-39555-5.

[6] Yan Yan and Elisabeth Oswald. "Examining the practical side channel resilience of ARX-boxes". In: *Proceedings of the 16th ACM International Conference on Computing Frontiers*. CF '19. Alghero, Italy: Association for Computing Machinery, 2019, pp. 373–379. ISBN: 9781450366854. DOI: 10.1145/3310273.3323399. URL: https://doi.org/10.1145/3310273.3323399.

[7] Matúš Olekšák and Vojtěch Miškovský. "Correlation power analysis of SipHash". In: *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. IEEE. 2022, pp. 84–87.

[8] Yan Yan and Elisabeth Oswald. "Examining the practical side channel resilience of arx-boxes". In: *Proceedings of the 16th ACM International Conference on Computing Frontiers*. 2019, pp. 373–379.

[9] Tobias Schneider and Amir Moradi. "Leakage assessment methodology: A clear roadmap for side-channel evaluations". In: *Cryptographic Hardware and Embedded Systems–CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings 17*. Springer. 2015, pp. 495–513.