Lightweight Implementation of the LowMC Block Cipher Protected Against Side-Channel Attacks

Javad Bahrami, Viet B. Dang, Abubakr Abdulgadir, Khaled N. Khasawneh, Jens-Peter Kaps, and Kris Gaj
jbahrami,vdang6,aabdulga,kkhasawn,jkaps,kgaj@gmu.edu
Department of Electrical and Computer Engineering
George Mason University
Fairfax, VA, U.S.A.

ABSTRACT

LowMC is a parameterizable block cipher developed for use in Multi-Party Computation (MPC) and Fully Homomorphic Encryption (FHE). In these applications, linear operations are much less expensive in terms of resource utilization compared to the non-linear operations due to their low multiplicative complexity. In this work, we implemented two versions of LowMC – unrolled and lightweight. Both implementations are realized using RTL VHDL. To the best of our knowledge, we report the first lightweight implementation of LowMC and the first implementation protected against side-channel analysis (SCA). For the SCA protection, we used a hybrid 2/3 shares Threshold Implementation (TI) approach, and for the evaluation, the Test Vector Leakage Assessment (TVLA) method, also known as the T-test. Our unprotected implementations show information leakage at 10K traces, and after protection, they could successfully pass the T-test for 1 million traces. The Xilinx Vivado is used for the synthesis, implementation, functional verification, timing analysis, and programming of the FPGA. The target FPGA family is Artix-7, selected due to its widespread use in multiple applications. Based on our results, the numbers of LUTs are 867 and 3,328 for the lightweight and the unrolled architecture with unrolling factor U = 16, respectively. It takes 14.21 μs for the lightweight architecture and 1.29 μs for the unrolled design with U = 16 to generate one 128-bit block of the ciphertext. The fully unrolled architecture beats the best previous implementation by Kales et al. in terms of the number of LUTs by a factor of 4.5. However, this advantage comes at the cost of having 2.9 higher latency.

KEYWORDS

LowMC, MPC, FHE, FPGA, TVLA, FOBOS

ACM Reference Format:
Javad Bahrami, Viet B. Dang, Abubakr Abdulgadir, Khaled N. Khasawneh, Jens-Peter Kaps, and Kris Gaj. 2020. Lightweight Implementation of the LowMC Block Cipher Protected Against Side-Channel Attacks. In *4th Workshop on Attacks and Solutions in Hardware Security (ASHES'20), November 13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3411504.3421219

ASHES'20, November 13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8090-4/20/11…$15.00
https://doi.org/10.1145/3411504.3421219

1 INTRODUCTION

Quantum computers could potentially become a serious threat to modern public-key cryptography. Therefore, in recent years, multiple novel public-key algorithms have been proposed to counter this threat. In particular, a recently developed digital signature scheme called Picnic has gained substantial interest due to its security characteristics. This scheme has been recently selected as an alternate Round 3 candidate in the NIST post-Quantum Cryptography standardization process. It has been mentioned in the proposed report by NIST [18] that this block cipher has not been studied as much as other block ciphers like AES. This statement highlights the importance of the security evaluation of this block cipher. Picnic is a public-key algorithm based on the underlying secret-key block cipher. The LowMC encryption is used to establish a secure relationship between the private and public keys. A single block u is encrypted using the key x. Then, the public verification key and the private signing key are generated based on the obtained ciphertext.

LowMC is a good candidate for such a cipher due to its low multiplicative complexity, which translates to a small number of AND gates required to implement this cipher. For instance, for the security level of 128 bits, LowMC requires 861 AND gates for full security [13]. For comparison, Simon requires 4352 [3], Fantomas 2112 [11], and Kreyvium 1537 AND gates [5]. Moreover, there exists a trade-off between the number of Sboxes and the number of rounds, and while one is decreased, another one must be increased. By doing so, one can choose whether the design's complexity comes from non-linear or linear operations.

Apart from the classical and quantum computing attacks, cryptographic cores are vulnerable to side-channel attacks. For instance, Differential Power Analysis (DPA) [14] is a serious threat to the hardware implementations of cryptographic cores. In DPA, the attacker analyzes the power consumption of the targeted device and non-invasively extracts cryptographic keys and other secret information. The DPA countermeasure used in this paper is a Threshold Implementation (TI), which is based on secret sharing and multiparty computation (MPC).

This paper is organized as follows: the previous work is discussed in the next section. The LowMC algorithm is described in Section 3. In Section 4, two unprotected architectures are introduced and analyzed. In Section 5, the SCA countermeasures and the

protected implementations are presented. In Section 6, the leakage assessment results are discussed. As depicted in this section, the protected implementations are substantially better in terms of their information leakage. Power and energy estimation are provided in the section 7, and finally, the conclusions are made in the last section of the paper.

2 PREVIOUS WORK

2.1 Hardware Implementations of LowMC

To the best of our knowledge, two hardware implementations of the LowMC block cipher have been reported to date [13]. One of these implementations follows the cipher's specification directly. The second is based on the optimization introduced by Dinur et al. [9]. This optimization reduces the time complexity and the required memory for the linear-layers from $r \cdot n^2$ to $r \cdot n^2 - (r-1) \cdot (n-s)^2$, where r is the number of rounds, n is a plaintext block size, and s is the number of bits processed by Sboxes. Both implementations have targeted Xilinx Kintex-7 and Artix-7 FPGAs. Two security levels, L1 and L5, have been supported, and the results for the unoptimized and optimized implementations have been compared with each other.

2.2 Power Analysis Attack Countermeasures

Over the years, many countermeasure techniques have been proposed to prevent power analysis attacks. As an example, the "hiding" method is about making the power consumption of a device independent of the intermediate values by either adding noise to the signal [15] or designing a circuit that consumes the same amount of power all the time [24]. This approach makes the attack harder, but an adversary can still break the implementation with a sufficient number of traces. Another class of countermeasures is "masking." This method uses random variables called masks to reduce the correlation between the secret key and the obtained leakage. Masking may still leak some information in the presence of glitches [16], [17].

In 2006, Nikova et al. proposed a new countermeasure known as Threshold Implementation (TI) [20]. TI is based on secret-sharing and is secure even in the presence of glitches. There have been a lot work targeting efficient implementation of TI, e.g., in PRESENT [21], AES [4], and Simon [23]. Diehl et al. applied TI to multiple authenticated ciphers and comprehensively compared their protection cost in [6] and [7]. Recently, Jati et al. performed a detailed analysis and design exploration using different TI techniques for GIFT [12].

3 LOWMC ALGORITHM

3.1 Introduction

LowMC is a block cipher first introduced in 2015 by Albrecht et al. This block cipher is a good candidate for Multi-Party Computation (MPC) and Fully Homomorphic Encryption (FHE), where non-linear operations impose a higher penalty compared to linear operations. An MPC protocol allows N players to jointly compute a function over inputs provided by every players. Each input is kept private and thus unknown to other players. Still, all players are guaranteed to generate the same output.

When used in such applications, LowMC offers low complexity of non-linear operations and flexible design parameters. Since the operations are defined over GF(2), specific metrics apply. First, Multiplicative Complexity (MC) is defined as the minimum number of multiplications (2-input AND gates) that are necessary and sufficient to implement a function using AND, XOR, and NOT gates. The proof of knowledge is a method in which a prover tries to convince a verifier that it knows a secret value without revealing this value itself. Since in this method the prover and the verifier interact, the proof can be lengthy. The number of AND gates in the underlying block cipher has a direct effect on the proof size. To minimize the non-linear complexity of the circuit, LowMC uses many linear (XOR) operations over GF(2). This is the reason why these linear layers are called dense layers. The non-linear part only affects a small part of the state. Hence, the number of algorithm's rounds must be increased to increase the block cipher's security. From security perspective, the most difficult computations to mask are the non-linear parts of the algorithm which are Sboxes in the LowMC block cipher. Since this block cipher has a partial non-linear operation, it requires a lower amount of randomness for protection, which makes it of great interest for applications that require SCA resistance [2, 10].

As mentioned before, LowMC has multiple parameters that the designer can change to achieve the desired effect.

3.2 LowMC Parameters

LowMC provides multiple security levels depending on the selected parameters. The algorithm's parameters are (1) the block size in bits, n, (2) the number of rounds, r, and (3) the number of bits affected by a non-linear operation, s.

Table 1: Parameters of the LowMC Block Cipher

Security Level	Block Size (n)	# of Sboxes (m)	Key Size (k)	# of Rounds (r)
L1	128	10	128	20
L3	192	10	192	30
L5	256	10	256	38

Algorithm 1 Pseudocode of LowMC Encryption

1: $k_0 \leftarrow K_0 \cdot k$
2: $s_0 \leftarrow p + k_0$
3: **for** $i = 1$ to r **do**
4: $\quad s_i \leftarrow S(s_{i-1}^{(0)}) \,||\, s_{i-1}^{(1)}$
5: $\quad s_i \leftarrow L_i \cdot s_i$
6: $\quad k_i \leftarrow K_i \cdot k$
7: $\quad s_i \leftarrow s_i + k_i$
8: $\quad s_i \leftarrow s_i + C_i$
9: **end for**
10: $c \leftarrow s_r$

The algorithm provides three different levels of security, L1, L3, and L5. As shown in Table 1, each security level has its own parameters. For instance, for security level 1, the sizes of the plaintext and Master Key are equal to 128 bits, the number of Sboxes is 10, and the number of rounds is 20. Since all the operations are over GF(2), addition is replaced by a bitwise XOR operation, and multiplication is replaced by a bitwise AND operation.

Based on the pseudocode provided in Algorithm 1 and the notations described in Table 2, the steps are as follows:

Table 2: Notation Used in the Algorithms 1 and 2

Symbol	Description
p	Plaintext
k	Master Key
k_i	Round Key
s_i	Intermediate State
r	Number of Rounds
$s_i^{(0)}$	$s = 3m$ Least Significant Bits of the State
$s_i^{(1)}$	$n - s$ Most Significant Bits of the State
$S()$	Substitution Box (Sbox) Operation
K_i	Round-Key Matrix for Round i
L_i	Linear-Layer Matrix for Round i
C_i	Constant for Round i
$\|\|$	Concatenation Operation
$\cdot$	Matrix-Vector Multiplication
$+$	XOR Operation
c	Ciphertext

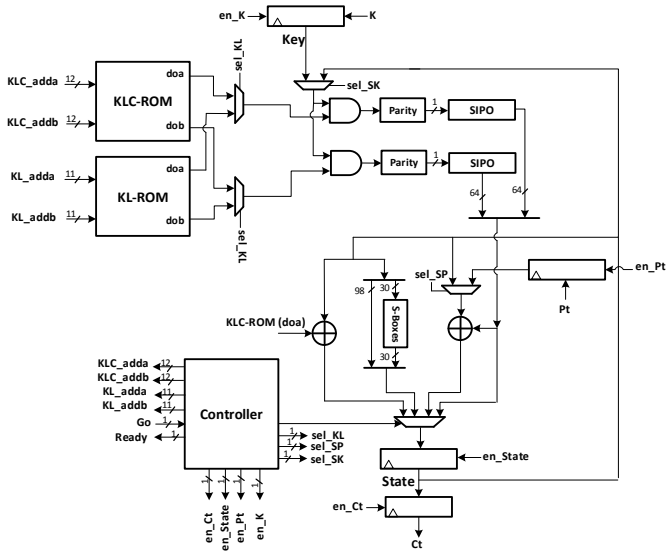

Figure 1: Top-level block diagram of the lightweight implementation. All bus widths are 128 bits, unless specified otherwise.

- **Key Whitening** $p + k_0$: The first RoundKey k_0 is XORed with the Plaintext p.
- **Sbox operation** $S(s_{i-1}^{(0)})||s_{i-1}^{(1)}$: The substitution operation is applied to $s = 3m$ least significant bits (LSBs) of the state. Hence, for the security level L1, this step modifies 30 least significant bits of the intermediate state, and 98 most-significant bits remain unchanged.
- **Linear Operation** $L_i \cdot s_i$: The linear-layer matrix for the round i is multiplied by the state s_i.
- **Constant Addition** C_i: This step combined with the previous step forms the Affine layer. A value of each constant is specific to each round.
- **Addition of the next round key** k_i: The next round key is added to the intermediate state. To generate k_i, the round key matrix K_i is multiplied by the Master Key k.

The entire process is repeated until the last round of the algorithm is completed. The number of rounds for the security levels L1, L3, and L5 is 20, 30, and 38, respectively. For the security level L1, there are 20 linear-layer matrices L_i, each having the dimensions of 128 by 128 bits. Since there is one extra key whitening step, there are 21 round key matrices K_i, and each of them has 128 rows and 128 bits in each row. Finally, there are 20 constants for each round of the algorithm, and the bit width of each constant is 128-bits. The round-key matrix K_i and linear-layer matrix L_i were selected independently and uniformly at random from all invertible matrices. The round constants C_i were selected at random as well. However, at this point, these values are fixed and treated as constants. They can be found, for example, in the reference implementation of Picnic. The choice of randomly generated invertible matrices maximizes the amount of diffusion achieved by the linear layers. It must be mentioned that it would be possible to use any sufficiently random source to generate the matrices and constants. However, no malicious sources should have an effect on these constants.

4 UNPROTECTED IMPLEMENTATIONS

4.1 Methodology

In this work, we have developed full hardware implementations using Register-Transfer Level (RTL) methodology. Since one of our targets is a lightweight implementation, the RTL methodology gives designers maximum flexibility and efficiency during the development time.

Each design is divided into the Datapath and Controller, and then encoded in VHDL. Taking into account a large number of constants used by LowMC, block memories (BRAMs) are employed. Functional verification, synthesis, and implementation steps are done using Xilinx Vivado. Our target FPGA family is Artix-7, which has the lowest power and cost among the Xilinx 7 Series FPGAs. The selected device is Artix-7 xc7a100tftg256-3. It has 63,400 LUTs, 126,800 flip flops, 15,850 slices, and 135 36-kbit BRAMs.

4.2 Lightweight Architecture

The lightweight architecture of LowMC is shown in Figure 1. The round key matrices K_i, linear-layer matrices L_i, and round constants C_i are stored in two memories called KLC-ROM and KL-ROM. These memories have n-bit words and the number of locations occupied in these memories are given by $(r+1) \cdot n$ for K_i, $r \cdot n$ for L_i, and r for C_i. The actual size of these memories are the smallest power of two greater than or equal to the number of memory locations used. For example, for the security level L1, the dimension of each K_i matrix is 128 by 128. Since we have 21 K_i matrices for the entire algorithm, the number of rows for the round-key matrices is $21 \times 128 = 2688$. Additionally, the number of rows for the linear-layer matrices is $20 \times 128 = 2560$, and the number of rows required for the round constants is 20. We stored K_0 to K_{15}, all the round constants C_i, and L_1 to L_{15} in the first memory called KLC-ROM, and K_{16} to K_{20}

and L_{16} to L_{20} in the second memory called KL-ROM. By using this memory structure, the number of unused locations in the memories, and as a result, the number of BRAMs is minimized. Since the first memory has 32×128 rows, it has a 12-bit address bus. The second memory has a 11-bit address bus since the number of occupied rows in the second memory is $10 \times 128 = 1280 < 2^{11}$. For increasing the processing speed of the implementation, we used dual-port BRAMs. As a result, each 128-row matrix is divided into two 64-row parts processed in parallel. In Artix-7, one 36-kbit BRAM can be configured to implement 8k x 4 bits ROM. The $4k \times 128$ bits KL-ROM requires 16 BRAMs and $2k \times 128$ bits KLC-ROM requires 8 BRAMs. As a result, the total numbers of required BRAMs is at most 24. Taking into account the synthesis tool's optimizations, the final number of BRAMs appeared to be 22.

The symbol of an AND gate represents 128 AND gates working in parallel. They execute concurrently 128 multiplications in GF(2). These multiplications are used to calculate the bitwise product of the Key and a row of the Matrix K_i, and the bitwise product of the State and a row of the matrix L_i. The Parity function calculates the parity of the obtained product, which is a 128-bit vector. The parity is one if an odd number of bits in a vector is equal to 1. This function is equivalent to a 128-input XOR. In Artix-7, the 128-bit Parity function can be implemented using three levels of 6-input LUTs. After level 1, the number of intermediate values is equal to $\lceil 128/6 \rceil = 22$. After level 2, it becomes $\lceil 22/6 \rceil = 4$. The final 4-bit XOR requires just 1 LUT. Thus, the total number of LUTs required is 22 + 4 + 1 = 27. One-bit outputs of Parity functions are inputs to two 1-to-64-bit Serial-In Parallel-Out (SIPO) units. Each 64-bit vector represents one half of a matrix by vector product. The final 128-bit value is formed using the concatenation operation.

The middle part of the circuit is responsible for the operations such as $s_i \leftarrow S(s_{i-1}^{(0)}) || s_{i-1}^{(1)}$, $s_i \leftarrow L_i \cdot s_i$, $s_i \leftarrow s_i + k_i$, and $s_i \leftarrow s_i + C_i$. Sboxes form the non-linear part of the algorithm and they only apply to $3 \cdot m$ bits of the intermediate state. This number is equal to 30 for all security levels summarized in Table 1.

The K port is an input for the Master Key, and the Pt port is used to enter Plaintext. The encryption starts when an external circuit activates the input Go. When the encryption of one 128-bit block of data is completed, the Ciphertext block is released at the output Ct, and the Ready signal becomes equal to one.

We used Xilinx Vivado for timing analysis of our design, and based on its report, the critical path of our design starts at the outputs of the synchronous-read KLC-ROM and ends at the input of each Serial-In Parallel-Out (SIPO) unit. This path includes two main components – a two-input AND gate and the Parity unit. Since we use a dual-port memory in this architecture, we have two copies of this path present in Figure 1.

4.3 Unrolled Architecture

One of the main drawbacks of the lightweight architecture is its large cycle count. As shown in Table 3, it takes more than 2,600 clock cycles to encrypt a single plaintext block. This number of clock cycles has been reduced substantially in the unrolled architecture shown in Figure 2.

This design can be generalized by unrolling logic U times, where U = 2, 4, 8, and 16. By using the notation $u = \log_2 U$, each memory

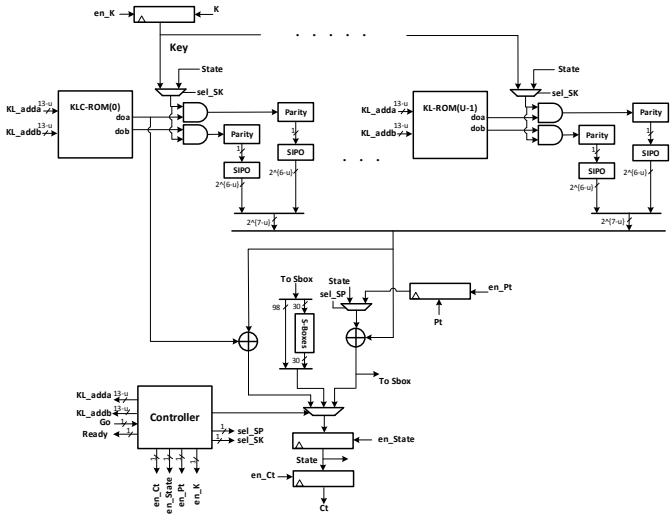

Figure 2: Generalized top-level block diagram of the unrolled implementation. All bus widths are 128 bits, unless specified otherwise.

has $\leq 2^{13-u}$ words and the address width becomes $13 - u$ bits. For $U = 16$ ($u = 4$), each memory has $2^{13-4} = 2^9$ words $\rightarrow 4 \times (1k \times 32)$ = 4 BRAMs. Thus, total number of BRAMs is equal to 16×4 BRAMs = 64 BRAMs. Based on the value of u, the bus width after SIPOs is 2^{6-u}, and after the concatenation operation, it is 2^{7-u} .

The main idea is to perform computations simultaneously on $2 \cdot U$ rows of matrices K_i and L_i. In order to do that, the memories storing K_i and L_i have been divided into U equal parts. Each part contains $128/U$ consecutive rows of the mentioned above matrices. Thus, the size of each ROM becomes $2^{13}/2^u = 2^{13-u}$ words. Each ROM is a dual-port ROM, and thus, two rows with indices j and j+$64/U$ can be accessed simultaneously. We use 20 unused locations in ROM(0) to store the round constants C_i. Therefore, the leftmost memory is denoted in Figure 2 as KLC-ROM(0), and all remaining ones as KL-ROM(j), with j=1..$U - 1$.

In Artix-7 FPGAs, each 36-kbit BRAM, when configured as a dual-port ROM, has the maximum word width of 36 bits. The number of required BRAMs is a function of U. As shown in Table 3, it varies between 22 and 32 for U between 1 and 8. Then, it doubles with each subsequent doubling the value of U.

Algorithm 2 Unrolled LowMC Pseudocode

$k_0 \leftarrow K_0 \cdot k$
$s_0 \leftarrow S((p + k_0)^{(0)}) \,||\, (p + k_0)^{(1)}$
for $i = 1$ to $r - 1$ **do**
  $s_i \leftarrow (L_i \cdot s_{i-1}) + C_i$
  $k_i \leftarrow K_i \cdot k$
  $s_i \leftarrow S((s_i + k_i)^{(0)}) \,||\, (s_i + k_i)^{(1)}$
end for
$s_r \leftarrow (L_r \cdot s_{r-1}) + C_r$
$k_r \leftarrow K_r \cdot k$
$c \leftarrow s_r + k_r$

The middle part of the circuit has also been slightly changed compared to the lightweight architecture to reduce the number of clock cycles by $2 \cdot r$. This reduction is accomplished by performing a) the round key addition and the Sbox operation in the same clock cycle, b) $s_i \leftarrow (L_i \cdot s_{i-1}) + C_i$ in a single clock cycle, when $L_i \cdot s_{i-1}$ is already available at the outputs of the circuit SIPOs. The modified pseudocode, reflecting this change is shown above as Algorithm 2.

Further decreasing the number of clock cycles would require doubling the number of BRAMs, which would make them highly underutilized. Instead, matrices K_i and L_i, as well as constants C_i, could be stored in distributed memory located in LUTs. However, doing that would dramatically increase the number of LUTs required by the implementation. A similar approach, with some additional optimizations is reported in [13].

4.4 Analysis of Results

In Table 3, we summarize the results for the lightweight and unrolled architectures for different values of the unrolling factor U, implemented using Artix-7 FPGAs. The provided ratios are over the lightweight implementation ($U = 1$). The unrolled architecture with $U = 16$ has over 11 times smaller cycle count at the cost of increasing the number of LUTs by a factor of 3.84, slices by a factor of 2.89, and BRAMs by a factor of 2.91. The number of flip-flops remains almost the same, and the maximum clock frequency goes down by 7%. As a result, the throughput increases by a factor of 11.05.

In Table 4, we compare results for our fastest unrolled architecture with the results for the optimized high-speed architecture reported in [13]. Since only the results for Kintex-7 are reported in [13], we regenerated our results targeting the same device of the same family. At the cost of allowing the use of 64 vs. 0 BRAMs, our architecture uses 4.5 times less LUTs, and 2.4 times fewer slices. The maximum clock frequency almost doubles. However, the throughput and latency are worse by a factor of approximately 2.92. Thus, our fully unrolled architecture is particularly suitable for applications with medium speed requirements, in which LowMC shares resources with functions using a bigger percentage of the FPGA slices than FPGA BRAMs.

In [13], LowMC is used as an One-Way Function (OWF) within the post-quantum cryptography (PQC) digital signature scheme Picnic. Considering that three instances of LowMC are required to implement Picnic, it is quite important to reduce and balance the number of required FPGA resources. We believe that our unrolled architecture with unrolling factor $U = 16$ accomplishes this goal, while still maintaining very high speed.

At the same time, a lightweight implementation of Picnic can be based on our lightweight implementation of LowMC, which is the first such an implementation reported in the literature to date.

5 THRESHOLD IMPLEMENTATIONS

5.1 TI Protection Methodology

Threshold Implementation (TI) is an algorithmic countermeasure against power analysis side-channel attacks. This method is based on the secret-sharing and Multi-Party Computation (MPC) concepts. The attacker cannot steal secret information by looking at any individual shares. In traditional Boolean masking, the protected implementations are vulnerable when the circuit is not glitch-free. This is because in the CMOS technology, the power changes during switching activities arising from glitches in the circuit. These transitions are relatively large compared to the normal activity of the circuit. As a result, they make the circuit susceptible to attacks. Three properties of a secure threshold implementation are:

- **Non-completeness**: Every function is independent of at least one share of each of the input variables.
- **Correctness**: The sum of the output shares gives the desired output.
- **Uniformity**: The output distribution preserves the input distribution.

A non-linear function of algebraic degree 2, such as a 2-input AND gate, can be shared using three TI shares since $d + 1$ shares are required to share a function of the degree d. However, achieving the TI uniformity property is not trivial. In this paper, we achieve this property by supplying fresh random bits (e.g., "resharing" or "remasking" randomness, used in [19]), coming from a Pseudorandom Number Generator (PRNG).

In LowMC, all operations except Sboxes are linear, or in other words, have degree 1. Hence, we apply a 2-share implementation to them. The original – without protection using Threshold Implementation (TI) – structure of the Sboxes used in this algorithm is shown in Figure 3. This structure is used to process the first $3 \cdot m$ least significant bits (LSBs) of the intermediate $State$. We apply a 3-share TI to AND gates, while having 2-share TI for the linear XOR gates. Since almost all parts of the circuit have 2-share TI, the inputs of the AND gates must be reshared from 2-share into 3-share. The original 2-share input of the AND gates is: x_a, x_b (two shares of x input) and y_a, y_b (two shares of y input). For generating different shares in this method, different random numbers – R_1, R_2, and R_3 – should be generated using PRNG. Resharing step, described in [22], is as follows:

$$
\begin{aligned}
x_1 &\leftarrow x_a \oplus R_2, \\
x_2 &\leftarrow x_b, \\
x_3 &\leftarrow R_2 \\
y_1 &\leftarrow y_a \oplus R_1, \\
y_2 &\leftarrow y_b, \\
y_3 &\leftarrow R_1
\end{aligned}
$$

To satisfy property 3, random masks M_1, M_2, and M_3 should be generated such that $M_1 \oplus M_2 \oplus M_3 = 0$.

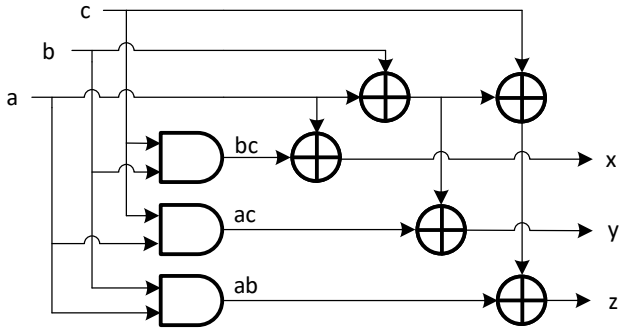

Figure 3: Substitution Box (Sbox) in the LowMC

Table 3: Results for the unprotected implementations on Artix-7

Metric	Lightweight (U=1)	Unrolled (U = 2)	Ratio (U = 2)	Unrolled (U = 4)	Ratio (U = 4)	Unrollel (U = 8)	Ratio (U = 8)	Unrolled (U = 16)	Ratio (U = 16)
LUTs	867	848	0.98	1,075	1.24	1,560	1.80	3,328	3.84
Flip Flops	658	667	1.01	665	1.01	663	1.01	661	1.00
Slices	470	400	0.85	549	1.17	765	1.63	1,356	2.89
Block RAMs	22	29	1.32	30	1.36	32	1.45	64	2.91
Max. Freq. (MHz)	189	198	1.05	187	0.99	181	0.96	175	0.93
Cycle Count	2,685	1,373	0.51	717	0.27	389	0.14	225	0.08
Latency (us)	14.21	6.93	0.49	3.83	0.27	2.15	0.15	1.29	0.09
Throughput (Mbits/s)	9.01	18.46	2.05	33.38	3.71	59.56	6.61	99.56	11.05

Table 4: Results for the unprotected unrolled implementations with (U=16) on Kintex-7

Metric	Unrolled (U = 16)	Kales et al. [13]	Ratio
LUTs	3,003	13,558	4.51
Flip Flops	661	504	0.76
Slices	1,601	3,838	2.40
Block RAMs	64	0	0.00
Max. Freq. (MHz)	241	125	0.52
Cycle Count	225	40	0.18
Latency (us)	0.93	0.32	0.34
Throughput (Mbits/s)	137	400	2.92

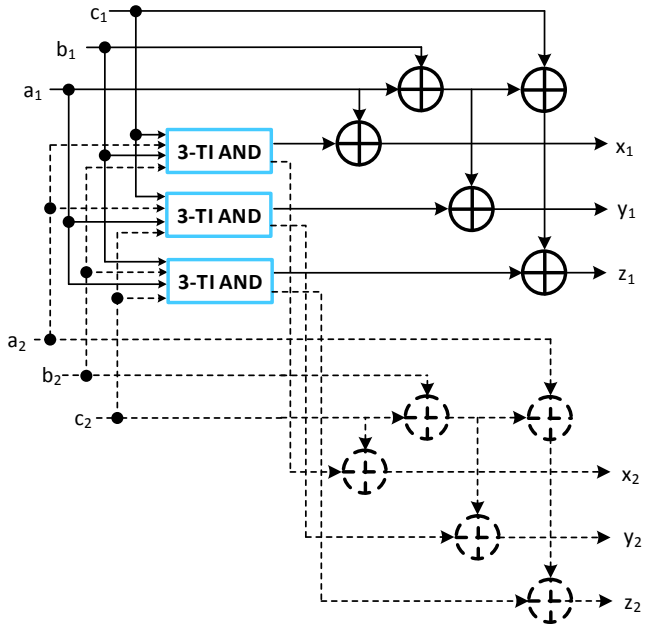

Figure 4: 3-share threshold implementation of the Sbox in the LowMC

$$M_1 \leftarrow R_1R_3 \oplus R_2R_3 \oplus R_3,$$
$$M_2 \leftarrow R_1R_3 \oplus R_2R_3,$$
$$M_3 \leftarrow R_3$$

The shared outputs are computed as follows:

$$z_1 \leftarrow x_2y_3 \oplus x_3y_2 \oplus x_2y_2 \oplus M_1,$$
$$z_2 \leftarrow x_3y_1 \oplus x_1y_3 \oplus x_3y_3 \oplus M_2,$$
$$z_3 \leftarrow x_1y_2 \oplus x_2y_1 \oplus x_1y_1 \oplus M_3$$

Finally, the output with three shares should be converted back to two shares using the following equations:

$$o1 \leftarrow z_1 \oplus z_2,$$
$$o2 \leftarrow z_3,$$

After applying these changes to the Sboxes in the LowMC, we used it in the protected versions of our implementations. The internal structure of Sboxes after this modification is shown in Figure 4. The conventional AND gates were replaced by the protected AND gates, and the number of XOR gates was doubled to support two output shares.

For evaluation, we first investigated the information leakage of the unprotected lightweight and the unprotected unrolled implementations using TVLA with 10,000 fixed-versus-random traces. Afterward, we modified the implementations in a way that make them protected against side-channel attacks.

5.2 FOBOS Platform

The Flexible Open-source workBench fOr Side-channel analysis (FOBOS) is an SCA platform suitable for data acquisition and analysis [1, 25, 26]. This platform consists of a data acquisition module used to collect power traces and an analysis module used to mount side-channel attacks and perform leakage assessment. In this work, we specifically use FOBOS2 with the Digilent Basys 3 control board, and the Picoscope 5000 USB-based oscilloscope. FOBOS2 is compatible with the NewAE CW305 (an Artix-7 FPGA target board), which we use to instantiate the Design-Under-Test (DUT).

The overall operation of FOBOS2 is described below. All test vectors are generated using Python scripts on the control PC. The control PC passes the test vectors one at a time to the control board using a USB interface. The control board forwards the test vectors to the DUT board, which runs cryptographic operations and returns results to the control board and to the PC. While the DUT is performing the operations, the control PC collects the instantaneous

power consumption of the DUT using the oscilloscope. The control board handles the DUT clock and the oscilloscope trigger, among other functions.

At the DUT side, FOBOS2 provides a simple wrapper that receives data from the control board and distributes it into three FIFOs. (1) The Public Data Input (PDI) FIFO (2) The Secret Data Input (SDI) FIFO (3) The Random Data Input (RDI) FIFO. The PDI FIFO is used to store plaintext, the SDI FIFO is used for the key, and the RDI FIFO is used to store random data used by protected implementations. Once the data is stored by the wrapper into the FIFOs, the DUT is allowed to run, and its output is accumulated in the Data Output (DO) FIFO. This data is forwarded back to the control board, as discussed previously.

The analysis module uses the traces collected by the control PC as input.

5.3 Protected Implementations

Since our designs are divided into two parts, datapath and controller, we discuss each part separately. Additionally, we tried to follow the hierarchical design rules, which make the modification phase required for the protection more time-efficient and easier.

A. Datapath: Since the datapath consists of multiple parts, we discuss each part individually:

Constants: The constants are reused in the protected implementations without any modifications.

I/Os: In the TI protection method, the Master Key, Plaintext, and Ciphertext should be divided into two shares. Hence, instead of using single input/output for each of them, we used two inputs for the Master Key and Plaintext, and two outputs for the Ciphertext.

Sbox: As mentioned before, the internal structure of the Sbox is modified, so the old version is replaced with the protected Sbox.

PRNG: For generating random bits used by protected Sboxes, we use the lightweight block cipher Trivium. Our implementation of Trivium generates 64 bits per clock cycle. Since for each operation of protected Sboxes, 90 bits of random data are required, one extra register is used to store 64 bits generated ahead of time.

AND Gate: The AND gates located in the critical path of our implementations do not need to be protected. The reason is that these AND gates operate on individual shares of the secret data, and revealing their information does not provide any useful information to the attacker.

B. Controller: The controllers are the same as in the unprotected implementations except for additional control and status signals used for the communication with the PRNG.

The block diagrams of the protected implementations are shown in Figures 10 and 11 in Appendix A.

The comparison between the unprotected and protected implementations is summarized in Table 5. An overhead of the protected implementations in terms of the number of LUTs fluctuates as a function of U, but remains in the relatively small range between 2.06 and 2.27. These variations are smaller in case of flip-flops (2.87-2.89), and higher in case of Slices (1.54-1.90). The cycle counts and the numbers of BRAMs remain exactly the same. Since the critical paths of our implementations are unchanged in protected versions, the maximum clock frequencies of the protected designs are only slightly lower compared to the unprotected ones. As a result, we are also able to maintain almost the same throughput as in the unprotected implementations.

In [8], Diehl et al. provided a descriptive comparison between unprotected and protected implementations of four lightweight block ciphers TWINE, SIMON, PRESENT, and LED. The general-purpose secret-key block-cipher standard AES was implemented as well using architectures suitable for lightweight applications. The results of the comparison among these five ciphers is shown in Table 6. Since the results presented in [8] were obtained using Spartan-3E, for the fair comparison we regenerated the results for all architectures of LowMC using a very similar FPGA from the Spartan-3 family, capable of handling the sufficient number of Block RAMs. The obtained results are shown in Table 7.

The unprotected implementations of LowMC use a significantly larger number of LUTs than all investigated lightweight block ciphers. Only the unprotected implementation of AES uses more LUTs than LowMC with $U = 1$ and $U = 2$. Additionally, all implementations presented in [8] did not use any Block RAMs, while the implementations of LowMC require between 47 and 64 BRAMs. The unprotected implementations of lightweight block ciphers and AES also operate at a higher clock frequency than all implementations of LowMC. Additionally, they outperform LowMC in terms of Throughput. The relative cost of protection in terms of the number of LUTs is higher for all ciphers investigated in [8], except AES. The negative effect of the SCA countermeasures on the maximum clock frequency and Throughput is the lowest in the case of SIMON and LowMC. The protected implementation of LowMC with $U = 1$ requires fewer LUTs than the protected implementation of TWINE, and only about one third more than the protected implementations of AES, LED, and SIMON. Overall, the protected implementations of LowMC are competitive with the protected implementations of lightweight block ciphers in terms of the number of LUTs, but not in terms of the number of BRAMs and Throughput.

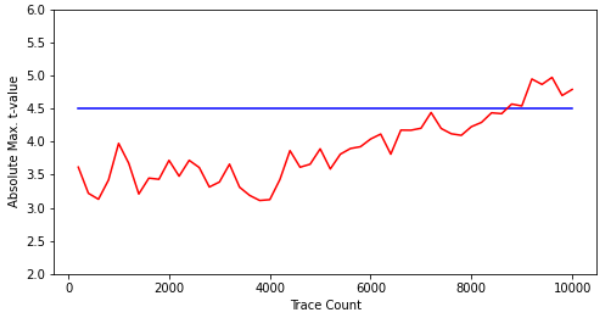

Figure 5: T-Value of the unprotected lightweight implementation vs. trace count

6 TVLA RESULTS

TVLA was performed on the protected and unprotected versions of both the lightweight and the unrolled implementations of LowMC. The designs were instantiated in the NewAE CW305 board. This board uses a Xilinx Artix-7 xc7a100tftg256-3 FPGA as a target. The

Table 5: Results for the unprotected and protected unrolled implementations on Artix-7

Metric	U = 1			U = 2			U = 4			U = 8			U = 16		
	Unpr.	Pr.	Ratio	Unpr.	Pr.	Ratio	Unpr.	Pr.	Ratio	Unpr.	Pr.	Ratio	Unpr.	Pr.	Ratio
LUTs (6 Inputs)	867	1,813	2.09	848	1,922	2.27	1,075	2,398	2.23	1,560	3,523	2.26	3,328	6,860	2.06
Flip Flops	658	1,904	2.89	667	1,913	2.87	665	1,911	2.87	663	1,909	2.88	661	1,907	2.89
Slices	470	724	1.54	400	759	1.90	549	916	1.67	765	1,217	1.59	1,356	2,317	1.71
Block RAMs	22	22	1.00	29	29	1.00	30	30	1.00	32	32	1.00	64	64	1.00
Max. Freq. (MHz)	189	172	0.91	198	182	0.92	187	178	0.95	181	175	0.97	175	159	0.91
Cycle Count	2,685	2,685	1.00	1,373	1,373	1.00	717	717	1.00	389	389	1.00	225	225	1.00
Latency (us)	14.21	15.61	1.10	6.93	7.54	1.09	3.83	4.03	1.05	2.15	2.22	1.03	1.29	1.42	1.10
Throughput (Mbits/s)	9.01	8.20	0.91	18.46	16.97	0.92	33.38	31.78	0.95	59.56	57.58	0.97	99.56	90.45	0.91
Power (mW)	299	315	1.05	316	355	1.12	337	404	1.20	412	480	1.17	794	951	1.20
E/bit (nJ/bit)	33.2	38.4	1.16	17.1	20.9	1.22	10.1	12.7	1.26	6.9	8.3	1.20	8.0	10.5	1.32

Table 6: Results for the unprotected and protected implementations of lightweight block cihpers and AES on Spartan-3E [8]

Metric	TWINE			SIMON			PRESENT			LED			AES		
	Unpr.	Pr.	Ratio	Unpr.	Pr.	Ratio	Unpr.	Pr.	Ratio	Unpr.	Pr.	Ratio	Unpr.	Pr.	Ratio
LUTs (4 Inputs)	296	2,946	9.95	565	2,151	3.80	595	1,707	2.86	727	2,175	2.99	1,182	2,387	2.01
Slices	229	1,777	7.75	403	1,404	3.48	408	1,221	2.99	486	1,290	2.65	806	1,736	2.15
Max. Freq. (MHz)	200	67	0.33	176	176	1	177	70	0.39	116	55	0.47	128	86	0.67
Throughput (Mbits/s)	355	118	0.33	329	326	0.99	366	143	0.39	134	73	0.54	94	63	0.67

Table 7: Results for the unprotected and protected implementations of LowMC on Spartan-3

Metric	U = 1			U = 2			U = 4			U = 8			U = 16		
	Unpr.	Pr.	Ratio	Unpr.	Pr.	Ratio	Unpr.	Pr.	Ratio	Unpr.	Pr.	Ratio	Unpr.	Pr.	Ratio
LUTs (4 Inputs)	1,025	2,929	2.85	1,156	3,157	2.73	1,429	3,952	2.76	2,183	5,232	2.39	3,445	7,964	2.31
Flip Flops	673	1,914	2.84	670	1,918	2.86	676	1,917	2.83	666	1,915	2.87	668	1,927	2.88
Slices	739	2,084	2.82	847	2,188	2.58	939	2,588	2.75	1,362	3,223	2.36	1,961	4,629	2.36
Block RAMs (RAMB16s)	47	47	1.00	64	64	1.00	60	60	1.00	64	64	1.00	64	64	1.00
Max. Freq. (MHz)	90	82	0.91	79	73	0.92	75	67	0.89	68	66	0.97	75	72	0.96
Cycle Count	2,685	2,685	1.00	1,373	1,373	1.00	717	717	1.00	389	389	1.00	225	225	1.00
Latency (us)	29.92	32.50	1.08	17.34	18.66	1.07	9.51	10.5	1.11	5.64	5.86	1.03	2.98	3.09	1.03
Throughput (Mbits/s)	4.27	3.93	0.92	7.38	6.85	0.92	13.45	12.19	0.91	22.69	21.84	0.96	42.95	41.42	0.96

FOBOS2 control board was used to control data communication with the DUT and control the DUT clock. The DUT was run at 1 MHz to reduce signal distortion, which is favorable to attackers. Picoscope 5000 was used for power measurements at a sampling frequency of 125 MS/s. The CW305 low-noise amplifier (LNA) was used to amplify the voltage drop across a 0.1 Ω resistor located between the power supply and the Artix-7 FPGA.

The generation of input shares corresponding to multiple fixed-vs-random traces was performed in software. The TVLA results for the unprotected lightweight and unrolled designs are shown in Appendix A in Figures 12 and 13. As expected, both unprotected designs leak information, as indicated in the figures by the t-value exceeding the threshold of $|t| = 4.5$. 10,000 traces were sufficient to demonstrate the significant leakage.

Figures 5 and 6 show how the t-values increase as we analyze more traces. The x-axis shows the number of traces, and the y-axis shows the maximum absolute t-value observed. It is clear that the unprotected unrolled implementation fails the test at a very low number of traces compared to the unprotected lightweight implementation, which consumes much less power.

On the other hand, Figures 7 and 8 show the TVLA results of the protected lightweight and unrolled implementations, respectively. In this case, we collected and analyzed 1 million traces, and still, no significant leakage can be observed. This experiment confirms the effectiveness of the implemented countermeasures.

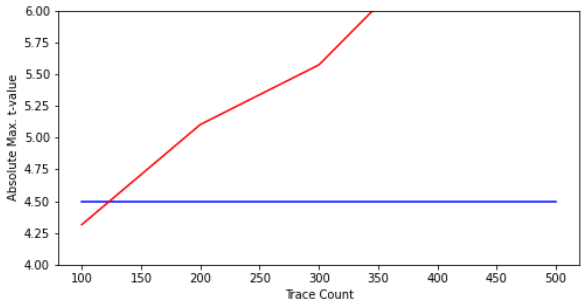

Figure 6: T-Value of the unprotected unrolled implementation vs. trace count

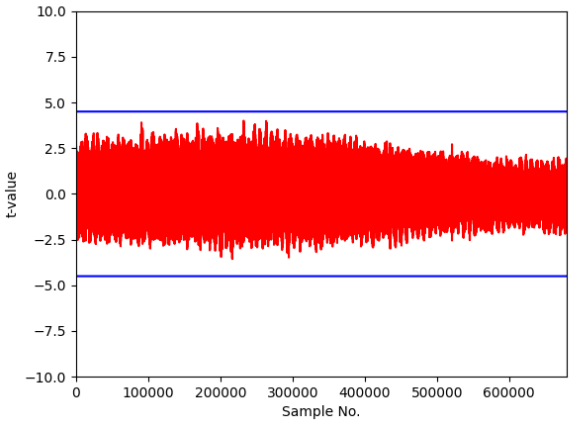

Figure 7: Fixed-versus-random TVLA of the protected lightweight implementation (1 million traces)

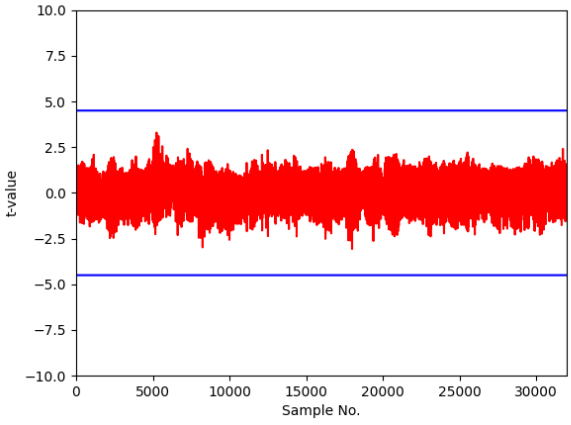

Figure 8: Fixed-versus-random TVLA of the protected unrolled implementation with $U = 16$ (1 million traces)

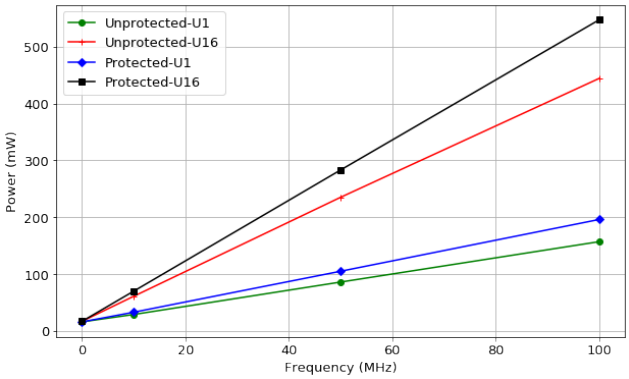

Figure 9: LowMC power estimations. Power was estimated at 10, 50 and 100 MHz.

7 POWER AND ENERGY ESTIMATION

Power consumption and energy per bit (E/bit) are major concerns in cryptographic implementations since they are directly related to power supply characteristics and battery life. We estimated power consumption using Vivado for protected and unprotected designs. Two corner cases were studied, lightweight (U=1) and fully unrolled (U=16). Power was estimated at three different frequencies; 10, 50, and 100 MHz and only the core FPGA supply Vcc_{int} was considered. The device used is Xilinx Artix-7 xc7a100tftg256-3. We performed vector-less post-place-and-route power estimation, and we set a toggle rate to 50% and ambient temperature to 25°C. The results are shown in Figure 9.

We observe that for all designs, the static power is very similar, about 15-17 mW. Also, as expected, power increases linearly with frequency. Furthermore, the unrolling factor U, and the existence of protection significantly affects the power consumption. We also observed that the BRAM power consumption constitutes 47-84% of total dynamic power.

For E/bit, we estimated the power consumption for each design ($U = 1$ to $U = 16$) at its maximum frequency using the same methodology described above. We then calculated $E/bit(nJ/bit) = Power(mW)/Throughput(Mbps)$. The results are listed in Table 5. We observe that the energy per bit generally trends down as the unrolling factor U increases, with the exception that E/bit for U=8 is slightly smaller than E/bit for U=16.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we first introduced the LowMC block cipher and the motivation behind using it inside of other cryptographic schemes. Two unprotected and the corresponding two protected architectures have been proposed, implemented, and analyzed. These implementations demonstrate the capability of using BRAMs to substantially reduce the number of LUTs and Slices required to implement LowMC. In particular, our unprotected lightweight implementation uses only 867 LUTs and our unprotected unrolled implementation with $U = 16$ 3,328 LUTs. Our fastest unrolled architecture uses over 4.5 fewer slices than the best high-speed implementation of LowMC reported to date. At the same time, it is slower in terms

of throughput by a factor of 2.92. Threshold implementations provide substantial improvements in the information leakage for both architectures. The lack of detectable leakage was verified experimentally using TVLA with 1 million traces. These implementations increase the number of LUTs by approximately a factor of 2.09 for the lightweight architecture and a factor of 2.06 for the unrolled architecture with the unrolling factor $U = 16$. They do not affect the number of BRAMs, and have a negligible effect on the circuit throughput and latency.

Our next goal is to incorporate the algorithmic optimizations by Dinur et al. [9] and make the corresponding implementations protected against side-channel attacks. Both lightweight and high-speed architectures will be implemented and analyzed. Additionally, we will use this work as a starting point for efficient and side-channel resistant implementations of the entire Picnic signature scheme.

REFERENCES

[1] Abubakr Abdulgadir, William Diehl, and Jens-Peter Kaps. Dec. 9-11, 2019. An Open-Source Platform for Evaluation of Hardware Implementations of Lightweight Authenticated Ciphers. In *2019 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2019*. Cancun, Mexico.

[2] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. 2015. Ciphers for MPC and FHE. In *Advances in Cryptology – EUROCRYPT 2015*, Vol. 9056. Springer Berlin Heidelberg, Berlin, Heidelberg, 430–454. https://doi.org/10.1007/978-3-662-46800-5_17

[3] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. 2015. The SIMON and SPECK Lightweight Block Ciphers. In *Proceedings of the 52nd Annual Design Automation Conference, DAC 2015*. ACM Press, San Francisco, California. https://doi.org/10.1145/2744769.2747946

[4] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. 2014. A More Efficient AES Threshold Implementation. In *Progress in Cryptology – AFRICACRYPT 2014*. Vol. 8469. Springer International Publishing, Cham, 267–284. https://doi.org/10.1007/978-3-319-06734-6_17

[5] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. 2018. Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression. *J Cryptol* 31, 3 (July 2018), 885–916. https://doi.org/10.1007/s00145-017-9273-9

[6] William Diehl. 2018. *Comparing Costs of Protecting Secret Key Ciphers Against Differential Power Analysis*. Ph.D. Dissertation. George Mason University, Fairfax, VA.

[7] William Diehl, Abubakr Abdulgadir, Farnoud Farahmand, Jens-Peter Kaps, and Kris Gaj. 2018. Comparison of Cost of Protection against Differential Power Analysis of Selected Authenticated Ciphers. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2018*. Washington, DC, 147–152. https://doi.org/10/ggbsck

[8] William Diehl, Abubakr Abdulgadir, Jens-Peter Kaps, and Kris Gaj. 2017. Side-Channel Resistant Soft Core Processor for Lightweight Block Ciphers. In *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, Cancun, 1–8. https://doi.org/10/ggbsfn

[9] Itai Dinur, Daniel Kales, Angela Promitzer, Sebastian Ramacher, and Christian Rechberger. 2019. Linear Equivalence of Block Ciphers with Partial Non-Linear Layers: Application to LowMC. In *Advances in Cryptology – EUROCRYPT 2019 (LNCS, Vol. 11476)*. Springer International Publishing, Cham, 343–372. https://doi.org/10.1007/978-3-030-17653-2_12

[10] B. Gérard, Vincent Grosso, M. Naya-Plasencia, and François-Xavier Standaert. 2013. Block Ciphers That Are Easier to Mask: How Far Can We Go?. In *Cryptographic Hardware and Embedded Systems - CHES 2013*, Vol. 8086. Springer Berlin Heidelberg, Berlin, Heidelberg, 383–399. https://doi.org/10.1007/978-3-642-40349-1_22

[11] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varıcı. 2015. LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. In *Fast Software Encryption*. Vol. 8540. Springer Berlin Heidelberg, Berlin, Heidelberg, 18–37. https://doi.org/10.1007/978-3-662-46706-0_2

[12] Arpan Jati, Naina Gupta, Anupam Chattopadhyay, Somitra Kumar Sanadhya, and Donghoon Chang. 2020. Threshold Implementations of GIFT : A Trade-Off Analysis. *IEEE Trans.Inform.Forensic Secur.* 15 (2020), 2110–2120. https://doi.org/10.1109/TIFS.2019.2957974

[13] Daniel Kales, Sebastian Ramacher, Christian Rechberger, Roman Walch, and Mario Werner. 2020. Efficient FPGA Implementations of LowMC and Picnic. In *The Cryptographers' Track at the RSA Conference 2020, CT-RSA 2020*. Springer, San Francisco.

[14] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Diferential Power Analysis. In *CRYPTO '99 - 19th International Conference on Cryptology*. Santa Barbara, CA.

[15] Stepfan Mangard, Elisabeth Oswald, and Thomas Popp. 2008. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Advances in Information Security, Vol. 31. Springer Science & Business Media.

[16] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. 2005. Side-Channel Leakage of Masked CMOS Gates. In *Topics in Cryptology – CT-RSA 2005*, Vol. 3376. Springer Berlin Heidelberg, Berlin, Heidelberg, 351–365. https://doi.org/10.1007/978-3-540-30574-3_24

[17] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. 2005. Successfully Attacking Masked AES Hardware Implementations. In *Cryptographic Hardware and Embedded Systems – CHES 2005*. Vol. 3659. Springer Berlin Heidelberg, Berlin, Heidelberg, 157–171. https://doi.org/10.1007/11545262_12

[18] Dustin Moody, Gorjan Alagic, Daniel C. Apon, David A. Cooper, Quynh H. Dang, John M. Kelsey, Yi-Kai Liu, Carl A. Miller, Rene C. Peralta, Ray A. Perlner, Angela Y. Robinson, Daniel C. Smith-Tone, and Jacob Alperin-Sheriff. 2020. *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*. Technical Report. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8309

[19] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. 2011. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *Advances in Cryptology – EUROCRYPT 2011*. Vol. 6632. Springer Berlin Heidelberg, Berlin, Heidelberg, 69–88. https://doi.org/10.1007/978-3-642-20465-4_6

[20] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. 2006. Threshold Implementations Against Side-Channel Attacks and Glitches. In *Information and Communications Security, ICICS 2006 (LNCS, Vol. 4307)*. Springer Berlin Heidelberg, 529–545. https://doi.org/10.1007/11935308_38

[21] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. 2011. Side-Channel Resistant Crypto for Less than 2,300 GE. *J Cryptol* 24, 2 (April 2011), 322–345. https://doi.org/10.1007/s00145-010-9086-6

[22] Aria Shahverdi. 2015. *Lightweight Cryptography Meets Threshold Implementation: A Case Study for SIMON*. Master's Thesis. Worcester Polytechnic Institute.

[23] Aria Shahverdi, Mostafa Taha, and Thomas Eisenbarth. 2015. Silent Simon: A Threshold Implementation under 100 Slices. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, Washington, DC, USA, 1–6. https://doi.org/10.1109/HST.2015.7140227

[24] Kris Tiri and Ingrid Verbauwhede. 2004. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*. IEEE Comput. Soc, Paris, France, 246–251. https://doi.org/10/cmcqgc

[25] Rajesh Velegalati and Jens-Peter Kaps. 2012. Introducing FOBOS: Flexible Open-Source BOard for Side-Channel Analysis. In *Third International Workshop on Constructive Side-Channel Analysis and Secure Design, COSADE 2012*.

[26] Rajesh Velegalati and Jens-Peter Kaps. 2013. Towards a Flexible, Open-Source BOard for Side-Channel Analysis (FOBOS). In *Cryptographic Architectures Embedded in Reconfigurable Devices, CRYPTARCHI 2013*.

ACKNOWLEDGMENTS

The authors would like to express their appreciation to Farnoud Farahmand for his valuable assistance during this research. This work has been partially supported by the National Science Foundation under Grant No.: CNS-1801512 and by the US Department of Commerce (NIST) under Grant No.: 70NANB18H218.

A APPENDIX

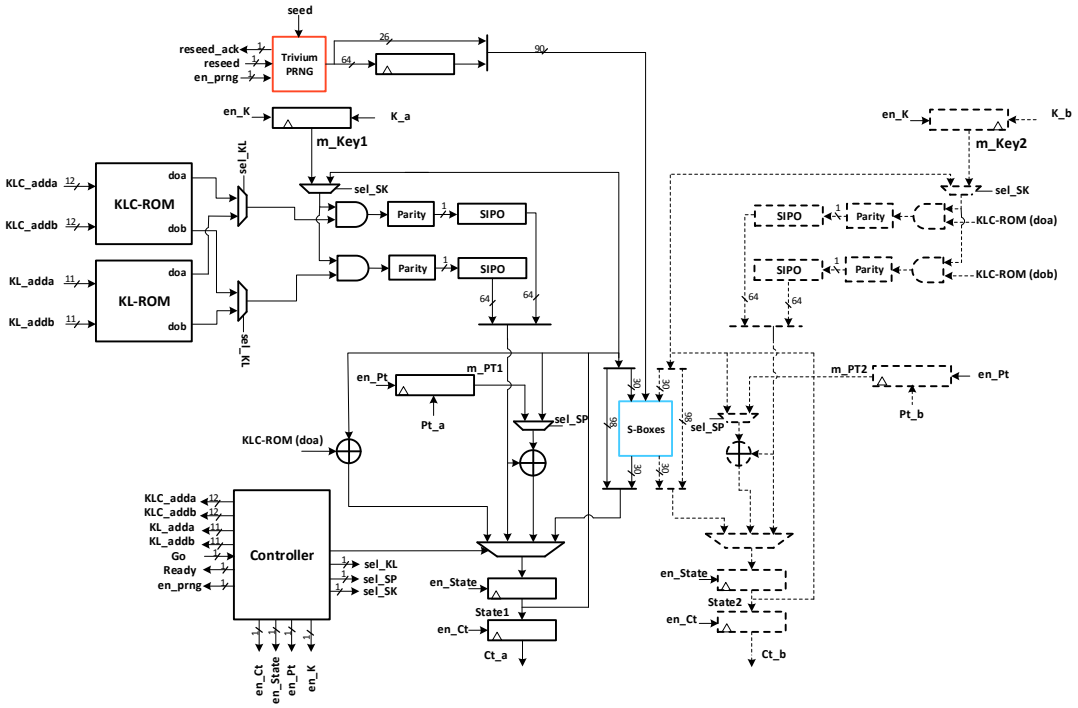

Figure 10: Top-level block diagram of the protected lightweight implementation. All bus widths are 128 bits, unless specified otherwise.

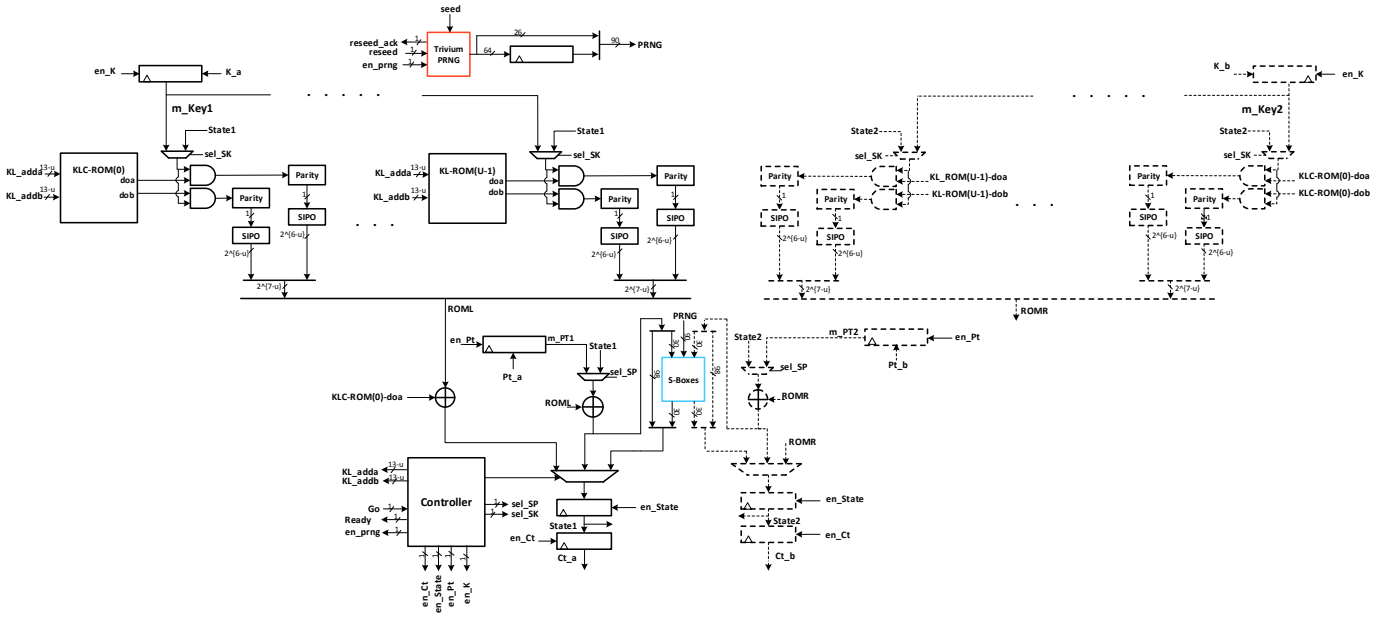

Figure 11: Generalized top-level block diagram of the protected unrolled implementation. All bus widths are 128 bits, unless specified otherwise.

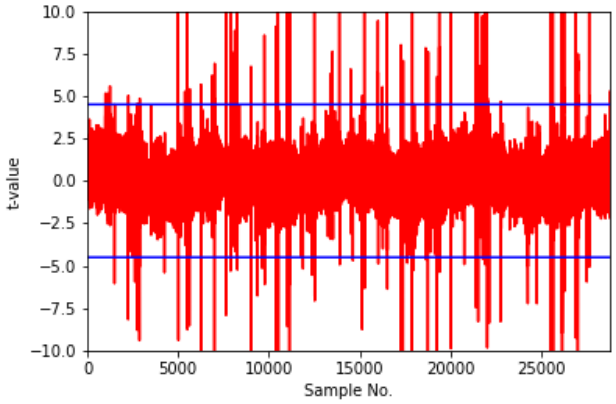

Figure 12: Fixed-versus-random TVLA of the unprotected unrolled implementation with $U = 16$ (10K traces)

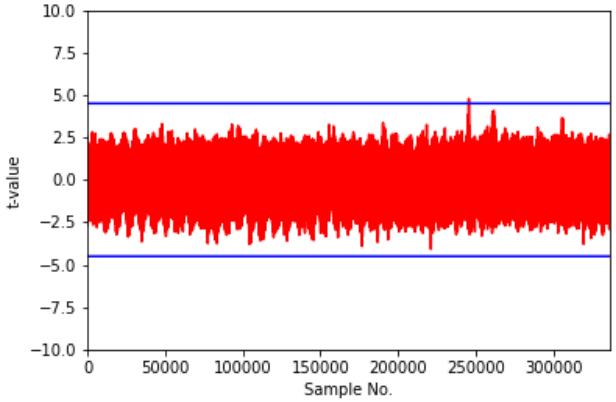

Figure 13: Fixed-versus-random TVLA of the unprotected lightweight implementation (10K traces)