Towards High-Level Synthesis of Polymorphic Side-Channel Countermeasures

Petr Socha, Martin Novotný
Czech Technical University in Prague
Faculty of Information Technology
Czech Republic
{petr.socha,novotnym}@fit.cvut.cz

***Abstract*—Side-channel attacks pose a severe threat to both software and hardware cryptographic implementations. Current literature presents various countermeasures against these kinds of attacks, based on approaches such as hiding or masking, implemented either in software, or on register-transfer or gate-level in hardware. However, emerging trends in hardware design lean towards a system-level approach, allowing for faster, less error-prone, design process, an efficient hardware/software co-design, or sophisticated validation, verification, and (co)simulation strategies. In this paper, we propose a Boolean masking scheme suitable for high-level synthesis. We implement a protected PRESENT encryption in C language, utilizing the concept of dynamic logic reconfiguration, synthesize it for Xilinx Artix 7 FPGA, and we compare our results regarding clock cycle latency and area utilization. We evaluate the effectiveness of proposed countermeasures using specific t-test leakage assessment methodology. We show that our high-level synthesis implementation successfully conceals the side-channel leakage while maintaining reasonable area and latency overhead.**

***Index Terms*—Cryptography, Side-Channel Analysis, Embedded Security, Internet of Things, High-Level Synthesis**

I. Introduction

With the upcoming Internet of Things era, we see a significant rise in embedded devices surrounding us in our everyday lives. To protect our privacy and to ensure the safety, many security measures must be taken, including the implementation of various authentication, authorization, and encryption schemes. Many of these algorithms, considered formally secure, are vulnerable to side-channel attacks when implemented improperly. These attacks, such as Differential Power Analysis, take advantage of data-dependent power consumption of the device [1], [2], or its electromagnetic radiation [3], and they typically aim at the secret key recovery. Side-channel attacks pose a severe threat to both software and hardware cryptographic implementations, especially in the IoT environment, where the attacker may easily gain physical access, leaving the device vulnerable to tampering.

Many countermeasures have been proposed to prevent side-channel attacks. These can be categorized into several classes based on the overall approach. Masking is a widely used technique based on a randomization of the processed data using random masks [4], [5], [6], [7], making it difficult for the attacker to predict any intermediate value and thus to exploit the leakage. Different approach is hiding, which aims at concealing the exploitable leakage in noise, e.g., by generating Gaussian noise, clock randomization [8], by using dual-rail logic [9], or by register/bus precharge with random data. Shuffling is another form of hiding, where, e.g., the cipher structure [10] or the control flow [11], [12] changes dynamically during the encryption. Some of the proposed countermeasures or their combinations can be implemented in modern hardware by utilizing dynamic (logic) reconfiguration [13], [14], [15].

Traditional and widely adopted hardware design methodology is based on the Register-Transfer Level (RTL) approach, i.e., modeling the digital system by registers, data signals, and logical operations between them. Such a system is usually described using concurrent languages like VHDL or Verilog. Emerging trends, however, lean to a system-level approach [16], [17], which brings many advancements, including more abstract design flow, optimal hardware/software co-design, simplified verification, validation and co-simulations, and many more. The system-level description may end up compiled into machine code, or synthesized into hardware implementation. High-level synthesis is a process of translating the C, C++ or SystemC algorithmic description to a clock-timed RTL or gate-level model, allowing further mapping onto final architecture/technology, e.g., an FPGA.

Our Contribution: In this paper, we propose a novel dynamic logic reconfiguration-based countermeasure approach to secure the PRESENT [18] encryption, described at the system level in C language, against side-channel attacks. We synthesize the implementation using High-Level Synthesis for Xilinx Artix 7 FPGA, and we evaluate the side-channel leakage using a specific leakage assessment methodology. We show that our protected high-level implementation successfully obscures the power consumption leakage, while managing to keep reasonable area and time overhead.

II. Preliminaries

In the following subsections, we briefly present the specifics and features of the high-level synthesis design flow. We then describe the PRESENT encryption algorithm based on a substitution-permutation network. Finally, we explain the dynamic logic reconfiguration-based approach to the Boolean masking.

A. FPGA Design using High-Level Synthesis

While RTL design is still a predominant digital design methodology, the rising complexity of nowadays digital systems calls for a more robust approach. With technologies such as Systems-on-chip (SoC) and Networks-on-chip (NoC), many new issues and challenges arise, regarding, e.g., optimal hardware/software co-design or validation/verification strategies. Also, there is a "design race" where designers and developers are pushed towards earlier deadlines and lower costs at the same time. System-level design, where hierarchical blocks are described on an algorithmic level, abstracted from the platform or technology specifics, seems like a promising emerging alternative to the current methodologies.

In this paper, we choose Xilinx Artix 7 FPGA as our target. Given this, we choose Xilinx Vivado Design Suite as our high-level synthesis toolchain [19], which provides synthesis from C, C++, or SystemC code to an IP core.

For the code to be synthesizable, only a subset of the C language must be used (similar to the VHDL language in case of the RTL design), with most of the basic constructs available, such as variables, arrays, loops, or conditions. Extra features like arbitrary-width integer types allow for better optimization. Furthermore, additional constraints and optimizations may be set using pragma directives, e.g., advising loops to be pipelined or unrolled, partitioning arrays, and more. Designers may also define the resulting I/O protocol, including various handshakes or bus interfaces like AXI. The final implementation is synthesized using a selected strategy based on the constraints and metrics such as area, throughput and latency, allowing to explore possible design space quickly. Figure 1 depicts the workflow using high-level synthesis.

There are limited research resources regarding usage of high-level synthesis in the cryptographic domain [16], [20], dealing mostly with time and area performance. The effects of various high-level synthesis parameters on the power side-channel are discussed in [17]. Techniques reducing side-channel vulnerabilities such as imbalanced code branches are presented in [21]. Information flow enforcement [22], [23] introduces prevention against threats such as exploitable I/O and buffer errors, unprivileged access, or timing attacks. To the best of our knowledge, there is no research regarding high-level cryptographic implementations with power side-channel countermeasures, preventing against attacks such as Differential Power Analysis, available in the present literature.

B. PRESENT Encryption

PRESENT by Bogdanov et al. [18] is a lightweight symmetric block cipher based on a substitution-permutation network (much like the more prolific Advanced Encryption Standard) with a block size of 64 bits and possible key sizes of 80 or 128 bits. The plaintext is encrypted by iteratively applying 31 transformations, called rounds. Each round consists of a round key addition (XOR), a non-linear substitution layer (4-bit S-boxes applied 16 times in parallel), and a linear permutation layer. After 31 rounds, the 32nd round key is finally added to produce the ciphertext. Figure 2 depicts the encryption algorithm. The versions with 80-bit and 128-bit keys differ only in the Key Schedule operation.

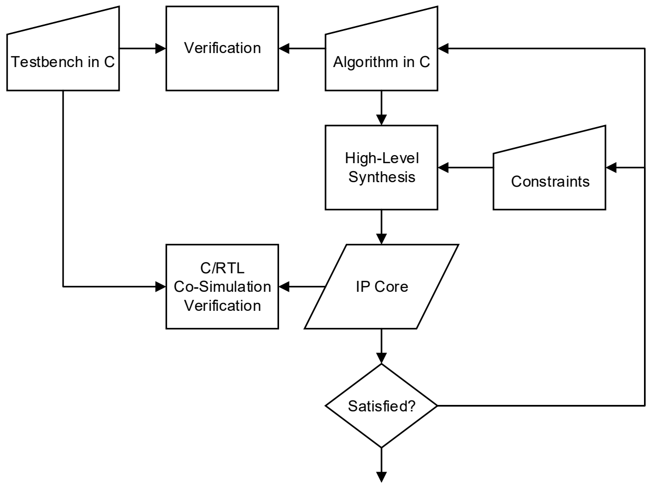


Figure 1: Example of a design flow using high-level synthesis.

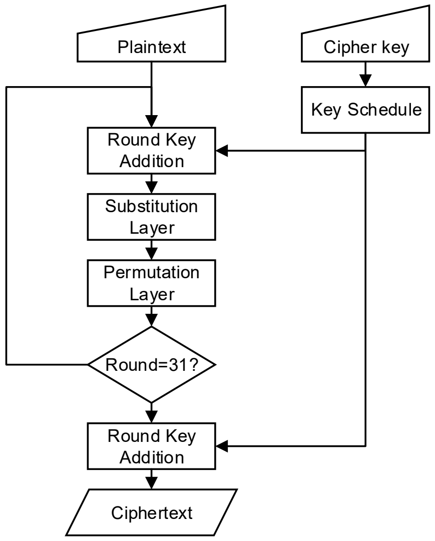


Figure 2: PRESENT encryption algorithm.

C. Boolean Masking

Boolean masking is a countermeasure against side-channel attacks proposed by Chari et al. [4]. Akkar et al. proposed a masking scheme for a substitution-permutation network (namely AES) in [5], with many others following [24], [25]. Mangard et al. [26] showed, however, that this concept of masking, in hardware implementations, is still vulnerable to first-order side-channel attacks because of glitches. This issue is solved by masking schemes such as Threshold Implementation proposed by Nikova et al. [6], or Domain Oriented Masking scheme proposed by Groß et al. [27].

Güneysu et al. proposed a scrambling of BRAM-based S-boxes in [8]. Similar dynamic logic reconfiguration-based concept, i.e., usage of precomputed masked S-boxes, which is not affected by glitches, was proposed by Sasdrich et al. [13] using more efficient CFGLUTs. In the following paragraphs, we briefly describe the masking scheme utilizing a reconfigurable substitution layer.

Assuming the PRESENT encryption algorithm accepts plaintext pt masked by XORing a random mask m:

$$state' := pt \oplus m, \tag{1}$$

where $state'$ is the masked cipher state, three round operations/layers need to be taken into account and altered appropriately so that equation

$$state = state' \oplus m \tag{2}$$

holds, allowing to finally obtain the ciphertext using $state'$.

The first layer, Round Key Addition, i.e., XOR, is a commutative and associative operation:

$$state' \oplus rk = (state \oplus rk) \oplus m. \tag{3}$$

Therefore, the addition of the round key rk does not require any further alteration since the output of the layer is already equal to the valid cipher state masked by m.

The last, the Permutation layer, is a linear transformation P, permuting bits of the cipher state. The output of the layer is therefore equal to the valid cipher state masked by a permuted mask:

$$P(state') = P(state) \oplus P(m), \tag{4}$$

which means the mask that would need to be subtracted to obtain the valid cipher state changes to $P(m)$ and Equation 2 would not hold anymore if no alterations were done.

The middle layer is a non-linear Substitution layer S, typically realized using a look-up table, in both hardware and software. One option to retain the validity of the output is to alter this look-up table into a masked substitution layer S':

$$S'(state') := S(state' \oplus m) \oplus P^{-1}(m), \tag{5}$$

which realizes the original substitution upon masked input value and outputs the substitution result masked by m processed with inverse permutation P^{-1}. This approach solves the mask alteration performed by the Permutation layer, since:

$$P(state \oplus P^{-1}(m)) = P(state) \oplus P(P^{-1}(m)), \tag{6}$$

making S' the only alteration that needs to be done for Equation 2 to hold. Using reconfigurable look-up tables (e.g., using arrays in software or BRAM, CFGLUT, or LUTRAM in hardware) for the S' layer, different masks may be used for every encryption [13], [15].

While the described Boolean masking scheme may often be sufficient for software implementations run on microcontrollers, where most leakage occurs on precharged memory buses, the situation is more complicated in CMOS registers, where the intermediate power consumption depends on bit transitions, i.e., Hamming distance between the old and the new value. Since

$$HD(x \oplus m, y \oplus m) = HW(x \oplus y \oplus m \oplus m) = HD(x, y), \tag{7}$$

where HW denotes Hamming weight and HD denotes Hamming distance, and assuming the algorithm operates one round per clock cycle, the masking described earlier protects only the first round of the encryption. Sasdrich et al. [13] solve this issue by applying Register Precharge, i.e., duplicating the cipher state register and interleaving the masked state by random data. This solves the problem, but it also reduces the throughput of the cipher to half.

Although the Boolean masking scheme may be straightforwardly described on the system level in C language, the Register Precharge is a technology-dependent countermeasure on the register-transfer level. This fact makes the masking scheme described earlier unsuitable for high-level synthesis, calling for a novel approach.

III. High-Level Synthesis of Alternating Masks Scheme

In the following subsections, we propose a masking scheme suitable for high-level synthesis, extending the one presented in subsection II-C. We present our implementation and compare the synthesis results with other approaches.

A. Alternating Masks Scheme

As explained in subsection II-C, the leakage on the working register is proportional to the Hamming distance between the old and a new value, which is, by definition, independent of any mask XORed to both values. To solve this issue, we propose using two independent random masks, m_1, m_2, and alternating these masks in consecutive rounds. This successfully masks the Hamming distance leakage, since

$$HD(x \oplus m_1, y \oplus m_2) = HW(x \oplus y \oplus m_1 \oplus m_2). \tag{8}$$

This approach is only sufficient if the implemented encryption algorithm operates at most one round per clock cycle, or, if partially unrolled, an odd number of rounds per clock cycle.

To implement this masking, we propose using two different altered substitution layers, S'_0 and S'_1, to be used alternatingly in odd rounds:

$$S'_1(state') := S(state' \oplus m_1) \oplus P^{-1}(m_2), \tag{9}$$

and even rounds:

$$S'_0(state') := S(state' \oplus m_2) \oplus P^{-1}(m_1). \tag{10}$$

Assuming the PRESENT encryption with 31 rounds, as described in subsection II-B, this results in one mask being used for masking the plaintext, with the output ciphertext being masked using the other mask.

Two different masks are generated for every encryption, and the two sets of masked 4-bit S-boxes (32 S-boxes in total) are implemented using dynamic logic reconfiguration.

This masking scheme can be easily described purely algorithmically using C language and, as shown further, is well suitable for FPGA high-level synthesis.

Listing 1: Top-level function.

```
present_block_t encrypt(present_block_t plaintext,
                        present_key_t key,
                        present_block_t maskIn,
                        present_block_t maskOut) {
  reconfigure(maskIn, maskOut);
  present_block_t state = plaintext;
  present_key_t roundKey = key;
  state = addRoundKey(state, roundKey);
  for(int round = 1; round < 32; round++) {
  #pragma HLS pipeline
    roundKey = updateKey(roundKey, round);
    state = sLayer(state, round);
    state = pLayer(state);
    state = addRoundKey(state, roundKey);
  }
  return state;
}
```

Listing 2: Substitution layer.

```
static present_sbox_t
              sBoxMasked[2][SBOX_COUNT][SBOX_RANGE];
present_block_t sLayer(present_block_t state,
                       present_round_idx_t round) {
  present_block_t newState;
  for(int i=0; i<16; i++){
  #pragma HLS unroll
    newState(i*4 + 3, i*4) =
        sBoxMasked[round%2][i][state(i*4 + 3, i*4)];
  }
  return newState;
}
```

Listing 3: Reconfiguration of S-boxes.

```
void reconfigure(present_block_t maskIn,
                 present_block_t maskOut) {
  #pragma HLS RESOURCE variable=sBoxMasked \
                       core=RAM_1P_LUTRAM
  #pragma HLS ARRAY_PARTITION variable=sBoxMasked \
                              complete dim=2
  present_block_t mask1[2]={maskOut, maskIn};
  present_block_t mask2InvP[2]={pInvLayer(maskIn),
                                pInvLayer(maskOut)};
  L1: for(int i = 0; i < 2; i++){
  #pragma HLS unroll
    L2: for(int j = 0; j < SBOX_RANGE; j++){
    #pragma HLS pipeline
      L3: for(int k = 0; k < SBOX_COUNT; k++){
      #pragma HLS unroll
        present_sbox_t idx =
            mask1[i](4*k+3, 4*k) ^ j;
        present_sbox_t val =
            sBoxClean[j] ^ mask2InvP[i](4*k+3, 4*k);
        sBoxMasked[i][k][idx] = val;
      }
    }
  }
}
```

B. Our Implementation

We implemented the masking scheme proposed in subsection III-A in C for Xilinx Vivado High-Level Synthesis. Listing 1 shows the top-level function, which determines I/O of the final IP core and defines the PRESENT encryption algorithm. First, the masked substitution layers S'_0 and S'_1 are computed using function $reconfigure()$. After that, 31 encryption rounds are performed, as described in subsection II-B. Notice the pragma *pipeline* directive, which in this case assures a single cycle iteration latency. All the functions, but $reconfigure()$, get inlined during the synthesis process.

The substitution layer is described in Listing 2. Notice the pragma *unroll* directive, which causes the loop iterations to be scheduled in parallel. Also, notice the bit-slicing indexing provided by Vivado's *ap_uint* type.

Listing 3 describes an area-optimized version of the dynamic logic reconfiguration of the substitution layers. Pragma $RESOURCE$ directive specifies a memory primitive used to implement a variable, in this case, LUTRAM, i.e., a single-port distributed RAM. When the memory primitive is not explicitly set, synthesis chooses one automatically to satisfy the elaborated read/write schedule, taking both latency and resource utilization in the account (this may, however, result in a nonoptimal solution, such as using 16 18K block RAMs).

Pragma $ARRAY\_PARTITION$ specifies partitioning of an array into smaller arrays, partially or completely, in the specified dimension. This results in using more instances of the underlying memory primitive, i.e., multiple smaller memories instead of one large memory, and it therefore also increases the number of R/W ports. In this case, the multi-dimensional $sBoxMasked$ array (declared in Listing 2) is partitioned completely in the second dimension, resulting in 16 memory instances (S-boxes) addressed using five bits ($round\%2$ and 4-bit input value). The 16 S-boxes, forming the substitution layer, are configured in parallel (loop $L3$), one input value at a time (loop $L2$). Even rounds and odd rounds substitution layers reconfigurations are constrained to be performed in parallel (loop $L1$); however, given the array partitioning and resulting access conflicts, they are scheduled sequentially by the synthesis tool. This version of dynamic logic reconfiguration is further referred to as Version 1.

In order to reduce the reconfiguration time complexity, the $sBoxMasked$ array may be further partitioned in the first dimension, resulting in 32 memory instances addressed using the 4-bit S-box input value. Swapping loops $L1$ and $L2$ now allows for parallel reconfiguration of both even rounds and odd rounds substitution layer S-boxes, resulting in lower time complexity at the expense of the area. This version of dynamic logic reconfiguration is further referred to as Version 2.

Unrolling all the three loops (and letting the synthesis resolve occurring read/write conflicts) results in even lower latency; this version is further referred to as Version 3.

C. Performance and Area Utilization

Table I compares area and latency of various PRESENT encryption FPGA implementations:

- Unprotected VHDL implementation for register-transfer level synthesis,

Table I: PRESENT encryption FPGA area and latency comparison.

Implementation	Area		Latency (clock cycles)		
	Memory (FFs)	Logic (LUTs)	Encryption	Extra	Total
Unprotected, register-transfer level synthesis, LUT-based S-boxes	150	150	31	0	31
Unprotected, high-level synthesis, LUT-based S-boxes	229	226	32	1	33
Alternating Masks Scheme, high-level synthesis, Version 1	392	420	32	38	70
Alternating Masks Scheme, high-level synthesis, Version 2	442	598	32	20	52
Alternating Masks Scheme, high-level synthesis, Version 3	439	771	32	17	49
S-box Decomposition + Masking + Register Precharge, register-transfer level synthesis, by Sasdrich et al. [13]	601	1508	62	32 (16)	94 (78)

- Unprotected C implementation for high-level synthesis,
- Three versions of the proposed Alternating Masks Scheme C implementation for high-level synthesis,
- Protected register-transfer level implementation by Sasdrich et al. [13], utilizing a similar concept of dynamic reconfiguration-based countermeasures.

The results for the first five implementations are Vivado's post-RTL synthesis utilization statistics. Results for the last implementation are based on those reported in [13]. All implementations use look-up table S-boxes. It is essential to say that the work by Sasdrich et al. combines three countermeasures: S-box Decomposition, Boolean Masking, and Register Precharge, while our proposed scheme may be considered equivalent to using only Boolean Masking + Register Precharge. Implementation by Sasdrich et al. targets Spartan 6 FPGA, which utilizes 6-input LUT primitives same as Artix 7. Sasdrich et al. propose to reduce the extra latency by precomputing the reconfiguration data during previous encryption, i.e., the latency in the brackets.

The single encryption latency of our implementation is almost half of that reported by Sasdrich et. al, since we do not use register precharge and we utilize two masks instead, making total latency of our implementations lower in all cases. Presented results demonstrate that a reasonable area overhead can be achieved when using the high-level synthesis. Furthermore, the system-level approach allows for quick exploration of the design space, evaluating the area and latency trade-off.

IV. Side-Channel Leakage Evaluation

In the following subsections, we explain the methodology used for side-channel leakage evaluation, and we compare the results for both unprotected and protected PRESENT encryption FPGA implementations.

A. Methodology

We evaluate side-channel leakage using specific t-test leakage assessment methodology [28], [29]. The protected high-level implementation, Version 1, is synthesized and implemented with Xilinx Vivado 2019.2 tools using the default synthesis strategy. The implementation runs on a dedicated side-channel experimental board [30], equipped with Xilinx Artix 7 FPGA, clocked by an external 100MHz crystal oscillator, which gets divided by an internal MMCM module down to 5MHz, used to clock the encryption IP core. The FPGA design receives plaintexts and random masks from an external source (the controlling PC) and sends out the ciphertext using the UART interface. The power consumption is measured using PicoScope 6404D oscilloscope and Langer EMV-Technik PA 303 preamplifier. Voltage over the FPGA core is sampled in the Vdd path, using 0.1Ω shunt resistor, with the preamplifier acting as a DC blocker, and with 20MHz bandwidth limiter enabled on the oscilloscope. Power traces are captured during every encryption, i.e., while the encryption IP core is active, capturing 8875 samples per trace with sample rate 625 MS/sec, i.e., 125 samples per clock cycle (the whole measured interval lasts $14.2\mu s$).

Two independent sets of power traces are measured:

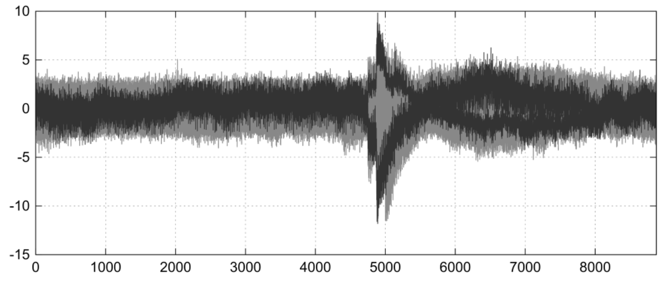
- unprotected encryption, i.e., with masks set to all zeroes,
- protected encryption, with uniform random masks.

In every data set, 1 million power traces are measured using uniform random plaintexts. Power traces in both data sets are processed independently in the following manner.

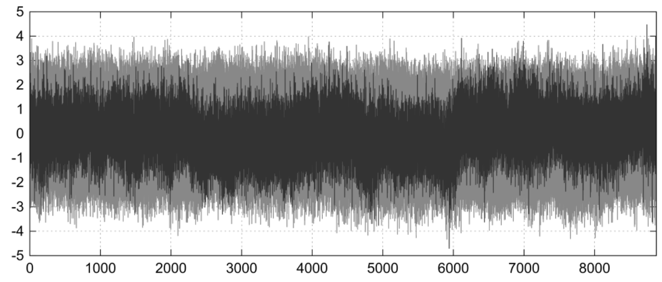
The data set is split into two disjoint groups according to a bit in a chosen intermediate cipher value. We choose the output of the substitution layer (S-boxes outputs), i.e., 64 different models, and XOR of consecutive rounds inputs (working register leakage), i.e., another 64 different models. For every model, Welch's t-test statistic is computed, at every sampling point independently:

$$t = \frac{\bar{X_1} - \bar{X_2}}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}}, \tag{11}$$

where $\bar{X_1}$, $\bar{X_2}$ are sample means, s_1^2, s_2^2 are sample standard deviations and N_1, N_2 are cardinalities of the first and the second group, respectively. The *null hypothesis* is that the two groups' means are equal, i.e., the groups are indistinguishable by their sample means. The hypothesis gets rejected for high values of $|t|$ according to Student's t-distribution and selected significance level. In side-channel leakage assessment, the value 4.5 or 5 is often considered a reasonable threshold for the $|t|$ value, to reject the hypothesis. This has to be done with the possibility of both false positives and false negatives in mind [31].

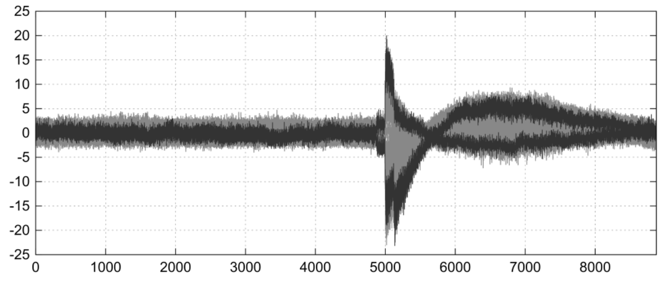


(a) Unprotected encryption.

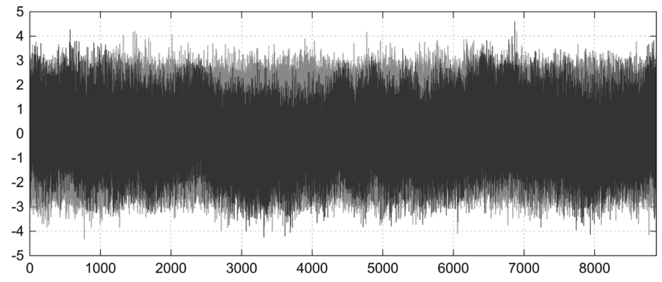


(b) Protected encryption.

Figure 3: Test results of 64 models based on substitution layer output (S-boxes outputs), where the t-value is shown on the vertical axis and the time samples during the encryption are shown on the horizontal axis.



(a) Unprotected encryption.



(b) Protected encryption.

Figure 4: Test results of 64 models based on XOR of consecutive rounds inputs (working register leakage), where the t-value is shown on the vertical axis and the time samples during the encryption are shown on the horizontal axis.

B. Results

As mentioned above, every t-test is based on one million power traces. All t-tests are measured using the Version 1 of the proposed implementation. Unprotected encryption is measured by setting both masks to all zeroes (so that $S = S'_0 = S'_1$). Figure 3 shows results of t-tests based on 64 models (64 overlaid curves), based on the substitution layer output, i.e., on S-boxes outputs, for both unprotected and protected PRESENT encryption. Figure 4 shows results based on XOR of consecutive rounds inputs, i.e., on working register Hamming distance leakage model. In each plot, two models reaching the highest and the lowest t-value are highlighted. As can be seen, the proposed masking scheme successfully conceals the first-order side-channel leakage.

V. Future Work

Given these results, more complex higher-order glitch-resistant masking schemes could be considered for a high-level synthesis. An algorithm for formal verification of higher-order masking schemes is presented in [32]. Side-channel analysis of a source code at compilation time was recently proposed in [33]. System-level FPGA design introduces many novel challenges, including automated side-channel leakage assessment and verification.

VI. Conclusion

In this paper, we proposed a Boolean masking scheme, utilizing dynamic logic reconfiguration, and suitable for high-level synthesis from the C language algorithmic description. The Alternating Masks Scheme was proposed as an alternative to the masking and register precharge combination to deal with Hamming distance leakage. In addition to a purely algorithmic description, the proposed scheme also allows for higher throughput compared to a combination of masking and register precharge.

We implemented PRESENT encryption with the proposed side-channel countermeasures in C language, and we synthesized the protected implementation for Xilinx Artix 7 FPGA. We have evaluated the area/latency trade-off, and we compared our results with unprotected implementations and with an existing state-of-the-art dynamic reconfiguration-based protected implementation. We showed that the overhead brought in by high-level synthesis is reasonable considering both area and latency.

To evaluate the effectiveness of the proposed side-channel countermeasure, we performed a specific t-test leakage assessment using one million power traces, focusing on the substitution layer output and the XOR of consecutive rounds inputs. Our results show that the proposed masking scheme successfully conceals the first-order side-channel leakage.

Acknowledgment

This work was partially supported by the Czech Technical University (CTU) grant No. SGS20/211/OHK3/3T/18.

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] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Annual International Cryptology Conference*. Springer, 1999, pp. 398–412.

[5] M.-L. Akkar and C. Giraud, "An implementation of des and aes, secure against some attacks," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2001, pp. 309–318.

[6] 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.

[7] E. Prouff and M. Rivain, "Masking against side-channel attacks: A formal security proof," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 142–159.

[8] T. Güneysu and A. Moradi, "Generic side-channel countermeasures for reconfigurable devices," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2011, pp. 33–48.

[9] T. Popp and S. Mangard, "Masked dual-rail pre-charge logic: Dpa-resistance without routing constraints," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2005, pp. 172–186.

[10] 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.

[11] B. Gierlichs, J.-M. Schmidt, and M. Tunstall, "Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output," in *International conference on cryptology and information security in Latin America*. Springer, 2012, pp. 305–321.

[12] S. Jeřábek and J. Schmidt, "Analyzing and optimizing the dummy rounds scheme," in *2019 IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2019, pp. 1–4.

[13] 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.

[14] N. Mentens, "Hiding side-channel leakage through hardware randomization: A comprehensive overview," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, 2017, pp. 269–272.

[15] P. Socha, J. Brejník, S. Jeřábek, M. Novotný, and N. Mentens, "Dynamic logic reconfiguration based side-channel protection of aes and serpent," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2019, pp. 277–282.

[16] E. Homsirikamol and K. Gaj, "Can high-level synthesis compete against a hand-written code in the cryptographic domain? a case study," in *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*. IEEE, 2014, pp. 1–8.

[17] L. Zhang, W. Hu, A. Ardeshiricham, Y. Tai, J. Blackstone, D. Mu, and R. Kastner, "Examining the consequences of high-level synthesis optimizations on power side-channel," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1167–1170.

[18] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2007, pp. 450–466.

[19] Xilinx, "Vivado design suite user guide: high-level synthesis (ug902)," 2018.

[20] A. Khalid, G. Paul, and A. Chattopadhyay, "High level synthesis for symmetric key cryptography," in *Domain Specific High-Level Synthesis for Cryptographic Workloads*. Springer, 2019, pp. 51–90.

[21] S. C. Konigsmark, D. Chen, and M. D. Wong, "High-level synthesis for side-channel defense," in *2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2017, pp. 37–44.

[22] Z. Jiang, S. Dai, G. E. Suh, and Z. Zhang, "High-level synthesis with timing-sensitive information flow enforcement," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[23] Z. Jiang, H. Jin, G. E. Suh, and Z. Zhang, "Designing secure cryptographic accelerators with information flow enforcement: A case study on aes," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[24] E. Trichina, T. Korkishko, and K. H. Lee, "Small size, low power, side channel-immune aes coprocessor: design and synthesis results," in *International Conference on Advanced Encryption Standard*. Springer, 2004, pp. 113–127.

[25] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen, "A side-channel analysis resistant description of the aes s-box," in *International Workshop on Fast Software Encryption*. Springer, 2005, pp. 413–423.

[26] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully attacking masked aes hardware implementations," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2005, pp. 157–171.

[27] H. Groß, S. Mangard, and T. Korak, "Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order," in *ACM Workshop on Theory of Implementation Security*, 2016.

[28] B. J. Gilbert Goodwill, J. Jaffe, P. Rohatgi *et al.*, "A testing methodology for side-channel resistance validation," in *NIST non-invasive attack testing workshop*, vol. 7, 2011, pp. 115–136.

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

[30] M. Bartík and J. Buček, "A low-cost multi-purpose experimental fpga board for cryptography applications," in *Advances in Information, Electronic and Electrical Engineering (AIEEE), 2016 IEEE 4th Workshop on*. IEEE, 2016, pp. 1–4.

[31] 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.

[32] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, and P.-Y. Strub, "Verified proofs of higher-order masking," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 457–485.

[33] N. Bruneau, C. Christen, J.-L. Danger, A. Facon, and S. Guilley, "Security evaluation against side-channel analysis at compilation time," in *International Conference on Algebra, Codes and Cryptology*. Springer, 2019, pp. 129–148.