Leveraging the Resources of Modern FPGAs in the Design of High-Performance Masked Architectures

Vincent Grosso[1] and Carlos Andres Lara-Nino[2]

1. UJM St-Etienne, CNRS, LabHC UMR 5516, SAINT-ETIENNE, France.
2. Universitat Rovira i Virgili, DEIM, TARRAGONE, Spain.

September 6, 2024

Abstract

Side channel attacks persist as the most relevant threat against cryptographic primitives like the advanced encryption standard (AES). Functions which "in paper" are considered secure can be compromised from their implementations. An adversary can study the information leakage found in the electromagnetic footprints of a device to gain insights regarding its operations. These leakages maintain relationships with the secrets held by the platform. In the AES case, the encryption keys. Masked implementations seek to disturb these relationships with an approach taken from multi-party computing. Sensitive operations are fragmented into multiple shares which are processed in a different and independent manner. This prevents the adversary from gaining a clear perspective of the process. Protected implementations tend to be costlier than their generic counterparts, thus they are a good fit for hardware acceleration. Unfortunately, the design of protected architectures implies a trade-off between latency, frequency, and the amount of resources. The latter is usually regarded as a fundamental characteristic, which results in an impact on the performance of masked architectures. Nonetheless, modern FPGAs have enough resources to enable high performance protected implementations. In this paper, we propose two of such architectures for AES. We discuss the potential performance advantages obtainable by leveraging the resources available in modern data-center acceleration FPGAs.

Keywords: AES, FPGA, Hardware Architecture, High Performance, Side-Channel Attacks, Threshold Implementation

Please cite as:

```
@InProceedings{GL24,
title = {{Leveraging the Resources of Modern FPGAs in the Design of High-Performance Masked
Architectures}},
author = {Grosso, Vincent and Lara-Nino, Carlos Andres},
booktitle = {XVIII Reuni{\'o}n espa{\~n}ola sobre criptolog{\'i}a y seguridad de la
informaci{\'o}n.  RECSI 2024},
pages = {1--6},
year = {2024},
publisher = {Ed.  Universidad de Le{\'o}n},
address = {Le{\'o}n, Spain}}
```

1 Introduction

Since their introduction in the late 90's, Side-Channel Attacks (SCA) [Koc96; KJJ99] have been a significant threat against cryptographic implementations. SCAs exploit physical characteristics (*e.g.* power consumption, execution time) from a device to recover sensitive information. These physical characteristics are dependent on the internal state of the device. At the same time, this internal state depends on data manipulated by the device and hence on the sensitive information. Therefore, there exists a link between the physical characteristics and the sensitive data. SCAs aim at exploiting this link, threatening the security of cryptographic implementations.

To prevent these vulnerabilities, various implementation techniques have been designed with the goal of decreasing the effectiveness of SCAs. One of the most popular countermeasures, masking, consists of implementing a secret-sharing approach and performing computations on shares, such that not all shares are used at the same instant of time. Masking was originally presented in [GP99]. Subsequently, different security models were developed. Perhaps the most used being the probing model [ISW03]. It introduces a security parameter—the implementation order—noted d. This is the smallest statistical moment that can be used in a successful attack (*i.e.* the smallest statistical moment dependent on the secret). Thus, to resist a d-th order attack, an implementation requires at least an $s = (d + 1)$ sharing of all sensitive data.

One of the greater problems in hardware masking is physical faults. Glitches have been exploited to break masked hardware implementations [MPG05; MPO05]. Indeed, glitches disrupt the independence of share's leakages which is required in the security reduction. Thus, they can reduce the effective order of the countermeasure. To limit the impact of glitches various solutions suggest utilizing registers to limit their propagation in $(d + 1)$ masking [Rep+15; GMK16; Fau+18; CS21; KM22]. However, the use of one, two or three layers of registers per operation quickly increases the latency and the area of an implementation. The number of register layers is dependent on the construction and whether the inputs are independent or not.

Recently, technological solutions have been explored to reduce the latency of masked implementations. These approaches replace the registers with locally-asynchronous globally-synchronous (LAGS) techniques [Sas+20; Sim+22; Nag+22]. Using locally-asynchronous techniques to synchronize signals and remove glitches allows to decrease the latency, but this impacts the frequency. This penalty is due to the requirement of additional combinatorial hardware layers in the circuit, which increases the propagation delay of its critical path (τ).

Another approach to resist glitches is based on multi-party computation over silicon and the so-called threshold implementations (TI) [NRR06; NRS08]. Such designs were initially presented as efficient and glitch-resistant first-order masked implementations. Efficiency being achieved thanks to the capability to compute function sharings. The threshold implementations require in general more shares than $(d+1)$-masking, as the algebraic degree of the function t intervenes in the number of shares. For first-order, it is either $t + 1$ or $t + 2$ [Pic+23] in comparison to 2 for $(d + 1)$-masking. In contrast, such functions can be computed in one cycle in comparison to at least $\mathcal{O}(log_2(t))$ cycles. Furthermore, it has been shown that TIs can achieve good trade-offs in various settings [Bos+17; Bil+13; Pos+11].

Nonetheless, the application of TIs to resist higher-order and multivariate attacks is not trivial. Indeed, the property of *uniformity* introduced by [NRR06] is not sufficient to guarantee security against multivariate attacks [Rep15]. To solve this problem, it is necessary to introduce a refresh at the output of a TI gadget [Rep+15]. According to [Bar+16], optimal refreshing functions can be created with $\frac{s(s-1)}{2}$ random bits. However, for $d \ll s$ it is possible to reduce the random data volume. Nonetheless, refreshing with less randomness than recommended by [Bar+16] has demonstrated some weakness [Moo+19]. Synchronization is also needed after each refresh operation. Thus, the number of shares as output of a gadget has a direct impact on the randomness requirement and the size of the architecture. We refer to this approach as $(td+1)$-masking.

Conventionally, masking schemes have sought to favor the implementation size. As argued earlier, this comes at the detriment of the latency or the frequency. This approach is consistent with the notion that maintaining a low-area profile can benefit a larger number of applications. Nonetheless, there is a growing market for high-performance cryptography in high-throughput applications. Streaming services, video games, online commerce, enhanced communications, and distributed computing systems are some of such niches. In these scenarios, the security of the data must be assured against all sort of adversaries. Yet, the performance-penalties associated with cryptography must be kept at a minimum to maintain the quality of service. Coincidentally, such applications tend to benefit from cloud services in their operation. Nowadays, enterprises like Amazon Web Services, Microsoft Azure and Alibaba Cloud allow their clients to dispose of hardware resources on-demand. In some cases this includes FPGA boards which may be physically linked to host of the virtual machine. If the application leverages these computing platforms they can dispose of a large cache of resources which can be destined to the implementation of cryptographic operations. At the same time, if the computing system is not hosted in-house, it makes sense to implement side-channel protections on such cryptographic architectures.

The FPGA devices available in the cloud are large. Amazon and Alibaba offer AMD-Xilinx Virtex UltraScale+ VU9P FPGAs through their *EC2 F1* and *Cloud F3* instances, respectively. Microsoft Azure offers AMD-Xilinx Alveo U250 FPGAs. The VU9P FPGAs feature 1.18M 6-input look-up tables (LUT), 2.36M d-type registers (FF) and 2160 block-RAM memories (BRAM). The U250 FPGAs feature 1.73M LUTs, 3.46M FFs, and 2688 BRAMs. For reference, high-performance implementations of AES employ some thousands LUTs, FFs, and tens of BRAMS [SS15]. This shows that there is surplus of resources which can be leveraged if the sole purpose of the FPGA is cryptography. This gives viability of TI approaches which haven't been previously studied.

In this work, we explore the efficiency of $(td+1)$-masking for obtaining a low-latency and high-frequency, higher-order protected architecture. This approach requires a large area to be implemented, but it's interesting for low-latency and high frequency use cases. We show that this strategy can be employed to create masked implementations of AES suitable for the FPGA devices available in the cloud.

2 Hardware masking

Threshold implementations were introduced in [NRR06] as an approach for first-order masking in the presence of glitches. The idea behind TIs is to implement a *sharing* of the sensitive data. The sensitive data x is represented by the sharing (x_0, x_1, x_2), such that $x = x_0 \oplus x_1 \oplus x_2$. The sharing should ensure that $\Pr[X = x|X_0 = x_0 \wedge X_1 = x_1] = \Pr[X = x|X_0 = x_0 \wedge X_2 = x_2] = \Pr[X = x|X_1 = x_1 \wedge X_2 = x_2] = \Pr[X = x]$. Then, the shares are processed in such a way that any shared function is independent of at least one share of each input variable. The function f is decomposed in different shared functions (f_0, f_1, f_2), such that $f(x) = f_0(x_1, x_2) \oplus f_1(x_0, x_2) \oplus f_2(x_0, x_1)$. This property is known as *non-completeness*, and it is important to resist glitches.

In [Pet19], Petrides studied the non-completeness in terms of a constrained set covering. He first introduced a formalism on the desirable property of the set and then derived constructions for non-completeness slicing. He also derived bounds on the number of output shares as function of three parameters: the numbers of input shares s, the algebraic degree of the function t, and the security order d. Non-complete covering sets can be used to build the $s \geq (td+1)$ masked gadgets.

In this work, we employ non-complete covering sets to implement a low-latency, high-frequency second-order masked implementation of AES. In particular, we focus on the $t = 3, d = 2$ case. Sharing for degree three is of paramount importance. For 4-bit permutation SBOXES, they allow to perform the sharing in one cycle. For 8-bit SBOXES, they can be used to reach the maximum degree of seven in two cycles. Therefore, in the case of AES the SBOX we set an optimal latency of two cycles. As reference, for $d = 2$, works in the literature have proposed latencies of six cycles [De +16].

Previous work [AZN21] suggested to decompose the inversion (x^{254}) in AES into two cubic functions $\left(x^{26}\right)^{49}$. The authors presented solutions for the first order but did not report any results on the higher order. It was mentioned that the decomposition could be used for higher-order implementations, however.

Petrides in [Pet19] presented several decompositions:

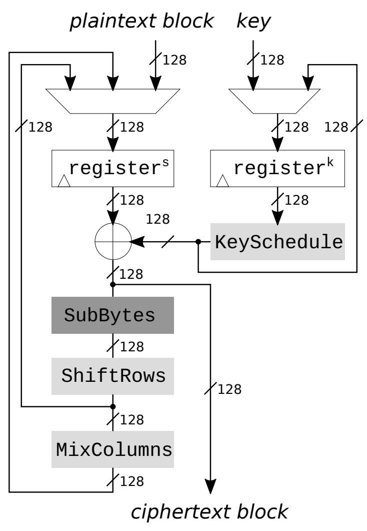
- $s = 7, t = 3, d = 2$ with 35 output shares.
- $s = 8, t = 3, d = 2$ with 18 output shares.
- $s = 9, t = 3, d = 2$ with 13 output shares.

With 7 input shares, the optimal case for $(td + 1)$, 35 output shares are produced. This restricts its practical usability. Thus, we sought to lower the number of output shares by manipulating the number of input shares:

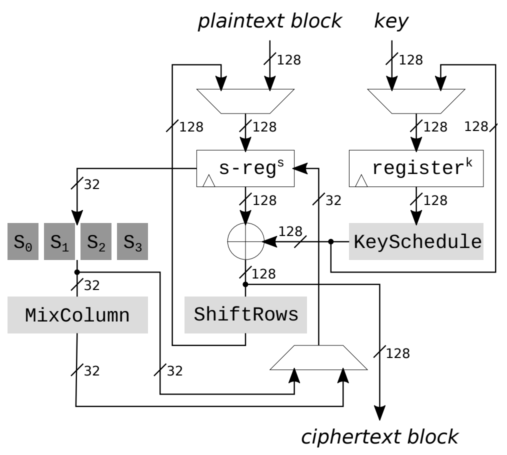
$s = 8$ the optimal number of output shares is 17, which is still large.

$s = 9$ the lower bound is 10, but we could only find a solution with 12 output shares. This produces an asymmetry which leads to an irregular datapath.

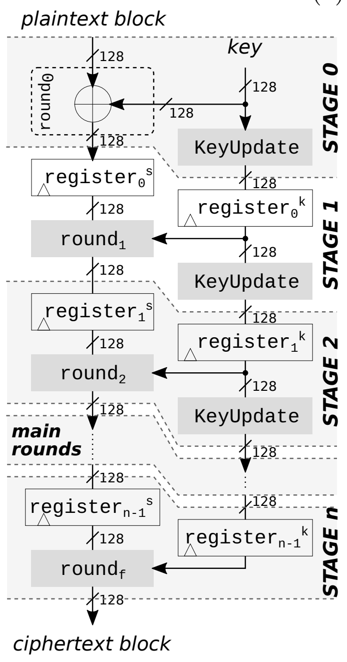
$s = 10$ we managed to find a solution which also has 10 output shares. This provides several advantages for maintaining a unified datapath. Thus, we selected $s = 10$.



(a) iterative



(b) serialized



(c) pipelined

Figure 1: Different architectures implementing the encryption function of AES128. The iterative design features a 3-MUX to account for the final round. In the serialized architecture the state register has been swapped by a shift-register. This design has a full-width loop to perform the key addition and state permutation in a single cycle; the substitution and non-linear layers are reduced to one quarter of their regular size, requiring four cycles to process the state. Note that this implies a swap in the substitution and permutation layers. The pipelined design is roughly n-times larger than the iterative design.

There are various ways to construct the component function from the non-complete set. In our case, we sought a formula to represent the SBOX. We selected a boolean formula to avoid complex field multiplications. We reduced the number of formula candidates to the one that contained at most monomials of degree t, in order to use the selected non complete set. Hence, we chose the Algebraic Normal Form (ANF) representation of the function. From the ANF, we built the component functions. From each monomial in the ANF, we calculated the shared version using the number of shares required. Then, we put each part of the shared monomial in a component function following to the chosen non-complete covering set.

3 Architecture design

AES is a well know algorithm used in a large set of applications. There are many works implementing this cipher in software [BS08; Osv+10] and hardware [Pra+05; GC09]. These proposals exhibit tradeoffs between performance and area, trying to improve the efficiency according to some given metric. Our work focuses on hardware implementations, however.

Multiple protected architectures of AES have also been proposed. There are those which maintain reasonable area and frequency profiles at the cost of larger latencies [Rep+15; GMK16; Fau+18; CS21]. As well as those which reduce the latency at the determent the critical path [Sas+20; Sim+22; Nag+22]. In contrast, we intend to propose low-latency and high-frequency architectures by removing the constraints in the implementation size. This is practical for certain applications with access to FPGA acceleration. Modern data-center acceleration devices are large enough to explore the performance boundaries in hardware masking.

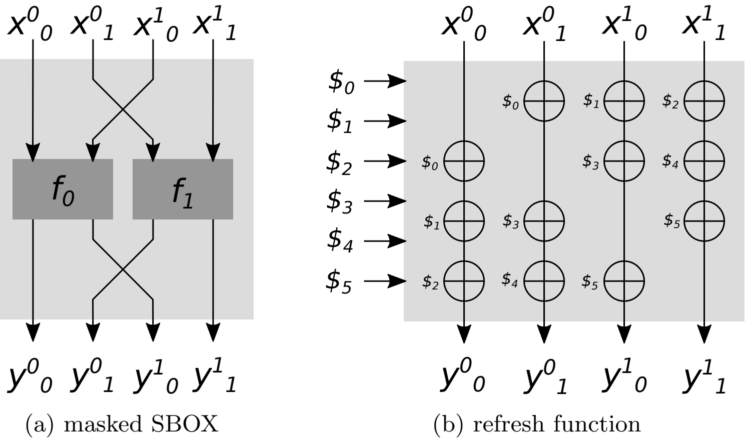


Figure 2: Key components of a threshold architecture. In this example, let $s = 4$ and suppose a 2-bit SBOX. Each input bit (x^0, x^1) is represented with two shares. Internally, the masked SBOX employs two linear functions (f_0, f_1) to produce the output shares such that $y^0 = y^0_0 \oplus y^0_1$ and $y^1 = y^1_0 \oplus y^1_1$.

Regardless of the implementation approach, the basic structure of the cipher is consistent. Substitutions (`SubBytes`), permutations (`ShiftRows`), and a non-linear layer (`MixColumns`) are intertwined with a key schedule to produce the ciphertext. An iterative architecture, as shown in Fig. 1a, will closely match the algorithmic specification by taking advantage of the dynamic register size of hardware designs. Thus, the latency will be equivalent to the number of rounds in the specification. These compositions are well rounded but mainly serve academic purposes. In the cases where the goal is to reduce the area of the circuit, a serialized architecture might prove to be more interesting. Such an design is illustrated in Fig. 1b. This strategy allows to reduce the datapath by processing the data into multiple slices. The latency of the implementation will be multiplied by the factor of the number of slices, however. On the other hand, if what is sought is to increment the performance, then it is possible to unroll the algorithm and then process multiple data by pipelining the resulting datapath. The initial latency of the pipeline will be equivalent to the number of stages. But as the data amount being processed increases the throughput will rise. The key of a successful pipelined design is to balance the distance between registers to obtain a consistent critical path. See Fig. 1c for an example.

3.1 Particularities of masked architectures

Masked architectures differ from ordinary hardware designs in two key ways. First, the SBOXES are replaced by masked gadgets. As illustrated in Fig. 2a, these circuits take s_i shares as inputs and produce s_o shares as outputs. Ideally, we seek that the number of inputs remains consistent with the number of outputs ($s_i = s_o$). However, as discussed in the previous section, this is not trivial with the chosen masking approach. The second constraint of $(td+1)$-masking is that the output of a gadget must be refreshed and subsequently a synchronization layer (registers) is expected. From Fig. 2b, the refresh function is simply a structured XOR of the shares with random input data. These random data must be produced from a cryptographic source like a TRNG. The amount of random data required depends greatly on the number of shares. A conservative refresh function requires $\frac{s(s-1)}{2}$ random bits, which is secure up to $d = s - 1$. However, for the cases where $d \ll s$ more efficient approaches can be used to reduce the number of random bits per refresh function.

Outside of the substitution layer, the other components of a threshold architecture are straightforward. For simplicity we consider that the plaintext and encryption keys exist in a "masked" state. That is, they have been expanded into shares and randomized through the application of the refresh function. Registers are increased to account for the expanded nature of the data. Permutations are applied at the level of nibbles. Other arithmetic operations are applied to a single share and at the end the result will be equivalent. The output is also provided as an expanded ciphertext. To retrieve its true value it suffices to collapse all the shares of a bit with the use of an XOR tree.

3.2 Iterative threshold architecture

As discussed in Section 2, with $t = 3$ only one cycle is required to implement a 4-input permutation SBOX. This is ideal for lightweight ciphers which use small SBOXES like Present and GIFT. For other algorithms, like AES, we can rely on the result by [AZN21] to decompose the inversion of AES into two cubic functions. In this way, an AES SBOX can be implemented through two gadgets of $t = 3, d = 2$. In our case we employ $s = 10$ which allows to maintain a consistent datapath width. Evidently, the use of two gadgets has the disadvantage that an intermediate refresh layer is required. But we can benefit from this refresh to introduce an intermediary register to reduce the critical path of the design. However, this has the impact of doubling the encryption latency. In Fig. 3 we present an iterative threshold architecture for AES.

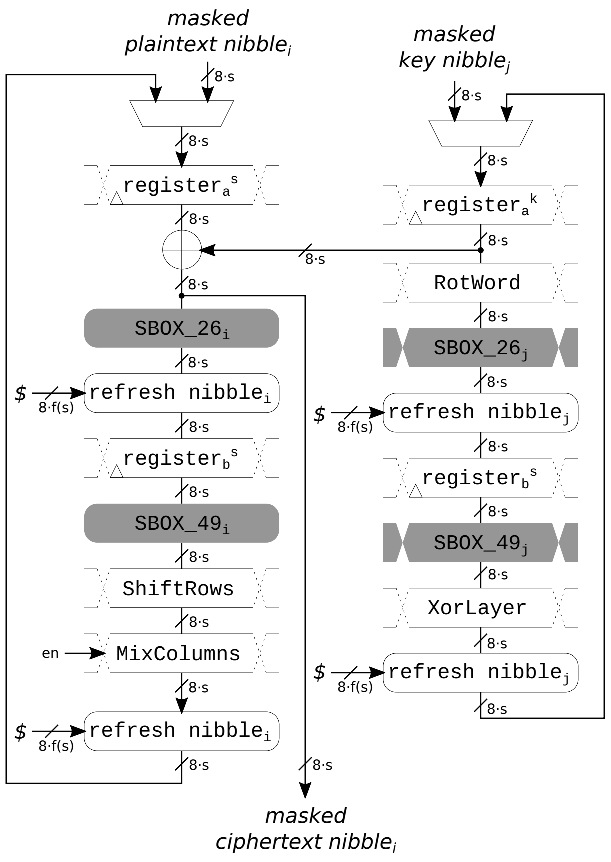


Figure 3: An iterative threshold architecture for AES; we depict the datapath of a single byte. In this case, an enable input for the `MixColumns` operation allows to reduce the size of the input MUX.

This architecture features four main registers of 128×10 bits each. The plaintext and keys are assumed to be masked before being loaded. The substitution box is implemented as two gadgets `SBOX_26` and `SBOX_49` which are used consecutively in the round function (16×2 gadgets) as well as in the key schedule (4×2 gadgets). For $s = 10, d = 2$ which satisfies $d \ll s$ we propose an efficient refresh function which requires only $f(s) = 12$ random bits. In total $(16 \times 2 + 4 \times 2) \times 10$ rounds $\times 12$ bits $= 600$ bytes of random data are required per block encryption. Please note that these data need not to be stored or transmitted as its stochastic nature does not affect the production of a deterministic ciphertext. The `AddRoundKey`, `ShiftRows`, and `MixColumns` operations are applied at the level of nibbles, rather than bits. With $s = 10$ their size ought to be multiplied by the same factor. For two of them, (`AddRoundKey` and `ShiftRows`), their hardware cost is so low that regardless of the factor the impact is negligible. Regarding `MixColumns`, its hardware cost is still small with regards to the cost of the gadgets.

A regular iterative architecture has a latency which is equivalent to the number of rounds of the cipher. However, due to the proposed SBOX decomposition, the intermediary register increases the latency of each loop by one cycle. Thus, for AES-128 this architecture has a latency of 20 cycles. This means that the throughput is reduced by half. Nonetheless, by implementing the appropriate control sequence it is possible to process two plaintext blocks consecutively. This 2-stage pipeline can mitigate the performance loss with an impact in the complexity of the design.

3.3 Pipelined threshold architecture

By taking advantage of the hardware resources available in data-center acceleration FPGAs we can explore the costs of a proper pipelined threshold architecture. One goal of such designs is to process one data block per cycle. Even if the initial latency is long, when large volumes of data are processed the throughput rises towards to the upper bound.

In the case of AES, we can select the initial round where the encryption key is added with the plaintext, the intermediary rounds, and the final round (without the `MixColumns` operation) as candidates for conforming the stages of the pipeline. However, with the proposed masking approach all the rounds after the initial step have an intermediary register. But with only two stages per round the critical path would be unbalanced. One of the stages would include a gadget and the other the second gadget plus the round logic. Another goal of a pipelined design is to maintain a balanced distribution of the hardware components. The maximum frequency is bound by the longest path between registers. Thus, we propose to create three stages for each one of the intermediary rounds. With an initial stage and two stages in the final round, this yields a 30-stage pipelined architecture. The key schedule is also partitioned in an equivalent number of stages to maintain coherency. This is illustrated in Fig. 4.

In regards to registers, an estimate of $(128 \times 2) \times 10 \times 30 = 76.8K$ FFs are invested in the pipeline. This is feasible when we consider the resource availability of the target FPGAs (a few millions in each case). The impact of unrolling the iterative threshold architecture from Fig. 3 is roughly equivalent to multiplying its hardware costs by ten. This means that a total of $(16 \times 2 + 4 \times 2) \times 10$ masked gadgets are required. This is the main component of the hardware cost.

4 Impact evaluation

In Table 1 we provide implementation results for the masked gadgets required to create the AES SBOX. We employed the AMD-Xilinx Vivado 2020.2 toolchain and targeted the Alveo U250 Data Center Accelerator Card available in the Microsoft Azure cloud instances.

From the results in Table 1, we can estimate the hardware costs for the iterative and pipelined threshold architectures. Recall that the gadget costs far

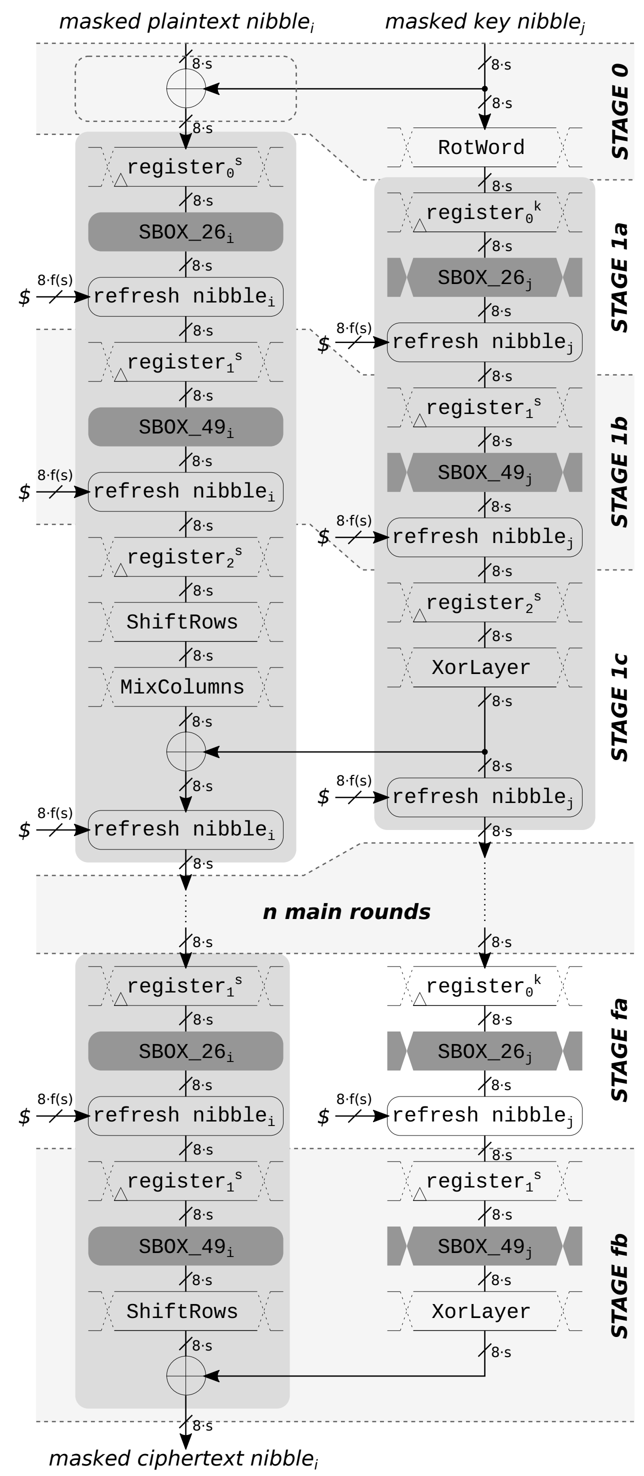


Figure 4: A pipelined architecture for AES

Table 1: Implementation results for the masked gadgets in the Alveo U250 Data Center Accelerator Card

Circuit	LUTs	τ (ns)	f_{max} (MHz)
`SBOX_26`	17,070	3.270	305
`SBOX_49`	18,747	2.941	340

out-weights the other parts of the circuit. These are summarized in Table 2.

Table 2: Hardware estimation for the proposed architectures

Arch.	`SBOX_26`	`SBOX_49`	LUTs	FFs
Iterative	20	20	$\sim$716K	$\sim$5K
Pipelined	200	200	$\sim$7.16M	$\sim$77K

The figures provided in Table 2 show that the iterative threshold architecture could be implemented in any of the FPGAs available from popular cloud providers. Just taking into account the raw cost of LUTs and FFs. In the practice, the LUT utilization could be reduced by leveraging other circuits available in the FPGA. For example DSPs and BRAMs. On the other hand, the pipelined architecture is roughly ten times larger. It exceeds the number of LUTs of any cloud FPGA. In order to implement this design we would need to employ one of the largest devices from AMD-Xilinx. The Versal Adaptive SoCs are the latest generation released by this manufacturer. Their premium series includes the VP1902, a SoC-FPGA with a reconfigurable fabric featuring 8.46M LUTs and 18.5M FFs. Currently it might seem impractical to consider to employ such a device. However, historical trends show that supply follows demand. If applications can benefit from these large FPGAs they will eventually become available on-demand.

5 Discussion

The protected architectures proposed in this work aim at improving the encryption throughput for applications which require to process large volumes of data. The chosen masked approach allows to obtain fast circuits. With a maximum frequency around 300 MHz per gadget, it is possible to match the operation rates of gigabit transceivers and DDR memories. This can be achieved by using the gadgets in performance-oriented architectures, for example the proposed pipelined design. This circuit has an initial latency of 30 cycles, but once the pipeline is full it is possible to process one plaintext block per cycle. The pipeline has also been carefully designed to maintain a good balance between the multiple stages. In particular, we note that in the gadget stages they are only combined with the refresh functions. Which are just a layer of XOR gates. Therefore, the critical path of the pipeline ought to be close to the critical path of the gadgets.

Works in the literature tend to favor the implementation size at the determent of the latency or the frequency. This evidently has an impact on the performance of the masked architectures. In [KM22] and [Cas+21] the authors propose implementations of the AES SBOX for $d = 3$ with latencies of four and eight cycles, respectively. For $d = 2$, [De +16] provides an implementation of the AES SBOX with a latency of six cycles. In [Mor+11] and [Bil+14; Bil+15] the authors provide similar implementations for $d = 1$ with latencies of five and three cycles per SBOX, respectively. Recall that our proposal requires only two cycles per SBOX. The latency of a full $d = 2$ AES encryption is reported at $\sim$200 cycles in [GMK16], whereas the SBOX proposed in [Mor+11; Bil+14; Bil+15] for $d = 1$ obtains latencies of 246 cycles per encryption. Our pipelined architecture can process one block per cycle.

In [Sim+22] the authors propose a 1-cycle implementation of the AES SBOX for $d = 1$ and $d = 2$, however this leads to an ASIC implementation (40nm) with a maximum frequency of 100 MHz. Other such designs for $d = 1$ and $d = 2$ are presented in [Nag+22] with a latency of 1 cycle per round and maximum frequencies of 192/169 MHz in silicon implementations (65nm). One of the fastest of such implementations is reported in [Sas+20] for $d = 1$, yet their 28nm implementation achieves only 400 MHz. In contrast, our designs can reach up to $\sim$300 MHz in FPGA. Bear in mind that ASICs tend to be much faster than FGPA prototypes.

6 Conclusions

In this paper, we have proposed two second-order high-performance masked architectures for AES. Our work aims at closing up a gap in the literature by exploring the boundary of resource utilization. We target large FPGAs which allow us to leverage their resource availability to propose iterative and pipelined protected designs. As far as we know, this is the first work of this genera in the literature.

Acknowledgments

C.A. Lara-Nino acknowledges the support of the Spanish *Instituto Nacional de Ciberseguridad (INCIBE)* through project HERMES; the Spanish government through project ACITHEC (PID2021-124928NB-I00); and the Catalonian *Agència de Gestió d'Ajuts Universitaris i de Recerca* (AGAUR) through grant 2021 SGR 00115.

References

[AZN21] Victor Arribas, Zhenda Zhang and Svetla Nikova. "LLTI: Low-Latency Threshold Implementations". In: *IEEE Trans. Inf. Forensics Secur.* 16 (2021), pp. 5108–5123. DOI: `10.1109/TIFS.2021.3123527`.

[Bar+16] Gilles Barthe et al. "Strong Non-Interference and Type-Directed Higher-Order Masking". In: *Proceedings of the 2016 ACM/SIGSAC Conference on Computer and Communications (CCS)*. ACM, 2016, pp. 116–129. DOI: `10.1145/2976749.2978427`.

[Bil+13] Begül Bilgin, Andrey Bogdanov, Miroslav Knezevic, Florian Mendel and Qingju Wang. "Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware". In: *Proceedings of the 2013 International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Springer, 2013, pp. 142–158. DOI: 10.1007/978-3-642-40349-1_9.

[Bil+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov and Vincent Rijmen. "A More Efficient AES Threshold Implementation". In: *Proceedings of the 2014 International Conference on Cryptology in Africa (AFRICACRYPT)*. Springer, 2014, pp. 267–284. DOI: 10.1007/978-3-319-06734-6_17.

[Bil+15] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov and Vincent Rijmen. "Trade-Offs for Threshold Implementations Illustrated on AES". In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 34.7 (2015), pp. 1188–1200. DOI: 10.1109/TCAD.2015.2419623.

[Bos+17] Erik Boss et al. "Strong 8-bit Sboxes with efficient masking in hardware extended version". In: *J. Cryptogr. Eng.* 7.2 (2017), pp. 149–165. DOI: 10.1007/s13389-017-0156-7.

[BS08] Daniel J. Bernstein and Peter Schwabe. "New AES Software Speed Records". In: *Proceedings of the 2008 International Conference on Cryptology in India (INDOCRYPT)*. Springer, 2008, pp. 322–336. DOI: 10.1007/978-3-540-89754-5_25.

[Cas+21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi and François-Xavier Standaert. "Hardware Private Circuits: From Trivial Composition to Full Verification". In: *IEEE Trans. Comput.* 70.10 (2021), pp. 1677–1690. DOI: 10.1109/TC.2020.3022979.

[CS21] Gaëtan Cassiers and François-Xavier Standaert. "Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.2 (2021), pp. 136–158. DOI: 10.46586/tches.v2021.i2.136-158.

[De +16] Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov and Svetla Nikova. "Higher-Order Threshold Implementation of the AES S-Box". In: *Proceedings of the 2015 International Conference on Smart Card Research and Advanced Applications (CARDIS)*. Springer, 2016, pp. 259–272. DOI: 10.1007/978-3-319-31271-2_16.

[Fau+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga and François-Xavier Standaert. "Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.3 (2018), pp. 89–120. DOI: 10.13154/tches.v2018.i3.89-120.

[GC09] Kris Gaj and Pawel Chodowiec. "FPGA and ASIC Implementations of AES". In: *Cryptographic Engineering*. Springer, 2009, pp. 235–294. DOI: 10.1007/978-0-387-71817-0_10.

[GMK16] Hannes Groß, Stefan Mangard and Thomas Korak. "Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order". In: *Proceedings of the 2016 Workshop on Theory of Implementation Security (TIS)*. ACM, 2016, p. 3. DOI: 10.1145/2996366.2996426.

[GP99] Louis Goubin and Jacques Patarin. "DES and Differential Power Analysis (The "Duplication" Method)". In: *Proceedings of the 2009 International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Springer, 1999, pp. 158–172. DOI: 10.1007/3-540-48059-5_15.

[ISW03] Yuval Ishai, Amit Sahai and David A. Wagner. "Private Circuits: Securing Hardware against Probing Attacks". In: *Proceedings of the 2003 Annual International Cryptology Conference (CRYPTO)*. Springer, 2003, pp. 463–481. DOI: 10.1007/978-3-540-45146-4_27.

[KJJ99] Paul C. Kocher, Joshua Jaffe and Benjamin Jun. "Differential Power Analysis". In: *Proceedings of the 1999 Annual International Cryptology Conference (CRYPTO)*. Springer, 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25.

[KM22] David Knichel and Amir Moradi. "Low-Latency Hardware Private Circuits". In: *Proceedings of the 2022 ACM/SIGSAC Conference on Computer and Communications (CCS)*. Springer, 2022, pp. 1799–1812. DOI: 10.1145/3548606.3559362.

[Koc96] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: *Proceedings of the 1996 Annual International Cryptology Conference (CRYPTO)*. Springer, 1996, pp. 104–113. DOI: 10.1007/3-540-68697-5_9.

[Moo+19] Thorben Moos, Amir Moradi, Tobias Schneider and François-Xavier Standaert. "Glitch-Resistant Masking Revisited or Why Proofs in the Robust Probing Model are Needed". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.2 (2019), pp. 256–292. DOI: 10.13154/tches.v2019.i2.256-292.

[Mor+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar and Huaxiong Wang. "Pushing the Limits: A Very Compact and a Threshold Implementation of AES". In: *Proceedings of the 2011 Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*. Springer, 2011, pp. 69–88. DOI: 10.1007/978-3-642-20465-4_6.

[MPG05] Stefan Mangard, Thomas Popp and Berndt M. Gammel. "Side-Channel Leakage of Masked CMOS Gates". In: *Proceedings of the Cryptographers' Track at the 2005 RSA Conference (CT-RSA)*. Springer, 2005, pp. 351–365. DOI: 10.1007/978-3-540-30574-3_24.

[MPO05] Stefan Mangard, Norbert Pramstaller and Elisabeth Oswald. "Successfully Attacking Masked AES Hardware Implementations". In: *Proceedings of the 2005 International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Springer, 2005, pp. 157–171. DOI: 10.1007/11545262_12.

[Nag+22] Rishub Nagpal, Barbara Gigerl, Robert Primas and Stefan Mangard. “Riding the Waves Towards Generic Single-Cycle Masking in Hardware”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2022.4 (2022), pp. 693–717. DOI: `10.46586/tches.v2022.i4.693-717`.

[NRR06] Svetla Nikova, Christian Rechberger and Vincent Rijmen. “Threshold Implementations Against Side-Channel Attacks and Glitches”. In: *Proceedings of the 2006 International Conference on Information and Communications Security (ICICS).* Springer, 2006, pp. 529–545. DOI: `10.1007/11935308_38`.

[NRS08] Svetla Nikova, Vincent Rijmen and Martin Schläffer. “Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches”. In: *Proceedings of the 2008 International Conference on Information Security and Cryptology (ICISC).* Springer, 2008, pp. 218–234. DOI: `10.1007/978-3-642-00730-9_14`.

[Osv+10] Dag Arne Osvik, Joppe W. Bos, Deian Stefan and David Canright. “Fast Software AES Encryption”. In: *Proceedings of the 2010 International Workshop on Fast Software Encryption (FSE).* Springer, 2010, pp. 75–93. DOI: `10.1007/978-3-642-13858-4_5`.

[Pet19] George Petrides. “On Non-Completeness in Threshold Implementations”. In: *Proceedings of the 2019 Workshop on Theory of Implementation Security (TIS).* ACM, 2019, pp. 24–28. DOI: `10.1145/3338467.3358951`.

[Pic+23] Enrico Piccione et al. “An Optimal Universal Construction for the Threshold Implementation of Bijective S-Boxes”. In: *IEEE Trans. Inf. Theory* 69.10 (2023), pp. 6700–6710.

[Pos+11] Axel Poschmann et al. “Side-Channel Resistant Crypto for Less than 2, 300 GE”. In: *J. Cryptol.* 24.2 (2011), pp. 322–345. DOI: `10.1007/s00145-010-9086-6`.

[Pra+05] Norbert Pramstaller, Stefan Mangard, Sandra Dominikus and Johannes Wolkerstorfer. “Efficient AES Implementations on ASICs and FPGAs”. In: *Proceedings of the 2004 International Conference on the Advanced Encryption Standard (AES).* Springer, 2005, pp. 98–112. DOI: `10.1007/11506447_9`.

[Rep+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs and Ingrid Verbauwhede. “Consolidating Masking Schemes”. In: *Proceedings of the 2015 Annual International Cryptology Conference (CRYPTO).* Springer, 2015, pp. 764–783. DOI: `10.1007/978-3-662-47989-6_37`.

[Rep15] Oscar Reparaz. *A note on the security of Higher-Order Threshold Implementations.* Preprint 2015/001. IACR Cryptol. ePrint Arch., 2015.

[Sas+20] Pascal Sasdrich, Begül Bilgin, Michael Hutter and Mark E. Marson. “Low-Latency Hardware Masking with Application to AES”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020.2 (2020), pp. 300–326. DOI: `10.13154/tches.v2020.i2.300-326`.

[Sim+22] Mateus Simoes et al. “Self-timed Masking: Implementing Masked S-Boxes Without Registers”. In: *Proceedings of the 2022 International Conference on Smart Card Research and Advanced Applications (CARDIS)*. Springer, 2022, pp. 146–164. DOI: 10.1007/978-3-031-25319-5_8.

[SS15] Abolfazl Soltani and Saeed Sharifian. “An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA”. In: *Microprocess. Microsyst.* 39.7 (2015), pp. 480–493. ISSN: 0141-9331.