Exploiting FPGA Block Memories for Protected Cryptographic Implementations

Bhasin, Shivam; Danger, Jean-Luc; Guilley, Sylvain; He, Wei

2015

Bhasin, S., Danger, J.-L., Guilley, S., & He, W. (2015). Exploiting FPGA Block Memories for Protected Cryptographic Implementations. ACM Transactions on Reconfigurable Technology and Systems, 8(3), 1-16.

https://hdl.handle.net/10356/83419

https://doi.org/10.1145/2629552

Exploiting FPGA Block Memories for Protected Cryptographic Implementations

Shivam BHASIN, TELECOM-ParisTech
Jean-Luc DANGER, TELECOM-ParisTech / Secure-IC S.A.S.
Sylvain GUILLEY, TELECOM-ParisTech / Secure-IC S.A.S.
Wei HE, Universidad Politécnica de Madrid

Modern Field Programmable Gate Arrays (FPGAs) are power packed with features to facilitate designers. Availability of features like large block memory (BRAM), Digital Signal Processing (DSP) cores, embedded CPU makes the design strategy of FPGAs quite different from ASICs. FPGAs are also widely used in security-critical application where protection against known attacks is of prime importance. We focus ourselves on physical attacks which target physical implementations. To design countermeasures against such attacks, the strategy for FPGA designers should also be different from that in ASIC. The available features should be exploited to design compact and strong countermeasures. In this paper, we propose methods to exploit the BRAMs in FPGAs for designing compact countermeasures. Internal BRAM can be used to optimize intrinsic countermeasures like masking and dual-rail logics, which otherwise have significant overhead (at least $2\times$) compared to unprotected ones. The optimizations are applied on a real AES-128 co-processor and tested for area overhead and resistance on Xilinx Virtex-5 chips. The presented masking countermeasure has an overhead of only 16% when applied on AES. Moreover Dual-rail Precharge Logic (DPL) countermeasure has been optimized to pack the whole sequential part in the BRAM, hence enhancing the security. Proper robustness evaluations are conducted to analyze the optimization in terms of area and security.

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: SPECIAL PURPOSE AND APPLICATION-BASED SYSTEMS—*Real-time and embedded systems*

Additional Key Words and Phrases: FPGA, Side-Channel Analysis, Block Memories, Countermeasures, Algorithm Architecture Adequation.

ACM Reference Format:
Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley and Wei He, 2013. Exploiting FPGA Block Memories for Protected Cryptographic Implementations. *ACM Trans. Reconfig. Technol. Syst.* V, N, Article A (January YYYY), 16 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Security is now one of the major driving factors of the high-end semiconductor industry. Often there is a need to secure the whole system-on-chip (SoC), which generally is achieved by embedded cryptographic cores (crypto-cores). Depending on the appli-

This research is partly supported by Strategic International Cooperative Program (Joint Research Type), Japan Science and Technology Agency (JST), and the French Agence Nationale pour la Recherche (ANR), via grant for project SPACES (Security evaluation of Physically Attacked Cryptoprocessors in Embedded Systems). We are grateful to Ankit Khandelwal (IIT, Bombay) and Dr. Amir Moradi (Ruhr-Universität Bochum) for interesting discussion about possible optimizations of RSM in terms both of size and of security.
Author's addresses: S. Bhasin, TELECOM-ParisTech, Paris (France). J.-L. Danger and S. Guilley, TELECOM-ParisTech, Paris (France) and Secure-IC S.A.S., Rennes (France). W. He, Universidad Politécnica de Madrid, Madrid (Spain).

cation, these crypto-cores are used to encrypt/decrypt sensitive data in all parts of the system, ranging from memory content to system-bus. A major threat known as "Side-Channel Attacks" (SCA [Brier et al. 2004]) has been pointed out about 17 years ago, but curiously the design of solid and efficient protections is still an open research area. SCA generally exploits the unintentional leakages from the physical implementation of the crypto-cores. This brings into play countermeasures to protect the physical implementation of cryptography, which can be classed into intrinsic and extrinsic countermeasures. Extrinsic countermeasures are applied in parallel to crypto-cores in order to confuse the attacker. Countermeasures involving generation of noise, misalignment of activity generally fall into this category [Güneysu and Moradi 2011].

Although extrinsic countermeasures have a comparatively less overhead, their resistance depends on the power of the attacker. Consider a noise generator which is deployed to provide $2\times$ SCA resistance w.r.t. the unprotected crypto-core. The power of the countermeasure is related to the extra effort required by the attacker to acquire twice the number of traces. If the attacker needs only a couple of seconds more to acquire the extra traces, then the security enhancement is negligible. Therefore a common practice is to combine several extrinsic countermeasures with protocol level countermeasures (such as *key refreshing* [Medwed et al. 2010]). However provable security is not assured.

Intrinsic countermeasures are the other solution which, as the name suggests, are built into the algorithm. These countermeasures modify the implementation of the cipher in order to leak little or no sensitive information from side-channels. Also these countermeasures often come with a non-negligible overhead. Intrinsic countermeasures further fall into two wide categories, i.e., masking and hiding.

Hiding countermeasures generally comprise of dual-rail precharge logic (DPL [Tiri and Verbauwhede 2004]). DPL is a circuit-level countermeasure which aims at flattening or removing the data-dependent leakage from the circuit. Removal of data-dependant leakage is achieved by putting in place a generated False (F) rail that works simultaneously together with the original True (T) rail for compensating each other's activity. DPL operates in two phases: *Precharge*, i.e., where all the non-memory logic cells are reset to a constant value, and *Evaluation*, where valid algorithmic data are computed and propagated. The two-phase operation with a dual-rail structure (theoretically) ensures constant activity and is therefore free from any exploitable data-dependent leakage.

Masking on the other hand is generally applied at the algorithmic level. The basic idea of masking is to protect all the sensitive intermediate values inside a cryptographic algorithm by applying a random mask [Goubin and Patarin 1999]. The random mask is removed at the end, which involves complex computation on the value of mask, generally done by implementing the masked path in parallel to the actual algorithm. The linear operations of a cryptographic algorithm can be easily tuned to masking. Masking the non-linear operations is not an easy task, as the overhead associated with it is exponential in the data bitwidth.

For a secure implementation, DPL needs balanced placement and routing of its component. Masking does not have such strict requirements at the circuit level but the non-linear operation is often hard to be realized in a secure manner. The availability of high-density block memories (BRAMs) in FPGA can help to solve both problems. BRAMs are capable of storing large tables, which are often present in the non-linear part of protected ciphers (e.g., masked/dual-rail substitution boxes, aka Sboxes). Thus intrinsic countermeasures become realizable in FPGA thanks to embedded BRAMs. Several other features (discussed in Sect. 2.1) are present in BRAMs which can be exploited to optimize the implementation of the cipher. BRAMs are also known to provide elevated security as compared to its logic counterpart [Velegalati and Kaps 2010], and are often recommended to implement intrinsic countermeasures. BRAM are also largely deployed in implementing hash function and other cryptographic applications.

In this paper, we concentrate on BRAMs present in FPGAs in the context of intrinsic countermeasures. In particular, we propose methods to efficiently use BRAM to implement countermeasures with reduced area overhead and higher SCA resistance. Although generic countermeasure is favourable, it as well makes sense to exploit new features to realize compact and robust countermeasure. Firstly, we propose a method to exploit the features of BRAM in order to implement masking and DPL countermeasures with limited overhead. The proposed optimizations are applied on a real AES-128 co-processor. All the AES implementations tested implement the Sboxes in BRAMs, as this configuration has been shown to offer enhanced resistance against SCA [Velegalati and Kaps 2010]. Next we analyze the security of these countermeasures in the presence of BRAM. We show that it is possible to use modern FPGA features to effectively implement intrinsic countermeasures.

The rest of the paper is organized as follows: Sect. 2 gives general background on BRAM architecture in FPGA, its application in masking and DPL countermeasures. Next in Sect. 3, we propose two methodologies to exploit BRAM features in an FPGA to optimize masking and DPL countermeasures respectively. The proposed optimization are applied on an AES-128 co-processor for experimental validation. The SCA evaluation of proposed protection methodologies is discussed in Sect. 4. Finally, Sect. 5 draws general conclusion.

2. BRAM IN CRYPTOGRAPHIC APPLICATIONS

In this section, we first discuss the features of an FPGA BRAM. A special focus is laid on the application of these features to optimize SCA countermeasures. Thereafter a general background of the used countermeasures, i.e., Masking and DPL, are provided.

2.1. Block RAM in Modern FPGA

Modern FPGAs possess large blocks of memories which are synchronous in nature. For example, the latest Xilinx FPGAs have several blocks of 36 Kbits true dual-port memories. The exact design of these BRAMs is not public but a few details about the general architecture of these BRAMs are documented [Xilinx 2011]. Fig. 1 shows one port of a dual-port BRAM in Spartan-6 FPGA. It can be deduced from the figure that the BRAM contains register to synchronize input data and address before accessing the memory array. The memory array is followed by a latch and an optional output register. BRAM also contains several signals to control the use of output register or set/reset the value of the latch and output register. The latch has been characterized to leak in Hamming Distance at the clock falling edge [Bhasin et al. 2013]. Altera AltSyncRams [Altera 2011] also possess a similar BRAM architecture. Therefore the presented solution can also be extended to Altera FPGAs.

As previously stated, BRAMs are recommended for crypto-applications. Tab. I summarizes the features of a BRAM and their use in relation to cryptographic applications.

Some of these options have already been used in cryptographic applications. **Internal Register at input** for state and **Dual-Port Nature** was first used by Drimer et al. in [Drimer et al. 2008]. **Reset** in the BRAM was also used in Separated Dynamic Differential Logic (SDDL [Velegalati and Kaps 2010]) to enable precharge propagation. In [Bhasin et al. 2013], authors have shown that the internal register at input (address) of BRAM leaks very little and is difficult to attack. Moreover, we assume that it is very unlikely to separate activities of the two ports of a BRAM being a hard-macro in $\leqslant 65$ nm CMOS.

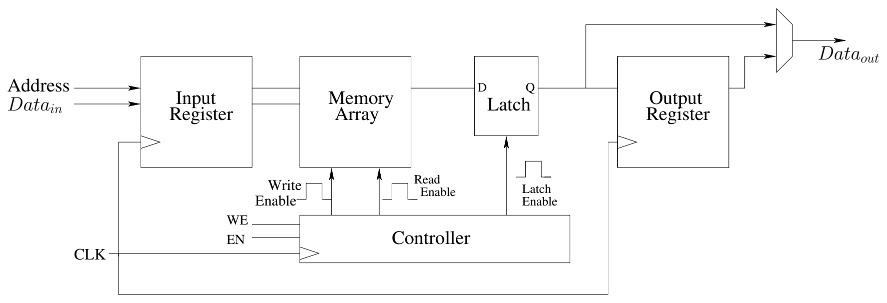


Fig. 1. Internal Architecture of Xilinx BRAM.

Table I. Features of a Xilinx BRAM. In the table, exponent 1 signifies an improvement in *area* or *performance*, whereas exponent 2 signifies an improvement in *SCA resistance*

BRAM Feature	**Application to Cryptography**
High Density RAM	To implement large data (such as GLUT)^1
Internal Register at input	To implement state register^1 Not connected to FPGA routing^2 No glitches^2
Dual-Port Nature	Single block for multiple Sboxes^1
Output Register	Available resource^1 To achieve better timing^2
Reset	To enable precharge propagation in DPL^{1,2}
Hard Macro in $\leqslant 65$ nm CMOS	Low leakage power^2 To balance placement^{1,2}

2.2. Masking and the use of BRAM

Masking relies on variable representation of sensitive data into randomized shares [Chari et al. 1999]. A d^{th}-order masking scheme splits a sensitive variable $Z \in \mathbb{F}_2^n$ into $d+1$ random shares, noted $\vec{S} = (S_i)_{i \in [\![0,d]\!]}$, in such a way that the relation $S_0 \perp \cdots \perp S_d = Z$ is satisfied for a group operation $\perp$ (e.g., the XOR operation in Boolean masking). For a simple Boolean masking scheme, order $d = 1$. When masking is implemented in hardware, generally the mask as well as the masked data are computed in parallel. Keeping this detail in mind, the leakage function for the first-order masking countermeasure in hardware can be expressed as:

$$L = HW(Z \oplus M) + HW(M) + B \ , \tag{1}$$

where HW is the Hamming weight and B then noise. The share M is the random mask uniformly distributed over $\mathbb{F}_2^n$ and the share $Z \oplus M$ is the masked variable. Variables Z and M are assumed to be mutually independent. The linear parts of the cipher are easier to be masked but the computation of non-linear Sbox S in presence of masking is difficult. It involves computing $S(Z) \oplus M'$ from the variables M, $Z \oplus M$ and M' (new mask) without compromising with SCA resistance.

To deal with this problem, one of the most common solutions is *the Generalized Look-Up Table* (GLUT [Prouff and Rivain 2007]). The main idea of GLUT is to precompute a look-up table, associated to the function $S' : (X, Y, Y') \mapsto S(X \oplus Y) \oplus Y'$. To compute the masked variable $S(Z) \oplus M'$, GLUT performs a table look-up of $GLUT[Z \oplus M, M, M']$. Thus the value $S(X \oplus Y) \oplus Y'$ has been precomputed for every possible 3-tuple of values. For first-order masking, the output mask and the input mask are equal (i.e., $M = M'$). In this case, the dimension of the table is $2n$ instead of $3n$ and the look-up table becomes $GLUT[Z \oplus M, M]$, where Z, M and M' are variable of n-bits. Owing to its structure the preferred target is a BRAM. Compared to an unprotected Sbox S $(n \rightarrow p)$ of size $2^n \times p$, a first-order masking GLUT requires $2^{2n} \times 2p$ space. Very often the hardware implementations computes the whole state in parallel, requiring multiple instances of GLUT. Therefore the basic GLUT technique can be sometimes difficult to be realized in FPGA when n is high (for example $n = 8$ in AES). The size of GLUT further explodes when the desired resistance is of order $d \geqslant 1$.

An optimized version of GLUT in FPGA logic was proposed in [Regazzoni et al. 2011] with a net overhead of roughly $3\times$. However the implementation of GLUT in logic is sensible to higher-order attacks (albeit of small order) which exploit the leakage due the glitches. In [Güneysu and Moradi 2011], authors propose a first-order SCA resistant countermeasure using BRAM scrambling. BRAM scrambling implements a $2^n \times p$ masked Sbox with a single mask. This Sbox uses the same mask for several encryption, which limits the order of SCA resistance. In the mean time, another Sbox which is masked with a different mask is written to the other port of BRAM. Once the second Sbox is ready, it is used for encryption while the first Sbox is refreshed with a new mask. Another first-order countermeasure in the same line was proposed in [Nassar et al. 2012], which proposes the reuse of Sboxes to reduce overhead. The main advantage of this masking scheme is that it does not need a parallel mask-computation path which also forms a basis for our masking scheme. Our masking scheme uses "precomputed" Sboxes with a random (secret) offset for every encryption. We show that it is possible to design a masking scheme with reduced entropy $< n$ bit, and achieve *monovariate* SCA resistance up to order d for a well chosen set of mask.

2.3. DPL and use of BRAM

The *modus operandi* of dual-rail circuits is to add redundant logic of opposite nature to achieve constant activity irrespective of the data processed. A DPL protocol converts every bit x to a pair $(x_{\mathrm{T}}, x_{\mathrm{F}})$. Complementary values of x_{T} and x_{F} are desired for a proper balance and thus considered as valid values. Similar values for the pair $(x_{\mathrm{T}}, x_{\mathrm{F}})$ can be used as separators between valid values. Thus DPL operates in two phases where valid values are propagated in evaluation phase and a spacer in precharge phase. Following the conditions stated above, DPL ensures a constant activity of each compound gate pair. However, when DPL expands from a single gate to a complex circuit, different placement and routing delays introduces other imbalances.

Fig. 2 shows the Wave Dynamic Differential Logic (WDDL [Tiri and Verbauwhede 2004]): one of the first introduced DPL also suitable for FPGA. It can be deduced that all logic gates (except inverters) lead to an overhead of 2 while flip-flops result in an overhead of 4. WDDL also has a restriction of using only positive gates which further adds up to the overhead. In Fig. 2, the gates G and $\overline{G}$ are well balanced but if their inputs arrive at different time, an imbalance cannot be avoided. Thus proper placement and routing are required for a secure DPL design, in absence of which, DPL could fail due to early propagation effect (EPE [Nassar et al. 2010]) or routing imbalance [He et al. 2012]. EPE arises from different evaluation time of a logic gate depending on difference in arrival of inputs, which can be due to the different logic depth of the several inputs. Routing imbalance is observed due to asymmetrical routing of T and F rails. Since then, several improvements to WDDL have been proposed to improve its resistance. One interesting proposal to counter the routing imbalance was called MDPL (Masked Dual-rail Precharge Logic). MDPL randomly swaps the true and false routing network to eliminate routing imbalance and also EPE in iMDPL (improved MDPL [Popp et al. 2007]). This security improvement of iMDPL came at an area overhead even greater than WDDL.

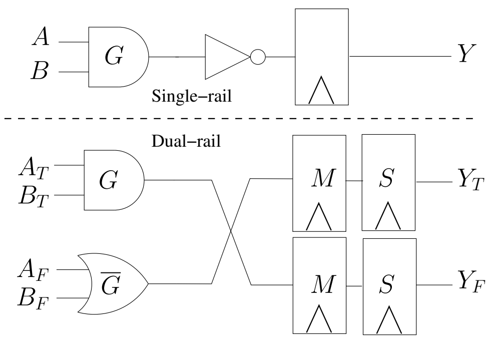


Fig. 2. WDDL building block. Top: *unprotected*; Bottom: *WDDL equivalent*.

Due to the discussed issues, DPL was not considered as a good countermeasure specially for FPGA application where a designer has very limited freedom over choice of gates, placement and routing. Thereafter a couple of DPL countermeasures were proposed which were able to use BRAM at a reasonable cost. One of the BRAM based DPL, is SDDL [Velegalati and Kaps 2010]. SDDL used BRAMs at an area overhead of $2\times$ compared to the unprotected design. This limited overhead comes from the reset feature present in the Xilinx BRAM which can reset the output as desired. The reset was used for precharge propagation at the output of the Sbox. Another DPL called BCDL (Balanced Cell-based Dual-rail Logic [Nassar et al. 2010]) can also use BRAM at an overhead of $4\times$ owing to a synchronization. The synchronization signal of BCDL also solves the problem of EPE. However both SDDL and BCDL do suffer from routing imbalance and therefore need back-end techniques for balancing the dual-rail. AES with T-tables reduces the fanout which in a way reduces routing imbalance and makes back-end balancing easier [Bhasin et al. 2011].

3. EXPLOITATION OF BRAM TO OPTIMIZE COUNTERMEASURES

In this section, we propose two methods benefited from BRAM features for implementing secure circuits at a reasonable cost. The first method is applied to masking countermeasures by exploiting large memory array and dual-port nature of the BRAM. The next method presents a new way (using BRAM) to organize the sequential part of crypto-algorithm in a compact and balanced manner.

3.1. Optimized Masking Implementation using BRAM

Several solutions are proposed to mask the non-linear operation (now called Substitution box or Sbox) of a cipher but all solutions have a significant overheads. Since we are using BRAM in our implementation, we focus on GLUT as the solution to mask the Sbox. GLUT is a precomputed table which accepts the masked Sbox input (n-bits) and the mask (n-bits) as inputs. It returns a masked Sbox output (p-bits) and the correction value (p-bits). For example, in DES a 64×4 Sbox is replaced by GLUT of size 4096×8. Similarly for AES, the size of the GLUT is 65536×16 for a 256×8 Sbox. Please note that in hardware where implementations are parallel in general, several instances of a Sbox are used and all of them must be masked. In a low-cost FPGA like Xilinx Virtex-5 LX30, a parallel DES implementation is still possible but not for AES. A single AES GLUT would occupy about 90% of the available BRAM, making a parallel AES implementation unfeasible.

It is possible to design a masking implementation which reduces the overhead of GLUT still keeping it resistant to the certain higher order of side-channel attacks.

Masking schemes can reduce the GLUT overhead by reusing the mask and thus reducing the overhead from $2^{2n} \times 2p$ to $2^{n+k} \times p$ where $k < n$ is the entropy of the mask. In other words, instead of using 2^n different values to mask the data, only 2^k values are used. For a proper hardware optimization, the number of Sboxes in a cipher N should be a multiple of k. Such an implementation generally protects against first-order attack, however by application of coding theory, the right set of mask can be chosen to resist zero-offset higher-orders (univariate attacks targeting a single Sbox [Nassar et al. 2011]). In the following, we consider univariate attacks which combine different leakages.

For simplicity, we restrict ourselves to ciphers (e.g., AES, PRESENT) where all the N Sboxes are the same and of bijective construction, i.e., of the format $2^n \times n$. Ciphers not abiding by these conditions are still possible to protect by this scheme with an extra overhead. The details of this masking scheme are as follows. Firstly, a set of 2^k n-bit mask M is chosen. Now both the input and output of each Sbox S are masked as: $S(x \oplus m_i) \oplus m_{i+1}$ where m_i and m_{i+1} are consecutive elements of the set M. Actually i and $i+1$ are to be understood as $(i \mod 2^k)$ and $((i+1) \mod 2^k)$ (omitted for simplicity of representation). The masked Sbox is now denoted as S_m and is of the same size as unmasked S. If 2^k is equal to N then all the Sboxes are unique. At each round of the algorithm, the Sboxes S_m are reused by circular rotation of one position. Let us consider a masked state $x' = x \oplus m_i$ is computed by S_{m_i} which is masked with m_i in the current round r. In the next round, x' is processed by $S_{m_{i+1}}$. Precisely the computation done by $S_{m_{i+1}}$ will be $S(x' \oplus m_i) \oplus m_{i+1}$ which is simplified to $S(x) \oplus m_{i+1}$. Similarly in the next round, mask m_{i+1} is removed at the input of Sbox $S_{m_{i+2}}$ and m_{i+2} is applied at the output. If the Sboxes are not bijective, an expansion function should be put in place to make the output of Sbox coincide with size of the mask.

The set of mask M can be public however the M should be shifted by a random offset before each encryption. M is chosen such that the jth order moment of the conditional leakage $\mathbb{E}(L^j | Z = z)$ given a guess on the sensitive variable Z are all the same for $j = 1, 2, \cdots, d$. Thus only an attack of order $(d+1)$ can succeed. Under this constraint, the masks set M must be an orthogonal array of strength d [Hedayat et al. 1999]. The linear operations are masked by a simple XOR operation with precomputed constants applied at the end of each round. The $N \times n$ bit constants are chosen as a function of initial offset and can be stored in BRAM as well. It is not always possible to find a solution for M which resists at order d. Another feature of FPGA which comes handy in such cases is dynamic reconfiguration. If it is not possible to find a solution for M at order d, designers can opt for several sets of M with order $< d$ and update them regularly. Since the mask dependent part is inside the memory, modern FPGA kits have specific tools which can reconfigure the FPGA to just change the BRAM content. Alternately, concurrent read and write technique used in [Güneysu and Moradi 2011] can be used by doubling the memory overhead.

The required rotation in the presented masking scheme can be done using barrel shifters. Since the barrel shifters are composed of series of multiplexers which are a major source of glitches in FPGA, they can cause unintentional leakage. Barrel shifters are also resource consuming and affecting the performance of the whole system. For example, a 128-bit barrel shifter would alone acquire around 100 slices in Xilinx Virtex-5 LX-30 FPGA. BRAMs can be very efficiently used in this application to get rid of barrel shifters and thus glitches. The scheme to organize the Sboxes and implement them in BRAM is shown in Fig. 3. These Sboxes can be further compressed by using dual-port memory. All the masked Sboxes S_m are placed in each BRAM. From one BRAM to another, S_m are laid with an offset of 1. Thus the BRAM has an input address of $n + k$ bits where n is the input of the S_m and k selects the correct masked Sbox from S_{m_0} to $S_{m_{N-1}}$. k forms the most significant bits of $n + k$. Thus the memory cost is multiplied

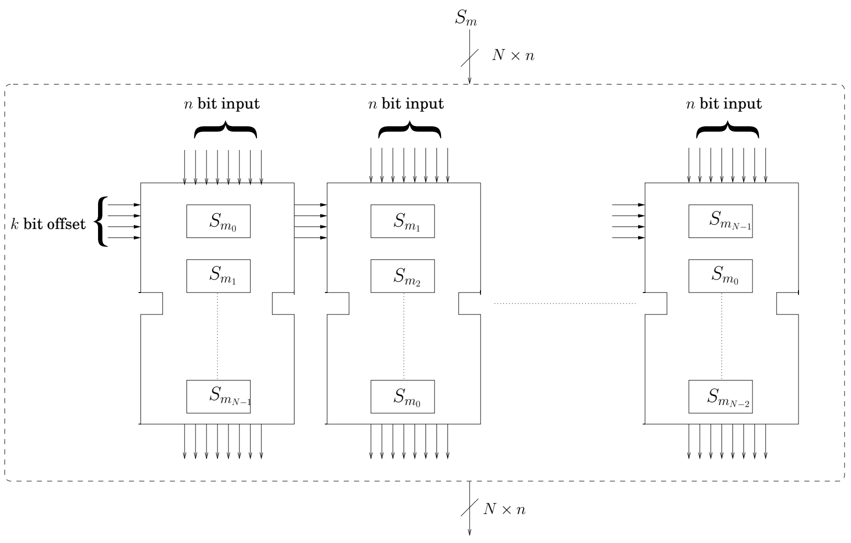


Fig. 3. Optimized implementation of proposed masking scheme without barrel shifters

by 2^k but can be small in terms of number of blocks. Since all the BRAM contain same Sboxes in different order, the dual-port feature can be used to access the same data with the corrected offset.

3.1.1. Application to AES-128. Now we apply the presented scheme to secure a parallel AES-128 co-processor which computes one round per clock cycle. For AES, $n = 8$ and $N = 16$. We found that it is possible to select a mask M for AES which resists up to order $d = 3$. M is the (set of the) cosets of the linear code $[8, 4, 4]$ and thus $k = 4$ for 16 masks. It can be proven that if the leakage is affine (a general case of "Hamming weight" shown in Eqn. (1)) [Carlet and Guilley 2013], the set

$$M = [\texttt{0x00, 0x0f, 0x36, 0x39, 0x53, 0x5c, 0x65, 0x6a, 0x95, 0x9a, 0xa3, 0xac, 0xc6, 0xc9, 0xf0, 0xff}] \quad (2)$$

should be order 3 resistant. To optimize the scheme we use the input register of BRAM as state register. An unmasked AES Sbox is 2 Kb which makes the composite Sbox (S_{m_0} to $S_{m_{N-1}}$) of size $16 \times 2 = 32$ Kb. This composite Sbox which easily fits in a Xilinx BRAM of 36 Kb, now has 12 bits of address, i.e., 8 bits corresponding to masked byte concatenated with 4 bits of offset. Moreover the dual-port feature of the BRAM can reuse the same memory space with two different ports. Thus $N = 16$ Sboxes need only $N/2 = 8$ BRAMs. The overhead of presented masking scheme as compared to unprotected reference AES is shown in Tab. II. The precomputed round unmasking constants are implemented in BRAM which consumes 8 extra blocks. Round unmasking constants can also be implemented in FPGA logic. Doing so, would require 128 slices instead of occupying 8 BRAMs at the risk of some unwanted glitches. The net overhead in terms of slices is only 16% with minor loss of frequency. Since higher-order attacks of order 4 and greater are difficult to realize in practice [Moradi 2012], an order 3 masking with mere overhead of 16% is a very practical solution.

3.2. Optimized DPL Implementation using BRAM

DPL involves duplication of each component of the circuit to ensure a balanced activity. Duplication of standard logic is simple which leads to an overhead of little over twice in terms of resources used. However a simple duplication of memory leads into exponential increase in overhead. A memory of size $2^n \times p$, will have an overhead of 2^{n+1}

Table II. Area and frequency overhead of masked AES after optimization on Virtex-5

Architecture	Unprotected	Masked	Overhead
Slices	733	856	$1.16\times$
Registers	0	0	$0\times$
BRAM	8	16	$2\times$
Max. Frequency [MHz]	144.3	141.1	$1.02\times$

up on duplication. This overhead can be reduced to just $2\times$ by using BRAM properties. The BRAM overhead is not the only problem. For a DPL circuit to have a constant activity in every cycle, a precharge spacer should flow through the whole circuit.

We propose a method to further optimize FPGA implementations of DPL both in terms of area and security. This optimization exploits the following features of BRAM: input register, output register with reset, dual-port nature and hard macro. A DPL flip-flop is made of 4 flip-flops (Fig. 2), where each flip-flops pair (master-slave) is located in the true and false rails. The input register can be used for the master flip-flop and the output register serves as the slave. The use of output register also introduces a latency of one clock cycle. The extra cycle latency is not a problem in DPL because it aids the two-phase DPL protocol. Moreover, the dual-port feature allows to implement the true and the false rails of the flip-flop. The optimization scheme is depicted in Fig. 4; it is a special instantiation of the Xilinx BRAM (refer to Fig. 1).

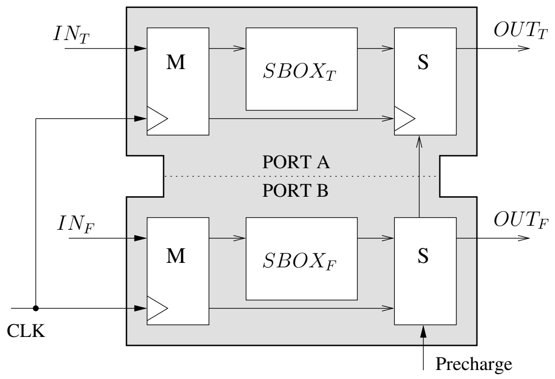


Fig. 4. Proposed scheme to implement a DPL Sbox and Flip-Flops in a BRAM.

A very common issue in DPL design is the propagation of precharge or the spacer. Since the BRAM will be preceded by some combinational circuit, the spacer is easily propagated to the input register of the BRAM. To precharge the output register, the reset (also known as SSR [Velegalati and Kaps 2010]) feature provides just the right solution. Only the combinational gates are implemented in FPGA slices. The proposed architecture brings a three-fold advantage for implementing DPL design into FPGA. Firstly, the logic is not used to implement two-stages (otherwise leading to $4\times$) of the state registers thus significantly reducing the overhead. Secondly, the regular structure of BRAM ensures proper and balanced placement of the main leakage source of the design, i.e., state register. Finally, it is known that leakage from a BRAM itself is less than flip-flops in FPGA slices, thus enhanced SCA resistance [Velegalati and Kaps 2010]. The balanced placement of XOR gates can be ensured by using $LUT6\_2$ from Xilinx to place the whole dual-rail cells (G and $\overline{G}$ in Fig. 2). Balancing routing in FPGAs is challenging, because FPGA architecture and CAD tools are not designed for these weird DPL structures. But some amount of balancing can be achieved by proper placement. Balancing routing is another area of research, and repair techniques like proposed in [He et al. 2012] can repair routing with extra effort, but it falls out of the scope of this paper.

3.2.1. Application to AES-128. To test our proposed optimization on a real DPL circuit, we had two choices: SDDL and BCDL. To our knowledge, SDDL and BCDL, are the only DPL subsets proposed which are capable of using BRAM at reasonable cost. Since SDDL suffers from security issue like EPE, we choose to apply our optimization on BCDL. The target algorithm is AES-128 using T-tables because T-tables merge SubBytes and MixColumns functions in a precomputed table, thus reducing the routing fanout. Precisely, we use the implementation of AES-128 protected by BCDL as described in [Bhasin et al. 2011] (detailed in Appendix A.1). Applying our technique to further optimize BCDL, BRAMs and flip-flops are merged into a single entity.

Table III. Area and Frequency Overhead of BCDL for AES module excluding key expansion after optimization on Virtex-5

Architecture	**Unprotected**	**BCDL**	**Overhead**
Slices	176	1128	$6.4\times$
Registers	0	0	$0\times$
BRAM	8	16	$2\times$
Max. Frequency [MHz]	258	283	$0.911\times$

The overhead of protecting AES with BCDL after applying our optimization is given in Tab. III. Both the unprotected AES and its BCDL version implement state registers in BRAM. The number of slices is increased by roughly $6.4\times$ as XOR in BCDL is costly, as also pointed in the original paper [Bhasin et al. 2011]. It is limited to 1-LUT per bit of XOR. The BRAMs are simply doubled, while the performance is improved due to the usage of output register of the BRAM. As BRAMs are a hard-macro, balanced placement of the sequential part of AES (128×4 bits registers) is ensured without placement constraints. DPL balancing has been checked by the post P&R i.e., absolutely close to on-device conditions. Besides, BCDL is free of glitch by design. We do a proof of concept study to quantify the gain of balanced placement and keep routing untouched for the two designs to have a fair evaluation.

4. SECURITY ANALYSIS

The previous sections dealt with the aspect of the proposed optimizations to masking and DPL. Now we analyze the implemented countermeasures from a security aspect with respect to SCA.

4.1. Attack Metrics and Experimental Platform

Let us denote a random variable L representing the side-channel leakage (e.g., power consumed) while computing $Z = f(X \oplus k)$. In this equation, k is the n-bit secret key and X is a variable quantity known to the attacker. Time is another parameter not shown. A standard SCA tries to find correct key $k^\star$ for which Z and L have maximum dependency. Since L is noisy, thus several measurements of Z are required to estimate L. For hardware implementation, the leakage L depends on the *Hamming Distance (HD) model*. It expresses at first-order the power consumption of CMOS gates in electronic devices as it corresponds to signal transitions. The leakage can be expressed as:

$$L = HD(Z, R) + B = HW(Z \oplus R) + B\,,$$

where B is the noise, and R is the reference state.

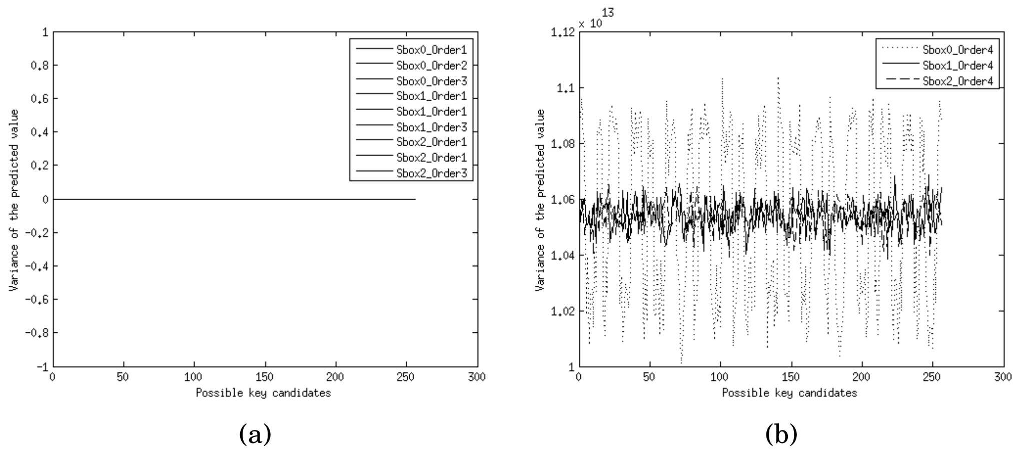


Fig. 5. Variance of the predicted leakage for first three Sboxes at (a) order 1, 2 and 3; (b) order 4 leakage L

For SCA analysis, we use Correlation Power Analysis (CPA [Brier et al. 2004]) as a distinguisher. CPA is a computation of the *Pearson Correlation Coefficient* ρ between the side-channel leakage T and the expectation of the leakage model L knowing Z, noted $\mathbb{E}(L|Z)$, which is equal to:

$$\rho(T, \mathbb{E}(L|Z)) = \frac{\mathbb{E}((T - \mathbb{E}(T)) \cdot (\mathbb{E}(L|Z) - \mathbb{E}(L)))}{\mathsf{Var}(T)^{1/2} \cdot \mathsf{Var}(\mathbb{E}(L|Z))^{1/2}} ,$$

where $\mathbb{E}$ and Var denote the expectation and the variance respectively. To analyze the efficiency of SCA, two metrics are used [Standaert et al. 2009]. The first metric is Minimum Traces to Disclosure (MTD), i.e., the minimum number of measurements needed to perform a successful attack. The other metric used is called guessing entropy which generally is useful when an attack is not successful. Guessing entropy gives the average number of key hypotheses to test to reveal the correct key.

We test our designs on Xilinx Virtex-5 FPGA soldered on a SASEBO-GII platform. For SCA, traces are acquired on a 54855 Infiniium Agilent oscilloscope with a bandwidth of 6 GHz and a maximal sampling rate of 20 GSample/s, using an antenna of the HZ–15 kit from Rohde & Schwarz. Since the analysis results can widely vary from one measurement setup to another, we always use a reference implementation to give readers an idea of the security gain achieved.

4.2. Security Analysis of Masked AES

To develop the leakage-function of masking we refer back to Eq. (1). For a first-order mask, the prediction function $z \mapsto \mathbb{E}(HW(Z \oplus M) + HW(M)|Z = z)$ reduces to a constant and makes simple SCA attacks impossible. To exploit the leakage from masked implementation zero-offset SCA [Waddle and Wagner 2004] are often used. These attacks are based on the principle that higher-order moments are related to the key.

For a masking of order d, the $d{+}1$ order moment can be key-dependant. Theoretically, $\rho((T - \mathbb{E}(T))^{d+1}, \mathbb{E}(L|Z))$ should result in a successful attack, where $T - \mathbb{E}(T)$ are the centered side-channel traces. However, as the order d increases, CPA becomes less practical because the noise in the traces is amplified. Moradi in [Moradi 2012] suggests that attacks of order $\geqslant 4$ can be considered far from practice. We acquired $150,000$ traces (averaged 16 times) for the masked implementation explained in Sect. 3.1.1.

The presented masking is a special case where the number of mask is 16 and the mask set is public. The secret is the 4-bit offset which is not known to the attacker. In this case, the leakage function can be written as $HW(Z \oplus M)^d$, d being the power at which the attacker raises the centered traces. The prediction function is $z \mapsto \mathbb{E}(HW(z \oplus M)^d)$, i.e., the leakage to the power d averaged over the whole set M for all offsets. We tested the values of possible predicted leakage (i.e. $\mathbb{E}((z \oplus M)^d)$) for 256 possible subkeys of two chosen Sbox for order $1-4$. As expected, the predictions came out to be constant for order 1, 2 and 3 which renders the attack impractical. Since the actual prediction is constant, its Hamming weight will also be constant. Only at order 4, the predictions vary from each other as shown in Fig. 5 (b) which points towards possibility of an attack. In Fig. 5 (b), it is normal to have a higher variance of prediction in Sbox 0 as it covers only 163 of the 256 values due to absence of ShiftRows. Sboxes 1 and 2 cover all 256 values and thus show lower variance. We tried a 4-th order attack on the set of acquired traces which failed probably due to limited number of traces or high noise at order 4. Thus the masking scheme is shown to be compact and secure at least up to order 3.

4.2.1. Security Validation on Masked AES with Reduced Constraints. In order to ascertain that your high-order order attack software is implemented correctly, we reduce some constraints on the masking implementation. In this evaluation, the masks set is:

$$M = [\texttt{0x00}, \texttt{0xff}, \texttt{0x00}, \texttt{0xff}, \texttt{0x00}, \texttt{0xff}, \texttt{0x00}, \texttt{0xff}, \texttt{0x00}, \texttt{0xff}, \texttt{0x00}, \texttt{0xff}, \texttt{0x00}, \texttt{0xff}, \texttt{0x00}, \texttt{0xff}].$$

This (artificial) mask set contains two complementary values and resists only attacks of order 1. Another major change was done in the implementation of the state register. Instead of implementing the state register as the input register of the BRAM, we implemented the state register in logic. By doing so, we increase the signal to noise ratio (SNR) of the leakage. It is shown that the leakage SNR can be multiplied by a factor as high as $100\times$, just by moving the state register from BRAM to FPGA logic [Bhasin et al. 2013]. Implementation of state register in logic might also introduce some extra glitches, which are otherwise hidden inside the BRAM. These glitches are capable of leaking sensitive information [Mangard et al. 2005]. In other words, the described changes strengthen the profile of the attacker.

We again acquired 150,000 side-channel traces (averaged 16 times) for this masked implementation with reduced constraints. First of all, we check the quality of side-channel leakage, by performing a CPA on the traces with the knowledge of mask set and random offset. The attack reveals the secret key in **4,700** traces. This is equivalent to attacking an unprotected implementation. Next we repeat a CPA, without any knowledge of the random offset to detect any first-order leakage. Since the chosen mask set resists first-order attacks, a CPA on these 150,000 traces fails to reveal the secret key (as shown in Fig. 6(a)). Next we perform a second-order zero-offset attack using $z \mapsto \mathbb{E}(HW(z \oplus M)^2)$ as the leakage prediction function. The traces are centered before applying the attack which eventually computes $\rho((T - \mathbb{E}(T))^2, \mathbb{E}(HW(z \oplus M)^2|Z))$. This attack of order 2 is able to break a masking implementation resistant to order 1 attacks with **40,000** traces. This is an $8\times$ increase w.r.t. to a 1-order protection; it explains why the implementation with the mask given in Eqn. (2) does not break with $150,000$ traces. Fig. 6(b) plots the result of the attack for Sbox 0, where we can clearly distinguish the correct key from all the false keys. Thus we can conclude that the proposed masking implementation of order d can be broken at order $d+1$. However as the order increases, certain practical factors (e.g., noise) make the attack more difficult.

4.3. Security Analysis of DPL AES

Theoretically, a DPL design should be leakage-free. DPL is a special case where any two evaluations are separated by a precharge phase. In a well-balanced DPL circuit,

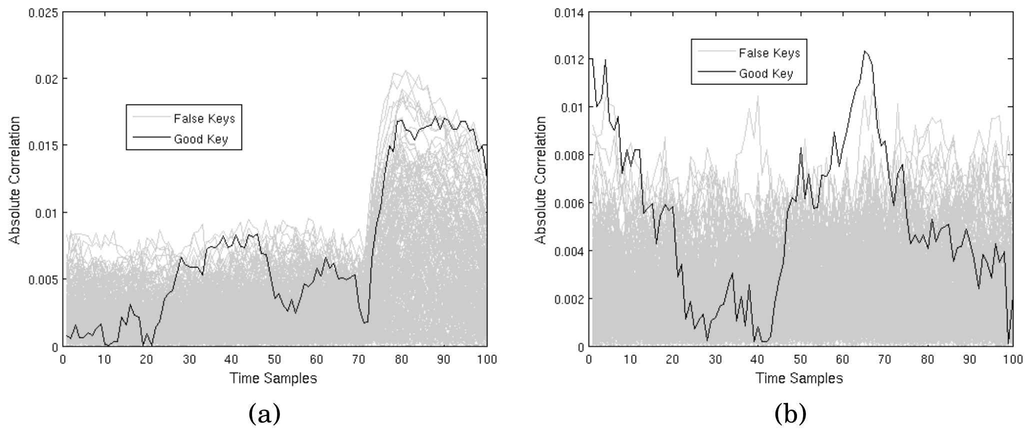


Fig. 6. Results of a CPA attack (a) order 1; (b) order 2, on a first-order resistant masking scheme

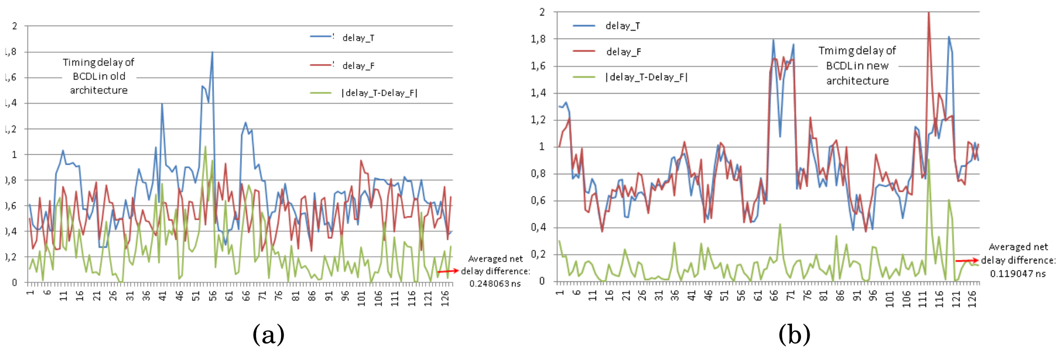


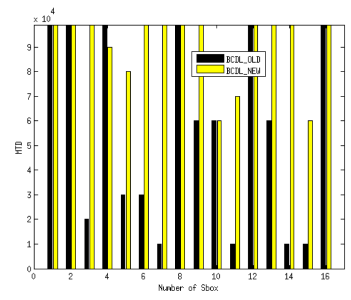
Fig. 7. Dual-rail timing bias in (a) BCDL_OLD, (b) BCDL_NEW

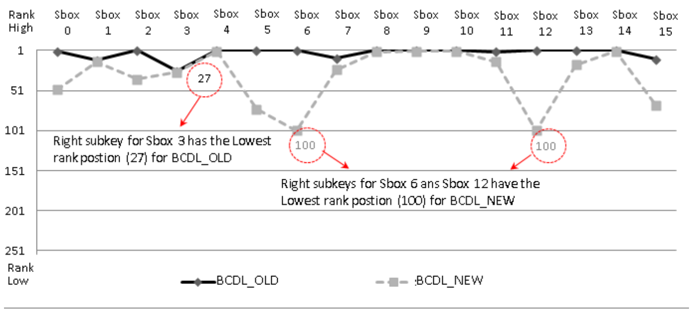
along with every Z, a $\overline{Z}$ is also computed, which can be modelled as:

$$L = HW(Z) + HW(\overline{Z}) + B\,.$$

Ideally, $HW(Z) + HW(\overline{Z})$ is a constant equal to n which reduces the leakage L to just noise. Now if we also consider placement and routing imbalance, Z and $\overline{Z}$ do not occur simultaneously. Thus for short periods of time, albeit capturable by an oscilloscope, L depends either on $HW(Z)$ or $HW(\overline{Z})$, which reduces the model to $HW(Z)$.

Now we try to quantify the security improvement brought due to balanced placement of BCDL by the proposed optimization. As previously stated, net delay bias has significant impacts on the balance between the dual rails of DPL logic. We achieved better routing balance between the nets for the security sensitive nets in the optimized BCDL (now referred as BCDL_NEW) as compared to original BCDL (now referred as BCDL_OLD) version. We would like to remind the readers that the BCDL_OLD implements the state register in FPGA slices. Fig. 7 depicts net delays and the differences between each of the 128 pairs of input nets to the flip-flop (same as BRAM input in BCDL_NEW). We choose these nets because they accumulate the maximum delay and therefore are most sensible to bias. Values for T and F rails are outlined by different colours. BCDL_NEW has a smaller delay difference as compare to BCDL_OLD. Aver-





(a) (b)

Fig. 8. (a) MTD and (b) Guessing entropy for the two BCDL circuits after CPA on 100k traces

aged delay bias from the old BCDL to the new one is reduced roughly from **0.25 ns** to **0.12 ns**, roughly a reduction factor of **1.48**.

Further we analyzed the two architectures using CPA over 100,000 traces which were averaged 16 times. The result is shown in Fig. 8. The Sboxes with MTD more than 100,000 traces indicate insufficient traces for a successful attack. So we plot the guessing entropy of the correct key in Fig. 8(b). It can be simply deduced from the plot that with the optimization, the resistance has been improved. We cannot directly connect the timing result with the CPA result because of lack of precise information on the physical properties of the device. However, both the timing and CPA results favour the improved BCDL (BCDL_NEW).

5. CONCLUSIONS AND PERSPECTIVES

In this paper, we investigated the power of BRAMs available in FPGAs to implement intrinsic countermeasures. BRAMs possess many features which can aid the design of cryptographic circuits. These features like presence of registers at input and output, ability to reset the output register, dual-port nature can be very well exploited. Also the regular structure of BRAM (hard-macro) saves the designers from applying specific placement constraints. We exploit these features to propose compact and secure implementation of existing countermeasures (masking and DPL). The optimizations have been applied on AES co-processor and tested on Xilinx Virtex-5 FPGA. Their security analyses reveal positive results. The masking countermeasure had an overhead of only 16% and was shown to be secure for the chosen model, thanks to the removal of barrel shifters. In the DPL countermeasure, the whole sequential part of AES that is also the main source of leakage was packed inside the BRAM with balanced placement by design. To our knowledge, the implementations proposed are the most compact of the state-of-the-art. Thus security is a another parameters which motivates integration of ample BRAM resources into FPGA chips.

Finally we would like to conclude that the security of a countermeasure depends on the specific leakage model of the device. Therefore it should be interesting to research formal methods to characterize leakage models for the given device.

As a perspective, we note that further tradeoffs are possible to explore in *low-entropy masking schemes* such as RSM. Typically, to avoid some bi-variate, multi-variate or collision attacks, more masks could be used (e.g., one per byte of the algorithm to protect). Another topic in this direction could be study the effect of using fewer values for the masks in the *leakage squeezing* masking scheme [Maghrebi et al. 2011].

REFERENCES

Altera. 2011. Stratix-II Device Handbook — Volume 1. (2011). http://www.altera.com/literature/hb/stx2/stratix2_handbook.pdf.

Shivam Bhasin, Sylvain Guilley, Annelie Heuser, and Jean-Luc Danger. 2013. From cryptography to hardware: analyzing and protecting embedded Xilinx BRAM for cryptographic applications. *J. Cryptographic Engineering* 3, 4 (2013), 213–225.

Shivam Bhasin, Sylvain Guilley, Youssef Souissi, Tarik Graba, and Jean-Luc Danger. 2011. Efficient Dual-Rail Implementations in FPGA using Block RAMs. In *ReConFig*. IEEE Computer Society, 261–267. Cancún, Quintana Roo, México. DOI: 10.1109/ReConFig.2011.32.

Sébastien Briais, Jean-Luc Danger, and Sylvain Guilley. 2013. A formal study of two physical countermeasures against side channel attacks. *J. Cryptographic Engineering* 3, 3 (2013), 169–180.

Éric Brier, Christophe Clavier, and Francis Olivier. 2004. Correlation Power Analysis with a Leakage Model, In CHES. *Proc. of CHES'04* 3156 (August 11–13 2004), 16–29. Cambridge, MA, USA.

Claude Carlet and Sylvain Guilley. 2013. Side-Channel Indistinguishability. In *HASP*. ACM, New York, NY, USA, Article 9, 8 pages. DOI: http://dx.doi.org/10.1145/2487726.2487735

Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. 1999. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO (LNCS)*, Vol. 1666. Springer. Santa Barbara, CA, USA. ISBN: 3-540-66347-9.

Saar Drimer, Tim Güneysu, and Christof Paar. 2008. DSPs, BRAMs and a Pinch of Logic: New Recipes for the AES on FPGAs. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 99–108. Stanford, Palo Alto, CA.

Louis Goubin and Jacques Patarin. 1999. DES and Differential Power Analysis. The "Duplication" Method. In *CHES (LNCS)*. Springer, 158–172. Worcester, MA, USA.

Tim Güneysu and Amir Moradi. 2011. Generic Side-Channel Countermeasures for Reconfigurable Devices. In *CHES (LNCS)*, Bart Preneel and Tsuyoshi Takagi (Eds.), Vol. 6917. Springer, 33–48.

Wei He, Andrés Otero, Eduardo de la Torre, and Teresa Riesgo. 2012. Automatic generation of identical routing pairs for FPGA implemented DPL logic. In *ReConFig*. IEEE, 1–6.

A. Samad Hedayat, Neil James Alexander Sloane, and John Stufken. 1999. *Orthogonal Arrays, Theory and Applications*. Springer, New York. 418 pages. ISBN 978-0-387-98766-8.

Houssem Maghrebi, Sylvain Guilley, and Jean-Luc Danger. 2011. Leakage Squeezing Countermeasure Against High-Order Attacks. In *WISTP (LNCS)*, Vol. 6633. Springer, 208–223. Heraklion, Greece. DOI: 10.1007/978-3-642-21040-2_14.

Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. 2005. Successfully Attacking Masked AES Hardware Implementations. In *Proceedings of CHES'05 (LNCS)*, LNCS (Ed.), Vol. 3659. Springer, 157–171. Edinburgh, Scotland, UK.

Marcel Medwed, François-Xavier Standaert, Johann Großschädl, and Francesco Regazzoni. 2010. Fresh Re-Keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices. In *AFRICACRYPT (LNCS)*, Vol. 6055. Springer, 279–296. Stellenbosch, South Africa. DOI: 10.1007/978-3-642-12678-9_17.

Amir Moradi. 2012. Statistical Tools Flavor Side-Channel Collision Attacks. In *EUROCRYPT (Lecture Notes in Computer Science)*, David Pointcheval and Thomas Johansson (Eds.), Vol. 7237. Springer, 428–445.

Maxime Nassar, Shivam Bhasin, Jean-Luc Danger, Guillaume Duc, and Sylvain Guilley. 2010. BCDL: A high performance balanced DPL with global precharge and without early-evaluation. In *DATE'10*. IEEE Computer Society, 849–854. Dresden, Germany.

Maxime Nassar, Sylvain Guilley, and Jean-Luc Danger. 2011. Formal Analysis of the Entropy / Security Trade-off in First-Order Masking Countermeasures against Side-Channel Attacks. In *INDOCRYPT (LNCS)*, Vol. 7107. Springer, 22–39. Chennai, Tamil Nadu, India. DOI: 10.1007/978-3-642-25578-6_4.

Maxime Nassar, Youssef Souissi, Sylvain Guilley, and Jean-Luc Danger. 2012. RSM: a Small and Fast Countermeasure for AES, Secure against First- and Second-order Zero-Offset SCAs. In *DATE*. IEEE Computer Society, 1173–1178. Dresden, Germany. (TRACK A: "Application Design", TOPIC A5: "Secure Systems").

Thomas Popp, Mario Kirschbaum, Thomas Zefferer, and Stefan Mangard. 2007. Evaluation of the Masked Logic Style MDPL on a Prototype Chip. In *CHES (LNCS)*, Vol. 4727. Springer, 81–94. Vienna, Austria.

Emmanuel Prouff and Matthieu Rivain. 2007. A Generic Method for Secure SBox Implementation. In *WISA (Lecture Notes in Computer Science)*, Sehun Kim, Moti Yung, and Hyung-Woo Lee (Eds.), Vol. 4867. Springer, 227–244.

Francesco Regazzoni, Yi Wang, and François-Xavier Standaert. 2011. FPGA Implementations of the AES Masked Against Power Analysis Attacks. In *COSADE*. 56–66. Darmstadt, Germany.

François-Xavier Standaert, Tal Malkin, and Moti Yung. 2009. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In *EUROCRYPT (LNCS)*, Vol. 5479. Springer, 443–461. Cologne, Germany.

Kris Tiri and Ingrid Verbauwhede. 2004. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *DATE'04*. IEEE Computer Society, 246–251. Paris, France. DOI: 10.1109/DATE.2004.1268856.

Rajesh Velegalati and Jens-Peter Kaps. 2010. Techniques to enable the use of Block RAMs on FPGAs with Dynamic and Differential Logic. In *International Conference on Electronics, Circuits, and Systems, ICECS 2010*. IEEE, 1251–1254.

Jason Waddle and David Wagner. 2004. Towards Efficient Second-Order Power Analysis. In *CHES (LNCS)*, Vol. 3156. Springer, 1–15. Cambridge, MA, USA.

Xilinx. 2011. Spartan-6 FPGA Block RAM Resources User Guide — UG383 (v1.5). (2011). http://www.xilinx.com/support/documentation/user_guides/ug383.pdf.

APPENDIX

A.1. Balanced Cell-based Dual-rail Logic (BCDL)

BCDL [Nassar et al. 2010] is a DPL countermeasure specially designed for securing FPGA-based implementation. The main advantage of BCDL comes from a global synchronization signal which we call P. It has been formally proven without glitches and free from EPE in [Briais et al. 2013]. A BCDL cell is divided into two stages, namely *precharge* and *evaluation* stages. Modern FPGAs like Virtex-5 possess $LUT6\_2$ which can implement a whole two input BCDL cell. BCDL is implemented using *bottom-up* approach where the main primitives of the algorithm are identified and secured. Thereafter the primitives in the reference design are replaced with secure primitives. Generally four BCDL primitives are deployed which are enough to implement modern crypto-algorithms like AES. A BCDL register and Sbox are earlier shown to be packed in a single BRAM (Sect 3.2). A BCDL multiplexer is made by doubling an unprotected multiplexer and ensuring that the selection signal should not be a data (dual-rail) signal. For constant activity, the dual multiplexer should process data of inverse polarity as the original. A BCDL XOR gate needs to be designed with care because it suffers the risk of glitches. To ensure glitch-free activity of BCDL XOR, the dual-outputs are described as:

$$O_T = (P \wedge \overline{I1_T} \wedge I1_F \wedge I2_T \wedge \overline{I2_F}) \vee (P \wedge I1_T \wedge \overline{I1_F} \wedge \overline{I2_T} \wedge I2_F)$$
$$O_F = (P \wedge \overline{I1_T} \wedge I1_F \wedge \overline{I2_T} \wedge I2_F) \vee (P \wedge I1_T \wedge \overline{I1_F} \wedge I2_T \wedge \overline{I2_F}),$$

where $I1$ and $I2$ are two dual rail inputs, O a dual-rail output and P the global precharge.