Hardware Virtualization for Protection Against Power Analysis Attack

Kai Yang, Jungmin Park, Mark Tehranipoor, and Swarup Bhunia
Florida Institute for Cybersecurity Research
University of Florida, Gainesville, Florida 32611
Email: {kyang84, jungminpark}@ufl.edu, {tehranipoor, swarup}@ece.ufl.edu

***Abstract*—Field programmable gate arrays (FPGAs) are being increasingly used in diverse Internet of Things (IoT) application space. Poor programmability of FPGAs compared to their processor counterparts remains an important challenge amidst their wide-spread usage. On the other hand, security of FPGA-based systems against physical attacks, in particular, side-channel attacks (SCAs) has emerged as a critical concern. Hardware virtualization, where instead of directly mapping a design to FPGA, it is mapped on top of a generic architecture, called *overlay*, has been shown to address the programmability challenge, leading to significantly higher productivity and several orders of magnitude reductions in compile time as well as bitstream size. However, unlike software or network virtualization, FPGA virtualization has not been studied with respect to its security benefits. In this paper, for the first time to our knowledge, we propose to utilize the properties of virtualization to address the FPGA security issues against a dominant mode of SCA, namely, power analysis attack. We note that while virtualization shows many intrinsic security benefits, we can efficiently implement masking approaches in novel ways onto this architecture to achieve high level of protection. Extensive security analysis is done to show large side-channel resistance improvement for a set of evaluation metrics.**

I. Introduction

FPGA market has grown significantly over the past decade with an estimated size of $6.36B USD in 2015, which is expected to nearly double by 2024 [1]. Such a compelling growth is due to rapidly increasing application space, which includes military and aerospace, consumer electronics, automotive systems, and the emergent IoT applications. The diversity of IoT devices, connection with often insecure networks, and use of less rigorous cryptographic measures make them a primary target for cyber attacks [2]. As billions of devices connect to Internet, malware through infected IoT devices can spread across the network, causing serious consequences such as extracting clients' secret information from cloud servers or denying to access servers.

As importantly, due to their long complex life, ease of physical access, and limited protection arising from tight resource constraints, they are vulnerable against various forms of physical attacks such as side-channel attacks. Adversaries could obtain sensitive values such as encryption keys by observing side-channel leakage information, such as power consumption, timing, or electromagnetic radiation. Among them, power analysis attacks have been studied extensively in both academia and semiconductor industry as a dominant form of SCA. Various power analysis attacks, such as, simple power analysis (SPA), differential power analysis (DPA), and correlation power analysis (CPA) have been developed to reveal the secret key from a crypto hardware module.

FPGAs are increasingly used in emerging applications e.g. in IoT and automotive systems, as they usually provide lower power, lower latency, and higher performance than processor-based implementations. They also provide higher flexibility than ASIC. Further, FPGA-based systems can be considered more trustworthy since it does not require leaking the mapped intellectual property (IP) to an untrusted foundry. However, FPGA devices are well-known to be hard to program than their processor counterparts leading to poor productivity. In the same time, they suffer from SCA vulnerability with power attack being a major attack vector. They could emanate side-channel leakages through timing, power and electromagnetic radiation. These leakages can be concealed or randomized by using hiding or masking technique. The masking method is used most frequently, but it generally poses overload problems such as dramatic slowdown and high area overhead.

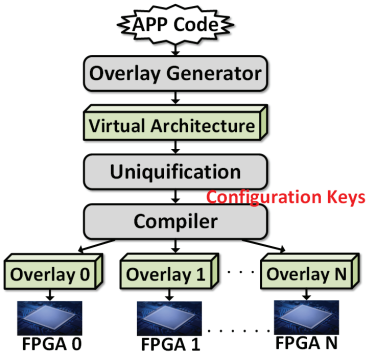

Fig. 1. Uniquified FPGA virtualization approach to achieve secure hardware.

Hardware or FPGA virtualization has emerged as an effective solution for FPGA programmability issue. Similar to software and network virtualization, FPGA virtualization hides the low-level physical resources in FPGA and provides an domain-matched generic layer, called ***overlay***, on top of which an application is mapped, instead of traditional direct mapping. While the productivity benefits for FPGA virtualization have been extensively studied in past decade, its security implication in terms of SCA has not been addressed.

We note that hardware virtualization using FPGA can provide effective protection against SCA at low overhead. Since the only cost of creating a new overlay is the time required for FPGA compilation, which has proved to be 10000x faster than FPGA-vendor tools [3], virtualization approach could achieve security via uniquification, which modifies the generated overlay with unique bitfiles for every deployed FPGA [4]. Fig. 1 illustrates the overall approach. In addition to improved security, overlays have a number of other advantages including simplified development and debugging, transparent high-level synthesis, bitfile portability across FPGAs, and 1,000x smaller bitfiles [3], which are critical for many IoT applications.

In this paper, we present a novel approach for protection against power analysis attacks by diversifying virtualized hardware across physical FPGAs and then developing special masking and hiding techniques to improve the resistance. In particular, the main contributions of this paper are as follows:

- It presents a hardware virtualization framework for reconfig-

978-1-5386-4731-8/18/$31.00 ©2018 IEEE

urable architecture and uses virtualization to simultaneously address FPGA productivity and security in terms of power analysis attacks. A unique overlay generator is used to generate the application-specific virtual architecture based on different design goals.

- It implements special masking and hiding techniques onto the overlay to greatly improve the security level against power analysis attacks.
- It evaluates the countermeasures with different SCA metrics such as test vector leakage assessment (TVLA) t-test statistic, Kullback-Leibler (KL) divergence, and success rate.
- It compares the security level and overhead with two other reference designs: directly mapping to FPGA, and mapping through overlay to FPGA using another masking method.

The rest of the paper is organized as follows. Section II presents the background and motivation for SCA countermeasures and hardware virtualization. Section III describes details of an example of virtualization architecture. Section IV introduces the proof of effectiveness of masking against power analysis attacks. Section V presents unique masking and hiding methods on the proposed architecture. Section VI describes three security evaluation metrics and the experiment setup. Section VII shows the results of different evaluation metrics. Finally, Section VIII concludes the paper.

II. Background and Motivation

In this Section, we provide background of power analysis attack and its general countermeasures, and then describe the effectiveness of the proposed overlay.

Correlation power analysis [5] is the most popular non-profiling attack for cryptographic hardware, where correlation between Hamming distance or weight model based on a guessing key and measured power traces caused by the secret key is analyzed. The guessed key with the maximum correlation is chosen as the secret key. Adversaries could acquire the guessed keys through data analysis based on large number of power traces, which are gathered by an oscilloscope.

Currently, there are two major countermeasures against power analysis attacks, hiding and masking. Hiding technique use different methods, such as power balancing, power isolation, and noise injection, to randomize the execution of the algorithm to reduce power leakage. However, these methods usually have very high area and performance overhead.

The idea of masking is to make the intermediate results of the cryptographic algorithm being executed independent of the secret key. Thus, attackers can no longer use the leakage information such as power traces to extract the encrypted key. The threshold implementation (TI) masking method was first proposed in [6]. Researchers have extensively studied this method, where all approaches require the use of random values in order to mask the data being processed. In order to implement nonlinear circuits, they require the introduction of additional fresh random values, which in turn requires very high overhead. Further, it is shown that many existing masking approaches are not secure against higher-order attacks [7].

Virtualized architecture can be an effective approach to address these shortcomings. The first reason is that overlay has very high scalability. The overlay generator could control the number of processing elements (PEs) in the compilation. The parallelism achieved through multiple PEs reduces the signal-to-noise ratio (SNR), and it can also be used to handle higher-order side-channel vulnerabilities. For example, targeting the n-th order SCA countermeasure, the number of PEs can be scaled to at least $n + 1$. Second, the overlay has very high flexibility. Each PE can be customized based on different design goals. The instruction set architecture (ISA), the instruction order of mapping, and the interconnect structure can vary from one PE to another in the same design to hide the leakage information.

III. MAHA Overlay

In this Section, we introduce a general overlay architecture, referred to as Malleable Hardware (MAHA), which we have developed for this work. Next, we describe MAHA architecture in detail, and present the major security advantages.

A. Hardware Architecture

Although the presented security advantages can potentially be gained from any overlay, we evaluate the approach using a modified MAHA overlay. It consists of a set of single-issue RISC-style processing elements. Each PE is referred to as a Memory Logic Block (MLB) and multiple PEs work independently. Different MLBs are connected through a multi-level hierarchical interconnect that balances bandwidth with scalability. We focus on power analysis attacks and map a 128-bit AES engine on the MAHA overlay to evaluate its security.

The microarchitecture of a single MLB is shown in Fig. 2. It contains common blocks such as datapath, memory, and schedule table. A custom datapath block supports common operations such as addition/subtraction and exclusive-or, all of which are common to the known applications. The schedule table holds all the instructions. A local register file holds the intermediate results from the application during processing. The data memory consists of high-density, 2D memory array, which holds the look-up table (LUT) results as well as the data being processed. Targeting AES application, we also use linear-feedback shift registers (LFSRs) to generate the random key and plaintext, and use the refresh module to update random values in the masking technique, which is described in Section V.

During program execution, MAHA fetches instructions from the schedule table. Depending on the instruction, the overlay will either select the datapath result or look-up table result for register write-back. The look-up table enables energy-efficient evaluation functions such as S-box and MixColumns in the AES encryption, while the lightweight datapath enables rapid execution of common functions for a domain. During compilation, instructions, data, and look-up table memory are spatially distributed among the MLBs, while local instructions within a given MLB execute temporally. The architecture is fully customizable for the target application, and can be implemented as an FPGA overlay using standard FPGA CAD tool flows.

B. Security Advantages

The MAHA overlay architecture has many advantages with respect to protection against side-channel attacks. Due to high flexibility and scalability, internal components such as the number of MLBs and the look-up table size could be customized. The parallelism using multiple MLBs could also reduce the SNR and hide the leakage information. Furthermore, in order to hide the power consumption characteristics, randomization techniques, such as ISA and instruction order randomization, are implemented within each MLB by making

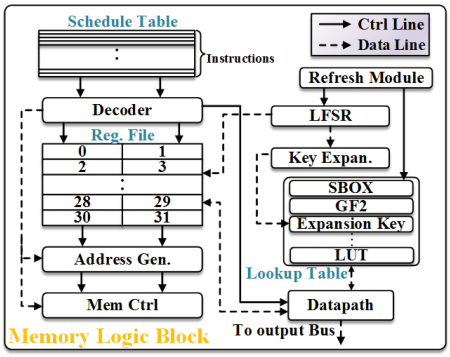

Fig. 2. Overview of the MLB architecture.

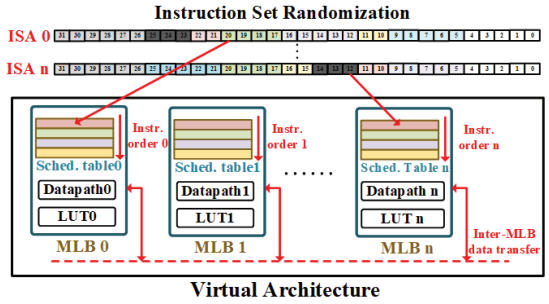

Fig. 3. MAHA overlay is highly customizable. Number of MLBs used is scalable, while every single MLB can have unique ISA and instruction order.

small modifications to the MLB structure. Fig. 3 illustrates the customization of MLBs.

On the other hand, the proposed architecture is suitable for typical masking method. Each shared input can be assigned to an individual MLB and multi-party computation can be executed in each MLB. In addition, its fast compilation feature [3] is helpful to refresh random values for masking at low cost. The detail of masking method for MAHA overlay is described in Section V.

IV. Masking Method

A masking method [8] suitable for the overlay architecture is introduced in this Section.

Definition 1 (Shared inputs): Each input x^j is split into $s+1$ shares: $x^j \rightarrow \bar{\mathbf{x}}^j = [x_0^j, x_1^j, \ldots, x_s^j]$. s random binary values, $r_{x_0}, r_{x_1}, \ldots, r_{x_{s-1}}$ are chosen for $x_0^j, x_1^j, \ldots, x_{s-1}^j$. And then x_s^j is encoded into $x_s^j = x^j \oplus x_0^j \oplus x_1^j \oplus \cdots \oplus x_{s-1}^j$.

Definition 2 (Secure random functions): The function f_i for $i = 0, \ldots, s-1$ is defined as the secure random function such that

$$z_i = f_i(\bar{\mathbf{x}}^1_{\mathbf{r}_1^i}, \ldots, \bar{\mathbf{x}}^p_{\mathbf{r}_p^i}) = \oplus_{k=1}^{p} \left(\oplus_{j \in \mathbf{r}_k^i} x_j^k \right) \quad (1)$$

, where $\bar{\mathbf{x}}^k_{\mathbf{r}_k^i}$ is a random vector consisting of randomly chosen variables from $\{x_0^k, \ldots, x_{s-1}^k\}$ such that $\mathbf{r}_k^i \subseteq \{0, \ldots, s-1\}$. Note that $\mathbf{r}_k^i$ is a random subset of $\{0, \ldots, s-1\}$.

Theorem 1: The secure random functions, f_i, have perfect secrecy with respect to the original inputs : $\Pr[z_i | x^1, \ldots, x^p] = \Pr[z_i]$.

Proof : x_j^k for $k = 1, \ldots, p$ and $j \in \mathbf{r}_k^i$ at Eq. (1) is a equally distributed random variable, and f_i for $i = 0, \ldots, s-1$ is a XOR function with the inputs of these random variables. Therefore,

$$\Pr[z_i | x_j^k] = \Pr[z_i] = 0.5.$$

And $\Pr[x^k | x_j^k] = \Pr[x^k]$ for $j \in \mathbf{r}_k^i$. With these equations,

$$\Pr[z_i | x^1, \ldots, x^p] = \Pr[z_i].$$

Definition 3 (Masking function): The function f_s is defined as the masking function such that

$$z_s = f_s(\bar{\mathbf{x}}^1, \ldots, \bar{\mathbf{x}}^p) = f \oplus_{i=1}^{s-1} f_i \quad (2)$$

Theorem 2: The masking function f_s has perfect secrecy with respect to the original inputs: $\Pr[z_s | x^1, \ldots, x^p] = \Pr[z_s]$.

The proof is presented in [8].

Definition 4 (The $s+1^{st}$-order masking): Let an original function with p inputs be $z = f(x^1, x^2, \ldots, x^p)$. Each input, x^k has s-random shares, $x_0^k, x_1^k, \ldots, x_{s-1}^k$, and an encoded bit, $x_s^k = x^k \oplus x_0^k \oplus x_1^k \oplus \cdots \oplus x_{s-1}^k$ for $k = 1, \ldots, p$. The $s+1^{st}$-order masking consists of s secure random functions and a masking function as follows:

$$z_i = f_i(\bar{\mathbf{x}}^1_{\mathbf{r}_1^i}, \ldots, \bar{\mathbf{x}}^p_{\mathbf{r}_p^i}) = \oplus_{k=1}^{p} \left(\oplus_{j \in \mathbf{r}_k^i} x_j^k \right) \quad \text{for} \quad i = 0, \ldots, s-1$$
$$z_s = f_s(\bar{\mathbf{x}}^1, \ldots, \bar{\mathbf{x}}^p) = f \oplus_{i=1}^{s-1} f_i$$

, where $\bar{\mathbf{x}}^k_{\mathbf{r}_k^i}$ is a random vector consisting of randomly chosen variables from $\{x_0^k, \ldots, x_{s-1}^k\}$ such that $\mathbf{r}_k^i \subseteq \{0, \ldots, s-1\}$. Note that $\mathbf{r}_k^i$ is a random subset of $\{0, \ldots, s-1\}$.

For example, if an AND function with 2 inputs, $z = f(x^1, x^2) = x^1 \cdot x^2$ is masked with 2 shares, x^1 and x^2 are encoded into $\bar{\mathbf{x}}^1 = [x_0^1, x_1^1]$ and $\bar{\mathbf{x}}^2 = [x_0^2, x_1^2]$, respectively. For a secure random functions, $[x_0^1, x_0^2]$ are randomly chosen as the inputs of f_0. By Eq. (1) and Eq. (2), 2 masked functions are the followings:

$$z_0 = x_0^1 \oplus x_0^2 \quad (3)$$
$$z_1 = (x_0^1 \oplus x_0^2) \oplus \left((x_0^1 \oplus x_1^1) \cdot (x_0^2 \oplus x_1^2) \right). \quad (4)$$

Note that the terms in Eq. (4) are not mapped into boolean gates but into a look-up table. This means that the original inputs x_0 and x_1 can not be exposed. A random function z_0 and the masking function z_1 has the perfect secrecy: $\Pr[z_0|x^1] = \Pr[z_0|x^2] = \Pr[z_0|x^1, x^2] = \Pr[z_0] = 0.5, \Pr[z_1|x^1] = \Pr[z_1|x^2] = \Pr[z_1|x^1, x^2] = \Pr[z_1] = 0.5$. In addition, the joint distribution of z_0 and z_1 is unbiased such that $\Pr[z_0, z_1] = c\Pr[\oplus_{i=0}^{1} z_i]$ based on Table I. This means that the average power consumption of each input pair $[x^1, x^2]$ is similar.

TABLE I. The number of cases of (z_0, z_1)

x^1	x^2	$z_0 z_1$ 00	01	10	11
0	0	2	0	0	2
0	1	2	0	0	2
1	0	2	0	0	2
1	1	0	2	2	0

The $s+1^{st}$-order masking provides two properties as Threshold Implementation for the provable security: correctness and uniformity. In terms of non-completeness, s random functions must be independent of at least one input share but the masking function f_s is dependent of all input shares. If f_s is implemented with AND gates and XOR gates, glitches as well as the intermediate output at the wires of AND gates can leak information about the original inputs [6]. In order to solve the problem, the f_s function as well as s random functions are mapped into look-up tables on FPGA.

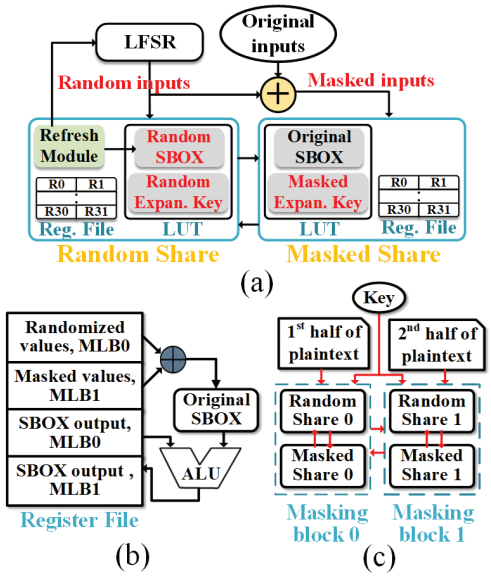

Fig. 4. (a) High level architecture of masking method using overlay on AES application; (b) Original SBOX input/output values are concealed; and (c) Combine masking and hiding technique using 4 MLBs.

V. Masking Implementation for Overlay

In this Section, the specific masking implementations using two MLBs and four MLBs are described. Fig. 4(a) shows the high level architecture of this masking method. The first MLB works as the random part while the second behaves as the masking part. Unlike conventional mapping that leverages plaintext and key as the input, we have concealed the original values using random values generated from LFSRs. These values are stored into the local register files and used as the plaintext and key for the first MLB, and they are XOR-ed with the original plaintext and key to generate the inputs for the second MLB. In this way, all the original inputs will not be stored in both MLBs. A refresh module is used to update the LFSR feedback, seeds, and random S-box values. In this way, the random inputs will always stay fresh, which makes the masking method effective.

Since S-box is a non-linear component and the most vulnerable part in the AES implementation, we focus on the masking of AES S-box. The AES S-box in the first MLB corresponds to random functions, $z_0^i = f_0^i(x_0^1, x_0^2, \ldots, x_0^8) = \oplus_{j \in \mathbf{r}_i} x_0^j$, where $\mathbf{r}_i$ is a randomly chosen subset of $\{1, 2, \ldots, 8\}$, for $i = 1, 2, \ldots, 8$. The random functions are mapped into the LUT with the 8-bit output and the 8-bit address corresponding to the input, $[x_0^1, x_0^2, \ldots, x_0^8]$. All possible number of the random look-up tables is $(2^8)^8 = 2^{64}$. Random LUTs can be updated by reconfiguring the bitfile on FPGA, which can increase the information entropy by 64 bits theoretically.

Masking functions for the AES S-box operation, $z_1^i = f_1^i(x_0^1, \ldots, x_0^8, x_1^1, \ldots, x_1^8) = f^i \oplus f_0^i$ for $i = 1, \ldots, 8$ are executed at the second MLB. If the masking functions are stored in the LUT with the 8-bit width and the 16-bit address, the LUT size would be $2^{16} \times 8 = 524,288$ bits, which is inefficient and impractical. In order to reduce the LUT usage, datapath is used to executes the masking function directly by bit-wise XORing the randomized value, $[z_0^1, z_0^2, \ldots, z_0^8]$ and the original S-box value, $[z^1, z^2, \ldots, z^8]$. The original S-box is mapped into the look-up table with the size of $2^8 \times 8 = 2,048$ bits at the compile time. The address of the look-up table is the same as the original input $[x^1, \ldots, x^8]$ which is split into the randomized value $[x_0^1, \ldots, x_0^8]$ at the first MLB and the masked value $[x_1^1, \ldots, x_1^8]$ at the second MLB. The address of the S-box look-up table can be obtained by XORing the randomized value, $[x_0^1, \ldots, x_0^8]$, which is transferred from the first MLB, and the masked value, $[x_1^1, \ldots, x_1^8]$. Note that the original input and output of the S-box are not stored in the local register file to prevent from generating side-channel leakage. The masking instruction takes two clock cycles, one for inter-MLB data transfer and the other for computing, and the overhead is moderate to obtain robust security. Fig. 4(b) shows the hardware structure for the masking function in the second MLB.

Another advantage of the overlay is that this masking method is scalable using more shared inputs. The first several MLBs perform as the random shares with different inputs and random S-box values, while the last MLB works for the masking part. As mentioned in Section III, we also implement hiding techniques such as parallelism and randomization onto the overlay, and combine the hiding techniques with the proposed masking method to further improve the security. The high level diagram of the combination method is shown in Fig. 4(c). First, AES instructions are divided into two parts and distributed into different masking blocks. Each masking block contains two MLBs, the random share and the masked share, and behaves similar to what is shown in Fig. 4(a). Original plaintext is also divided into two parts and sent to different masking blocks, separately. The entire key is used as input for both masking blocks, while the random key in each block is different. Intermediate values could be transferred between masking blocks for mapping purpose such as ShiftRow operations. After encryption, the 128-bit ciphertext is the combination of results from two masking blocks, 64 bits each.

VI. SCA Security Metrics and Experimental Setup

In this Section, we briefly describe three different security metrics, and introduce our experimental setup to collect power traces.

A. Evaluation Security Metrics

Test Vector Leakage Assessment methodology [9] is generally used to investigate the side-channel vulnerability for AES implementation, where two datasets with specific inputs are used to check if they are distinguishable to each other. While it could check if the design has vulnerability, it cannot estimate how much SCA resistance the implementation has. And KL divergence [10] could quantify side-channel leakage between two sets. Lastly, the real side-channel attacks are implemented onto MAHA overlay, where success rate is calculated based on the total number of attacks and the successful time.

B. Experimental Setup

In this experiment, we use the SAKURA-G board, which contains two SPARTAN-6 FPGAs to implement the hardware design and to communicate between the implementation and the PC. Tektronix MDO3102 oscilloscope is used to measure the voltage drop between 1Ω shunt register connected to the Vdd pin. The clock frequency of the MAHA overlay is 24 MHz and the sampling rate and bandwidth of the oscilloscope are 100 MS/s and 250 MHz, respectively.

VII. Results and Analysis

In this Section, we arrive at the results using different metrics, then evaluate the side-channel attack resistance and performance for different implementations.

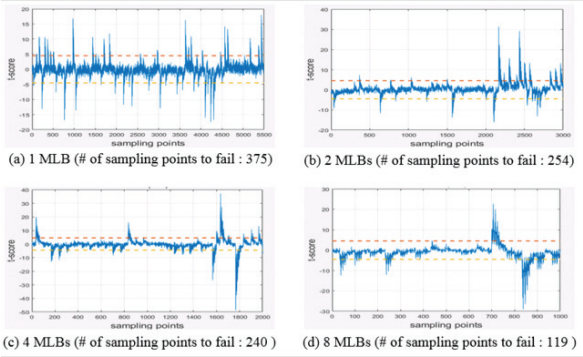

Fig. 5. T-test results of non-masking implementations using different number of MLBs.

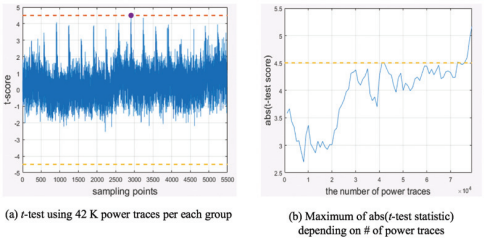

Fig. 6. 1^{st}-order t-test results of masking implementation with 2 MLBs without refreshing the random S-box value and LFSR's seed.

A. TVLA t-test

First, we evaluate only the hiding method using different number of MLBs, ranging from 1, 2, 4 to 8, in MAHA overlay. Note that the datapath of a single MLB is 8 bits thus the overlay architecture with n MLBs has $n \times 8$-bit datapath. Fig. 5 shows the t-test results with respect to different number of MLBs without masking method. 10K power traces are collected for each group in the t-test. We can discover that t-test results get closer to the middle, and the number of sampling points to fail in the test decreases, as the number of MLBs used increases. But still, all implementations without masking method failed in t-test. The parallelism can reduces the SNR, but merely use of hiding method cannot prevent power analysis attacks.

Then, we evaluate the first-order and second-order masking implementations on the MAHA overlay which consists of 2 MLBs and 3 MLBs, respectively. If the random S-box LUTs are not refreshed or updated properly, the masking implementation cannot pass the t-test even if the plaintext and key are masked with random bits generated by LFSRs. The seeds in the LFSRs also should be refreshed regularly in order to increase the information entropy. Without refreshing the random S-box values and the seeds, the t-test with 42K power traces per group is failed. Fig. 6(a) shows the t-test result using 42K power traces each group, and the maximum absolute value of t-test score is shown in Fig. 6(b), where the number of power traces in each group ranges from 1K to 80K without refreshing. To solve this problem, we add a refresh module in each MLB to update the random S-box values and the LFSR seeds every 40K encryptions. With this method, implementations using 2 MLBs and 3 MLBs pass the first-order and second-order t-test using 200K power traces each group.

Finally, we evaluate the implementation combines the masking and hiding techniques using 2x2 MLBs. It also passes the first-order t-test. Fig. 7 shows the first-order and the second-order t-test results

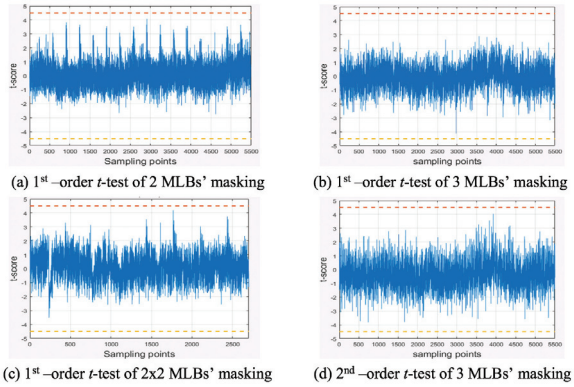

Fig. 7. 1^{st} and 2^{nd}-order t-test results of various masking implementations.

TABLE II. KL DIVERGENCE OF MASKING IMPLEMENTATIONS

	2MLBs	3 MLBs	2x2 MLBs
Max	4.79×10^{-4}	1.7×10^{-3}	4.33×10^{-4}
Mean	5.11×10^{-5}	6.54×10^{-5}	6.01×10^{-5}
Min	8.67×10^{-5}	1.25×10^{-5}	3.03×10^{-8}

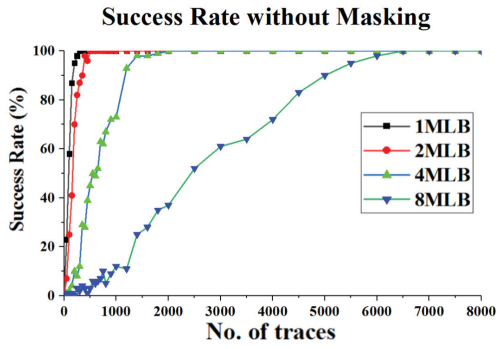

Fig. 8. Success rate for implementations without masking method.

of different implementations, where 200K power traces are used in each group.

B. Kullback-Leibler Divergnece

In order to compare masking implementations in terms of side-channel leakage, KL divergence between two different groups is exploited. Table II shows the maximum, mean and minimum KL divergence of three different masking implementations. The differences between these designs are very small, while 3MLBs' masking implementation has the smallest leakage and 2MLBs' masking implementation has the largest, based on the mean of KL divergence.

C. Success Rate

As discussed, we apply CPA attack for implementations with and without masking method incorporated. For the masking part, we focus on both the first-order and second-order CPA attacks using 2 MLBs, and 3MLBs, respectively. For the non-masking part, we use 1, 2, 4, and 8 MLBs to generate the results. Fig. 8 shows the success rate for implementations without the masking method. All implementation results could reach 100% once the sample size is large enough, while the minimum number of traces needed for 100%

TABLE III. OVERHEAD OF IMPLEMENTATIONS WITH AND WITHOUT MASKING METHOD USING DIFFERENT NUMBER OF MLBS

	No. of Slice Reg	No. of Slice LUTs	No. of Cycles	Bitstream Size (Byte)	Total Power (mW)	No. of Block RAMs
Non-masking, 2 MLBs	2757	3233	247	4792	930.38	5
Masking, 2 MLBs	4520	4226	345	5576	943.43	6
Non-masking, 4 MLBs	5028	5923	125	7632	947.39	10
Masking, 4 MLBs	7317	7172	184	8576	956.49	11
Non-masking, 8 MLBs	9853	11209	74	12608	966.53	18

TABLE IV. OVERHEAD AND PERFORMANCE OF DIFFERENT MAPPING TECHNIQUES

	No. of Slice Reg	No. of Slice LUTs	No. of Cycles	Bitstream Size (Byte)	Total Power (mW)	No. of Block RAMs
Direct mapping, without masking method	1307	1206	169	4.5M	932.64	2
Through overlay, with our masking method	4520	4226	345	5576	943.43	6
Through overlay, with TI masking method	7154	6836	345	8236	967.57	10

success rate increases as the MLB number grows. For the masking part, both the first-order and second-order CPA attacks cannot reveal the secret key with 200K power traces for implementations using 2 MLBs and 3 MLBs, respectively. Besides, the first-order CPA attack for 2x2 MLBs' masking implementation combined with the hiding technique is also failed.

D. Overhead analysis

In this part, we evaluate the overhead of the overlay and proposed masking methods. We first obtain a baseline comparison between implementations without masking methods using various number of MLBs in order to see how the parallelism affects the area and performance overhead. Table III lists the area and performance overhead with 2, 4, and 8 MLBs. Since each MLB uses local LUT and register file to hold intermediate values and results, the number of registers/LUTs, and the bitstream size used are approximately positive correlated with number of MLBs. On the contrary, the number of cycles reduces as we increases the parallelism. The total energy is also reduced as the power consumption and clock frequency are similar for different designs.

Then, we compare the overhead between implementations with and without masking method incorporated. Table III lists the area, power, and execution time for 2 MLBs and 4 MLBs with and without masking method. The additional instructions such as masking functions (Fig. 4(b)) and inter-MLB data transfers increase the total number of cycles, by 1.4x on average. And the extra computation logic increases the number of registers and LUTs by 1.5x and 1.4x, respectively.

Finally, we compare the overhead of proposed overlay using masking method with two other mapping techniques: (1) Mapping AES directly on FPGA without masking method, and (2) Mapping AES through MAHA overlay using TI masking methods. Table IV lists the overhead and performance for direct mapping, mapping with TI masking method, and mapping with the proposed masking method. Comparing with direct mapping, the MAHA overlay architecture could achieves high resistance against power analysis attacks at reasonable overhead. At the same time, since the minimum number of MLBs used in the TI masking method is three, while our method only needs two, the proposed masking method has much less overhead compared with TI masking method.

VIII. CONCLUSION

We have presented a flexible FPGA overlay which is amenable to architectural diversity for the purpose of security against power analysis attacks, while simultaneously addressing the productivity challenges. Through extensive security analysis and comparative evaluation using a set of metrics, we have shown that the proposed approach makes it more difficult for an adversary to obtain secret information through power side-channel leakage. We have presented a new overlay, called MAHA, as well as a custom overlay generator based on MAHA to create different countermeasures for the power analysis attack. Unique masking and hiding techniques are also implemented onto the overlay. The results show large improvement in the side-channel attack resistance at much lower area and performance overhead compared to direct mapping to FPGA. While this paper shows the improvement of side channel attack resistance on power analysis attack, it also creates several new research pathways. Future research efforts can be directed toward extending the MAHA architecture for other side-channel attacks such as timing attack and fault injection attack.

REFERENCES

[1] G. V. Research, "Field Programmable Gate Array (FPGA) Market Analysis," *http://www.grandviewresearch.com/industry-analysis/fpga-market*, 2016.

[2] M. Aman, K. C. Chua, and B. Sikdar, "Mutual Authentication in IoT Systems Using Physical Unclonable Functions," *IEEE Internet of Things Journal*, 2017.

[3] J. Coole and G. Stitt, "Adjustable-Cost Overlays for Runtime Compilation," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*. IEEE, 2015, pp. 21–24.

[4] G. Stitt, R. Karam, K. Yang, and S. Bhunia, "A Uniquified Virtualization Approach to Hardware Security," *IEEE Embedded Systems Letters*, 2017.

[5] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 16–29.

[6] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold Implementations Against Side-Channel Attacks and Glitches," in *International Conference on Information and Communications Security*. Springer, 2006, pp. 529–545.

[7] E. Oswald, S. Mangard, C. Herbst, and S. Tillich, "Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers," in *Cryptographers Track at the RSA Conference*. Springer, 2006, pp. 192–207.

[8] J. Park and A. Tyagi, "T-Private Logic Synthesis on FPGAs," in *Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 63–68.

[9] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A Testing Methodology for Side-Channel Resistance Validation," in *NIST non-invasive attack testing workshop*, 2011, pp. 158–172.

[10] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.