Built-in Self-Evaluation of First-Order Power Side-Channel Leakage for FPGAs

Ognjen Glamočanin*, Louis Coulon*, Francesco Regazzoni† and Mirjana Stojilović*
*École polytechnique fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland
†ALaRI, Università della Svizzera italiana, CH-6904 Lugano, Switzerland

ABSTRACT

Embedded and cyber-physical systems are pervading all aspects of our lives, including sensitive and critical ones. As a result, they are an alluring target for cyber attacks. These systems, whose implementation is often based on reconfigurable hardware, are typically deployed in places accessible to attackers. Therefore, they require protection against tampering and side-channel attacks. However, a side-channel resistant implementation of a security primitive is not sufficient, as it can be weakened by an adversary, aging, or environmental factors. To detect this, legitimate users should be able to evaluate the side-channel resistance of their systems not only when deploying them for the first time, but also during their entire service life. The most widespread and *de facto* standard methodology for measuring power side-channel leakage uses Welch's t-test. In practice, collecting the data for the t-test requires physical access to the device, a device-specific test setup, and the equipment for measuring the power consumption during device operation. Consequently, only a small number of cyber-physical systems deployed in the field can be tested this way and the tests to reevaluate the device resistance to side-channel attacks cannot be easily repeated. To address these issues, we present a design and an FPGA implementation of a built-in test for self-evaluation of the resistance to first-order power side-channel attacks. Once our test is triggered, the FPGA measures its own internal power-supply voltage and computes the t-test statistic in real time. Experimental results on two different implementations of the AES-128 algorithm demonstrate that the self-evaluation test is very reliable. We believe that this work is an important step towards the development of security sensors for the next generation of safe and robust cyber-physical systems.

CCS CONCEPTS

• **Hardware** → **Reconfigurable logic and FPGAs**; • **Security and privacy** → *Tamper-proof and tamper-resistant designs*; *Side-channel analysis and countermeasures.*

ACM Reference Format:
Ognjen Glamočanin*, Louis Coulon*, Francesco Regazzoni† and Mirjana Stojilović*. 2020. Built-in Self-Evaluation of First-Order Power Side-Channel Leakage for FPGAs. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '20), February 23–25, 2020, Seaside, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3373087.3375318

1 INTRODUCTION

The pervasive diffusion of embedded and cyber-physical systems in automotive, industry 4.0, healthcare, power grid, and all other aspects of our lives imposes new challenges for system developers. The criticality of the applications and the sensitivity of data require the use of appropriate security primitives everywhere. The deployment of these devices in hostile environments, often accessible to adversaries, calls for side-channel attack countermeasures and the ability of detecting unwanted tampering. Finally, the service life of these devices—typically longer than that of consumer electronic devices—requires the ability to deal with device performance degradation, which can affect its resilience to side-channel attacks.

To detect tampering, some devices have shields and sensors that detect changes in light, temperature, or even attempts to remove mechanical protections. However, these methods cannot be used to detect other tampering attempts [8] or changes due to aging or device maintenance. Ensuring that a security primitive works as expected requires monitoring during its operation, preferably directly on the host device. For instance, one can monitor the statistical properties of random number generators [17] or samplers in lattice-based constructions [6]; if they are not as expected, one can suspect that the device has suffered from tampering or malfunctioning.

Despite these initial and successful attempts, the development of on-chip sensors for real-time monitoring of security primitives is still a largely unexplored area of research. A security problem that would certainly benefit from the use of real-time on-device sensing is that of physical side-channel attacks, in which the adversary exploits the weaknesses of the device implementation to extract secret information from it. Among them, power side-channel attacks, which exploit the information unintentionally leaked through the power consumed by the device, have proven to be very effective on CPUs and FPGAs [7, 18]. Moreover, successful attempts of tampering with the devices have shown that the information leakage can be increased in order to deploy these power side-channel attacks more easily. Schnelleberg et al. demonstrated that removing decoupling capacitors, which ensure that local voltage variations are filtered at the board level, results in 100× less measurements needed to break the secret key [11].

Our threat model targets embedded/cyber-physical systems often accessible to adversaries, which may be subject to aging, malfunctioning, or tampering with the intent to make the device leak a higher-than-desired amount of information via the power side channel. A large body of research is devoted to countermeasures against power side-channel attacks. However, their effectiveness is currently only verified before the device is deployed and almost never reevaluated, which makes it impossible to detect an increase in power side-channel leakage due to malfunctioning or tampering.

In this paper, we describe and design a standalone test that performs on-chip voltage measurements and evaluates the susceptibility to power analysis attacks of an FPGA implementation of a cryptographic core. We assess the test accuracy using the AES-128 algorithm as a case study and compare the results with those obtained using standard in-lab testing procedures and equipment. The *de facto* standard method for estimating power side-channel leakage is Welch's t-test [5]; it requires measuring the power-supply voltage during a number of encryptions and computing the t-test statistic over the acquired data. To measure the FPGA power-supply voltage as frequently as possible, we employ sensors directly implemented in the FPGA fabric. Our test hardware computes the first-order t-test statistic and is calibrated on the target cryptographic core. Although verified using AES, the methodology we propose is general and can be applied to any cryptographic core. Moreover, our built-in test can be triggered remotely, periodically, or on power-up, thus providing continuous monitoring of the device susceptibility to the first-order correlation and differential power analysis attacks during its entire service life.

In the remainder of the paper, we first discuss previous work. Then, in Section 3, we introduce the t-test leakage-estimation methodology and describe how we adapt it for computing the t-test metric in hardware. Section 4 describes the architecture and FPGA implementation of our system. Section 5 presents and discusses the experimental results, while Section 6 concludes the paper.

2 STATE OF THE ART

Most of the previous work on the monitoring of security primitives targets resistance against fault attacks or tolerance to errors. The most natural example of such monitors is the error correction circuitry included into hardware and software implementations of cryptographic primitives. A typical case is the use of parity codes applied to the AES algorithm [2]. Yet, albeit adapted to security primitives, this monitoring of the functionality is still an application of classical error correcting codes.

The first security primitive that has been monitored with a dedicated hardware watchdog is a true random number generator (TRNG): to ensure that the sequences produced by TRNGs respect strict statistical properties, Yang et al. [17] designed on-the-fly statistical tests suitable for hardware implementation on FPGAs. Checking the statistical properties of the results has also been proposed to counteract fault attacks on other cryptographic primitives, such as lattice based ones: Howe et al. [6] use a battery of statistical tests to verify if the distribution produced by the sampler is the expected one (Gaussian or binomial). These statistical tests have been designed for FPGAs and included as hardware monitors into the samplers in lattice based algorithms.

While certainly useful, these monitors focus only on *active* attacks. However, evaluating the leakage exploitable with *passive* attacks using monitors is a task largely unexplored in literature. The most relevant work related to our contribution is probably the one of Sonar et al. [13]: they implement a hardware watchdog to measure the side-channel leakage of a cryptographic primitive. Our work extends their initial ideas, providing various novel contributions: instead of using a model of leakage, which requires training and does not accurately model unexpected changes in the environment, we measure the actual leakage from the device (we design a sensor and integrate it into the user design as a part of our monitor). Thus, we replace the training phase with a simple calibration procedure, which immediately tailors our monitor to different cryptographic algorithms and implementations. By performing the calculation on real power traces, we remove the strong assumptions of a constant mean and a negligible variance of the fixed plaintext traces used for the t-test, which could lead to wrong results. Finally, we improve the computation of the t-test statistic, since we measure leakage in multiple samples of a trace instead of in only one.

3 HARDWARE-FRIENDLY APPROACH TO COMPUTING T-TEST STATISTIC

One of the fundamental questions in many scientific fields is whether two datasets are significantly different from each other. An answer to it can be obtained using Welch's t-test, in which the test statistic follows a Student's t-distribution [5, 12]. In cryptography, the non-specific t-test is commonly used to evaluate first-order side-channel leakage. To compute it, two sets of power traces need to be collected: one while a constant plaintext is encrypted (Q_F) and the other while randomly chosen plaintexts are encrypted (Q_R). Additionally, the decision whether to encrypt the fixed or random plaintext must be made in a nondeterministic or a randomly-interleaved fashion, to avoid predictable initial conditions. The t-test statistic can then computed as

$$t = \frac{\mu_F - \mu_R}{\sqrt{\frac{s_F^2}{n_F} + \frac{s_R^2}{n_R}}}, \tag{1}$$

where μ_F (resp. μ_R) is the sample mean and s_F^2 (resp. s_R^2) the sample variance of the set Q_F (resp. Q_R), and n_F (resp. n_R) is the cardinality of the set Q_F (resp. Q_R). If $|t|$ exceeds the threshold of 4.5, the test has detected a leakage with a confidence of at least 0.99999 [12]. If the power traces contain multiple samples, the test needs to be repeated for every one of them.

The cardinality of the trace sets being in the order of tens of thousands and more, a straightforward implementation of the t-test computation in hardware would result in very low performance and in an excessive overhead in terms of resources and power consumption. Hence, the statistic must be computed online. Sonar et al. [13] propose recomputing the sample mean and variance with every new recorded trace. However, we will demonstrate that this frequent computation is inefficient and not needed to achieve correct t-test results. Instead, we propose a method that performs minimal computation when a new trace is collected and recomputes the statistic parameters only after a batch of k power consumption traces is collected. The parameter k is defined by the user, according to the needs of the target application.

3.1 Online Computation of Mean and Variance

For a set of traces X of cardinality N, where each trace has M samples, the mean and the variance of a trace sample x over all N

traces are defined as follows:

$$\mu = \mu_N = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{2}$$

$$s_N^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_N)^2. \tag{3}$$

Here, i is the index of the trace in the set $\mathbf{X}$. The above equations are called *standard two-pass algorithm* [3], as they require traversing the data twice: once to compute the mean and second time to compute the variance. This is impractical, because the number of traces (the dataset size) can be prohibitively large, thus requiring an unacceptable amount of memory and computational overhead to store and access all the data. A standard practice to avoid the two-pass nature of expressions in (2) and (3) is to manipulate the expression for the variance into a *one-pass* computation as

$$s_N^2 = \sum_{i=1}^{N} x_i^2 - \frac{1}{N}\left(\sum_{i=1}^{N} x_i\right)^2. \tag{4}$$

However, this form is numerically unstable when implemented in floating-point arithmetic. In fact, it is susceptible to large cancellation errors or even a negative estimated variance. The instability can be remedied by using an *online* method [3], for instance Welford's algorithm [15], in which the statistics are recomputed with every new sample that arrives:

$$\mu_{n+1} = \mu_n + \frac{1}{n+1}(x_{n+1} - \mu_n), \tag{5}$$

$$s_{n+1}^2 = \frac{1}{n+1}\left(s_n^2 + (x_{n+1} - \mu_{n+1})(x_{n+1} - \mu_n)\right). \tag{6}$$

The disadvantage of (6) is that the new mean needs to be ready before the update of the variance can take place. Schneider et al. address this by first introducing the term central sum

$$CS_{d,n} = \sum_{i=1}^{n}(x_i - \mu_n)^d \tag{7}$$

and then deriving the formula for updating it incrementally [12]:

$$CS_{d,n+1} = CS_{d,n} + \sum_{m=1}^{d-2}\binom{d}{m} CS_{d-m,n}\left(\frac{\mu_n - x_{n+1}}{n}\right)^m + \\ + \left(\frac{n-1}{n}(\mu_n - x_{n+1})\right)^d \left[1 - \left(\frac{-1}{n-1}\right)^{d-1}\right]. \tag{8}$$

Finally, the variance is only one scaling factor away:

$$s_{n+1}^2 = \frac{1}{n+1} CS_{d,n+1}. \tag{9}$$

Let us introduce the term *partial central sum*: $CS_d(n,k)$. It corresponds to the central sum computed over k samples in the range $(n+1, \ldots, n+k)$:

$$CS_d(n,k) = \sum_{i=n+1}^{n+k}(x_i - \mu_n)^d. \tag{10}$$

From (5), (6), (9), and (10), after several transformations, we arrive to the expressions that allow us to control the frequency of the updates to once every k new traces are recorded:

$$\mu_{n+k} = \mu_n + \frac{1}{n+k} CS_1(n,k), \tag{11}$$

$$s_{n+k}^2 = \frac{n}{n+k}s_n^2 + \frac{1}{n+k}CS_2(n,k) - \left(\frac{1}{n+k}CS_1(n,k)\right)^2. \tag{12}$$

The above equations remove the constraint of having to recompute the statistics after every new recorded trace, which was the case in the work by Sonar et al. [13]. This, in turn, reduces the computation time while, as we will demonstrate later, it does not deteriorate the accuracy of the t-test metric.

Finally, the expressions in (10), (11), and (12) are very hardware friendly: the only values that need to be saved on chip, besides the running mean and variance, are the two partial central sums.

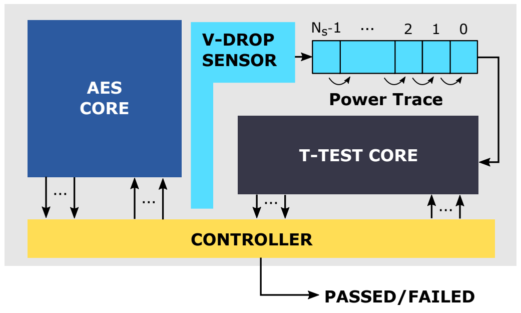

Figure 1: System block architecture composed by a voltage-drop sensor, the t-test side-channel leakage estimation core, a trace delay buffer and a controller. The AES cryptographic core under test is also reported in the figure.

3.2 Online Computation of t-test Statistics

To avoid computing square root, similar to Sonar et al. [13], we use the squared t-test statistic and an equal number N of random- and fixed-plaintext traces:

$$t^2 = \frac{(\mu_R - \mu_F)^2}{\frac{1}{N}\left(s_R^2 + s_F^2\right)}. \tag{13}$$

As previously discussed, if the t value exceeds the threshold $|t| > 4.5$, the test has detected leakage with a confidence of at least 0.99999 [12]. Hence, we implement the following comparison:

$$N(\mu_R - \mu_F)^2 > 4.5^2\left(s_R^2 + s_F^2\right) \tag{14}$$

Power trace samples for which this comparison returns logical 1 are considered *leaky*.

4 FPGA DESIGN AND IMPLEMENTATION

The block architecture of our leakage-evaluation circuit is illustrated in Fig. 1. It comprises a highly-sensitive digital sensor for on-chip power-supply voltage measurements, a memory buffer to temporarily store a power trace, the t-test core, and a controller, besides the cryptographic core under evaluation (in our case study, an AES-128).

4.1 Algorithm for Leakage Evaluation

Our system for online self-evaluation of the first-order power side-channel leakage follows the steps of the function `BuiltInTest`, shown in Algorithm 1. Function parameters that influence the test duration are E_{MAX}, the desired maximum number of encryptions to run during the test, k, the rate at which t-test statistic is updated, and N_S, the number of data samples per power trace. The higher the value of k, the smaller the number of operations that the built-in test needs to perform and, consequently, the shorter the time for the test to complete. N_S is the number of sensor samples recorded per every encryption of a plaintext. Its value depends on the frequencies at which the cryptographic core and the sensor are running and N_E, the number of cycles need to perform the entire encryption. The valid range for N_S is from one sample to at most $\lfloor N_E \times T_S / T_E \rfloor$, where T_S and T_E are the clock periods of the sensor and the cryptocraphic core, respectively.

The algorithm has two phases. During the first phase, using the test-only key *KeyT*, the system runs k encryptions of the fixed plaintext $PlainT_{\text{F}}$, k encryptions of the random plaintext, and $2 \times k \times N_S$ online updates of the first and the second partial central sums in (10) (for each sample of the corresponding trace). During encryption, the t-test core is disabled, minimizing any effects its power consumption may have on the measurements. During the second phase, the t-test core updates the mean, the variance, and the t-test statistic for every power-trace sample. For simplicity, the random plaintext of the current encryption is the ciphertext from the previous encryption. Before every encryption, the cryptographic core is reset, to ensure constant and data-independent initial conditions. The end condition (whether the device passed or failed the test) depends on the desired use case of the system. For instance, one may want to signal that there is a problem as soon as a leaky sample is detected; this is the scenario illustrated in Algorithm 1. In another situation, one may want to be less conservative and allow a small, but limited, number of leaky samples during the test, and report failure only if that number is exceeded by too large a value. In yet another use case, one may want to compare the number of leaky samples between different test runs, to detect if the board has undergone changes or tampering. Whatever the target deployment scenario, the change to be made to the algorithm (and, consequently, the system implementation) remains minimal.

Algorithm 1: Function `BuiltInTest` that checks if the first-order power side-channel leakage is present.

```
Input: $E_{\text{MAX}}$: number of encryptions during which no
       first-order power side-channel leakage is allowed
Input: $k$: size of the batch of traces
Input: $N_S$: number of sensor samples per power trace
Input: $KeyT$: test key, for use only during built-in testing
Input: $PlainT_{\text{F}}$: fixed plain-text input
Input: $PlainT_0$: initial plain-text input
Variables: TRACES, MEANS, CS1P, CS2P, VARS: memories
           for keeping power-supply traces, means, partial
           first central sums, partial second central sums, and
           variances
Output: Passed or Failed
$PlainT_R \leftarrow PlainT_0$
$ttest.clock.disable()$
$t = 0$
while $t < E_{\text{MAX}}$ do
    CS1P.$clear()$
    CS2P.$clear()$
    for $e \leftarrow 1$ to $k$ do
        /* Encrypt fixed plaint-text.                */
        $AES.reset()$
        $CipherT_{\text{F}} \leftarrow AES.encrypt(PlainT_{\text{F}}, KeyT)$
        $trace \leftarrow load(sensor(), N_S)$
        /* Update partial central sums.              */
        $ttest.clock.enable()$
        foreach $trace\ sample\ i,\ i \in [0..N_S - 1]$ do
            $x_F \leftarrow CS1(trace(i),$ MEANS$(2i),$ CS1P$(2i))$
            $y_F \leftarrow CS2(trace(i),$ MEANS$(2i),$ CS2P$(2i))$
            CS1P.$store(x_F, 2i)$
            CS2P.$store(y_F, 2i)$
        $ttest.clock.disable()$
        /* Encrypt random plaint-text.               */
        $AES.reset()$
        $CipherT_{\text{R}} \leftarrow AES.encrypt(PlainT_R, KeyT)$
        $trace \leftarrow load(sensor(), N_S)$
        $PlainT_R \leftarrow CipherT_{\text{R}}$
        /* Update partial central sums.              */
        $ttest.clock.enable()$
        foreach $trace\ sample\ i,\ i \in [0..N_S - 1]$ do
            $x_R \leftarrow CS1(trace(i),$ MEANS$(2i+1),$ CS1P$(2i+1))$
            $y_R \leftarrow CS2(trace(i),$ MEANS$(2i+1),$ CS2P$(2i+1))$
            CS1P.$store(x_R, 2i+1)$
            CS2P.$store(y_R, 2i+1)$
        $ttest.clock.disable()$
    /* Update mean, variance, and t-test.            */
    $ttest.clock.enable()$
    foreach $trace\ sample\ i,\ i \in [0..N_S - 1]$ do
        $m_F \leftarrow mean($MEANS$(2i),$ CS1P$(2i))$
        $m_R \leftarrow mean($MEANS$(2i+1),$ CS1P$(2i+1))$
        MEANS.$store(m_F, m_R, 2i, 2i+1)$
        $s_F \leftarrow var($VARS$(2i),$ CS1P$(2i),$ CS2P$(2i))$
        $s_R \leftarrow var($VARS$(2i+1),$ CS1P$(2i+1),$ CS2P$(2i+1))$
        VARS.$store(s_F, s_R, 2i, 2i+1)$
        $ttSuccess \leftarrow ttest(m_F, m_R, s_F, s_R, t)$
        if $ttSuccess = False$ then
            return Failed
    $ttest.clock.disable()$
    $t \leftarrow t + k$
return Passed
```

4.2 Controller

The controller is responsible for triggering the encryption, disabling the t-test core to prevent any influence on the sensor measurements, enabling the t-test core, monitoring the test results, making decisions whether the leakage test passed or failed, and taking appropriate actions.

4.3 Power-Supply Voltage Sensor

Even though FPGAs contain embedded system monitors, these on-chip measurement circuits have a relatively low sample rate and cannot detect nanosecond voltage fluctuations caused by logic switching at high frequency [19]. Therefore, to capture on-chip power-supply voltage variations, we implement a voltage-fluctuation sensor directly on the FPGA. Two sensor designs have been proposed so far, both indirectly monitoring FPGA power-supply voltage:

delay-line based sensors [19] and RO-based sensors [14, 19]. We opted for the former, because it can capture voltage variations in time intervals as short as few ns. The measurement resolution of the delay-line sensors is almost constant and corresponds to the delay of a single level of combinational logic in the carry-chain FPGA primitives. In comparison, RO-based sensors need considerably longer time to produce a value: they count the number of RO oscillations during a fixed time. As a consequence, their measurement resolution depends on the measurement duration; the longer the time to obtain a power-trace sample, the better the resolution and accuracy.

4.4 T-test Core

The t-test core contains RAM memory for keeping the first-order and the second-order partial sums in (10), the running means in (11) and variances in (12), and t-test scores for all N_S power-trace samples. Due to the pipelined implementation of the core, the traces processed by our system can contain an arbitrarily large number of samples, without increasing the datapath resource utilization. The multiplicative factors $\frac{1}{n+k}$ and $n+k$ are computed offline and saved in look-up tables. The t-test computation can be performed in floating-point or fixed-point arithmetic, depending on the resources offered by the host FPGA. Our test platform, Sasebo-GII [4], is equipped with a Xilinx Virtex-5 LX30 FPGA. This FPGA does not have hardened floating-point units, but it does have DSP blocks, which we harness for all the required arithmetic operations. Conversion from floating-point to fixed-point representation leads to precision loss and imposes the challenge of finding the right number of bits to represent the integer and the fractional parts of each intermediate variable in the t-test computation. The DSP blocks in the Virtex-5 FPGA [16] perform addition/subtraction on two 48-bit wide operands, which enables a minimal loss in precision. However, the DSP multipliers take one 25-bit and one 18-bit wide input. These limited bitwidths require careful analysis of the range of values the intermediate variables may have; this analysis is one of the steps in the t-test calibration process described in Section 4.6.

4.5 Encryption Core

The encryption core is the circuit that needs to be monitored, because of its susceptibility to power side-channel leakage. As a case study, we use AES-128 encryption cores, but our monitor is sufficiently general to be used with any cryptographic core that leaks information through the power side channel.

4.6 System Calibration

Once implemented, the system needs to be calibrated. This is required for two reasons: first, the delay-line sensor needs to be calibrated so that the values it produces are all within the observable range (none of the values reach min/max sensor limits). Second, the width of the integer and the fractional parts of all intermediate variables need to be set. To ensure that the sensor works in the operating range even when the on-chip voltage drops or becomes excessively noisy due to tampering or change of working conditions, we make the delay-line long (160 bits). Then, we run a sequence of encryptions, each time recording the sensor readings and tuning the sensor parameters, until the maximum value in the steady state is calibrated to ~130. After the sensor calibration, we run a number of fixed and random plaintext encryptions and record the power-consumption traces. Then, we compute the t-test statistic on the collected traces, in floating-point arithmetic. Finally, using the computed t-test statistic as a reference, we employ standard methods for floating-point to fixed-point format conversion to find the integer/fractional-part widths of all the intermediate variables, which respect DSP-block constraints and produce as accurate t-test results as possible.

5 EXPERIMENTAL RESULTS

We validate the system using the Sasebo-GII [4] platform and two AES-128 cores implemented on the Xilinx Virtex-5 LX30 FPGA; the first AES is a nonpipelined 128-bit datapath architecture [1], while the second is a 32-bit four-stage pipelined architecture [9]. Both cores operate at 24 MHz—the maximum frequency of the communication bus used for providing the plaintexts and offloading the ciphertexts from the FPGA [10]. The sensor and the t-test core are clocked at 96 MHz. To collect the oscilloscope traces, a Tektronix MDO3104 Oscilloscope with 1 GHz sampling frequency and an eight-bit ADC is used. The parameter k is set to 1024 traces.

To evaluate the system performance, we perform two experiments on each of the two AES cores. First, we record the oscilloscope traces of a cryptographic core only. Second, we record the sensor and the oscilloscope traces of the complete system, and the t-test core output. In every experiment, 28×2^{10} traces are collected: half using a fixed plaintext and half using random plaintexts. For both AES cores, the experiments are repeated 10 times.

Fig. 2 shows the percentage of the *leaky* trace samples in the function of the number of collected traces, averaged over all experiment runs. The full (resp. dashed) curves correspond to the results obtained using the oscilloscope (resp. sensor) traces and a t-test software routine in floating-point precision. The diamonds correspond to the hardware t-test core results. The almost perfect alignment of the diamonds and the dashed curve indicates a very good accuracy of our t-test computation in hardware. A common characteristic among all these curves is that the average leakage increases with the number of power traces. Additionally, due to the noise introduced by the components of the built-in self-evaluation system, the standalone AES core (in green) has slightly more leaky samples than the complete system (in blue).

In the case of the nonpipelined AES, the results obtained using the sensor traces closely match those obtained using the oscilloscope traces. However, when the pipelined AES is used, sensor traces seem to contain more leaky samples than the oscilloscope traces. To understand why, we look at the change in the numerators (subtraction of means) and the denominators (sum of variances) in (1), when they are computed using the sensor traces or using the oscilloscope traces. The results show that, due to the lower precision of the on-chip sensor compared to the oscilloscope ADC, the sensor traces for both AES cores have, on average, ~60% smaller sum of variances than the oscilloscope traces. At the same time, for the nonpipelined AES, the subtraction of means has ~70% higher value than for the pipelined AES (because the former occupies more resources, consumes more power, and thus creates higher voltage fluctuations). The values of the numerators in (1), for both

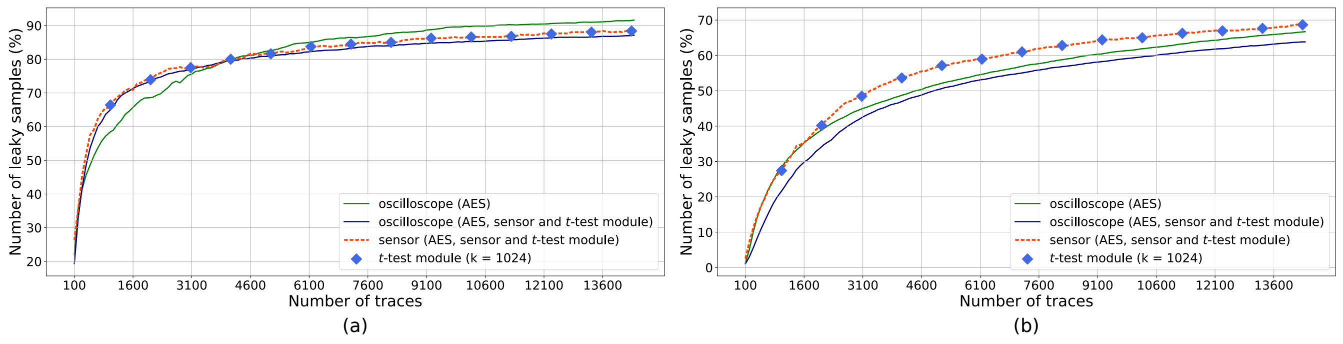

Figure 2: The percentage of power trace samples for which the t-test fails, in the function of the number of traces. The figure on the left corresponds to the nonpipelined AES, while the figure on the right corresponds to the pipelined AES core. Solid lines are obtained using the oscilloscope traces and a t-test software routine in floating-point precision. Dashed lines are obtained using the FPGA sensor traces and the same software routine, while the diamond markers—which almost perfectly overlap the dashed line—are the result of the FPGA t-test module.

the sensor and the oscilloscope traces of the nonpipelined AES, are similar and large enough to compensate for the difference between the corresponding denominators. This not being the case for the pipelined AES, the denominators in (1) have a larger influence on the t-test, resulting in a slightly increased number of leaky samples in the sensor traces.

In the next experiment, we compare the t-test values obtained using the oscilloscope traces and the floating-point computation in software with the values computed by the hardware t-test module. After aligning and downsampling the oscilloscope traces, we count the number of false positives (samples erroneously declared as leaky) and of false negatives (samples erroneously declared as not being leaky). In the case of the nonpipelined AES, we counted only 3.88% false negatives and 5.74% false positives, on average. In the case of the pipelined AES, we counted 5.63% and 13.51%, respectively. These results match the offsets between the corresponding dashed and solid lines in Fig. 2.

Table 1 summarizes the FPGA resource utilisation. The leakage-estimation part of the design (the sensor, the control logic, the DSP and the RAM blocks) uses up to ~10% of the LUTs and registers, up to ~10% of the on-chip memory, and up to ~44% of the DSP blocks available. Minor mismatch between the systems with pipelined and nonpipelined AES is due to the change in the sensor calibration and the width of the AES core datapath.

Compared to the design by Sonar et al. [13], our t-test core requires 3.8× less LUTs, 1.7× less slice registers, 2.6× less BRAMs, and 18.3× less DSP blocks, for the same number of trace samples ($N_S = 64$). This is because, unlike their solution which requires replicating the t-test module for every new trace sample, we reuse the datapath resources in a pipelined fashion and only increase the storage requirements for keeping the partial sums, the means, and the variances of the newly added samples only. Finally, Sonar et al. report the maximum relative error of 20% between the floating-point and the fixed-point leakage estimation, whereas we do not calculate the t-test value itself—we calculate its binary equivalent (1 if $|t| > 4.5$, 0 otherwise). Comparing the binary t-test value obtained using floating-point processing of sensor traces with the output of the FPGA t-test core shows the maximum of 0.8% of incorrectly classified trace samples.

Table 1: FPGA resource utilization breakdown. The number of available resources shown first. In the left columns of the two AES groups, the resources occupied by the encryption core and related modules. In the right columns, the resources used by the leakage-estimation parts of the system.

Resource Type	FPGA Total	AES		Pipelined AES	
		Encrypt. Core	Leakage Est.	Encrypt. Core	Leakage Est.
LUTs	19200	11.90%	8.23%	5.16%	9.16%
Registers	19200	4.55%	7.31%	6.30%	7.47%
DSP48E	32	-	43.75%	-	40.63%
RAMB18	64	3.13%	6.25%	3.13%	7.81%
RAMB36	64	-	9.38%	-	7.81%

6 CONCLUSIONS

In this paper we have presented and validated a built-in test for self-evaluation of power side-channel leakage, suitable for FPGAs. The system consists of a digital sensor to measure the on-chip voltage fluctuations and an engine to calculate the t-test statistic on-the-fly and thus verify the presence of information leakage. Our design is validated using two AES accelerators implemented on FPGA. When evaluating the t-test statistic, our built-in test achieves results comparable to those obtained using state-of-the-art lab equipment. The system proposed in this paper allows, for the first time, a real-time assessment of power side-channel leakage during the device operation in the field. This work will open new frontiers for ensuring security of safe and robust cyber-physical systems, which is currently verified only before the deployment.

7 ACKNOWLEDGEMENTS

Francesco Regazzoni received support from the European Union Horizon 2020 research and innovation program under CERBERO project (grant agreement number 732105).

REFERENCES

[1] AIST and Tohoku University. 2019. AES Encryption Core. (2019). http://www.aoki.ecei.tohoku.ac.jp/crypto/ Accessed: 2019-12-12.

[2] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. 2003. Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard. *IEEE Transactions on Computers* 52, 4 (April 2003), 492–505.

[3] Tony F. Chan, Gene H. Golub, and Randall J. Leveque. 1983. Algorithms for Computing the Sample Variance: Analysis and Recommendations. *The American Statistician* 37, 3 (Aug. 1983), 242–247.

[4] Research Center for Information Security. 2009. SASEBO-GII Quick Start Guide. http://satoh.cs.uec.ac.jp/SASEBO/en/board/sasebo-g2.html. (2009). Accessed: 2019-20-01.

[5] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. 2011. A Testing Methodology for Side-channel Resistance Validation. NIST Non-Invasive Attack Testing Workshop. (2011).

[6] James Howe, Ayesha Khalid, Marco Martinoli, Francesco Regazzoni, and Elisabeth Oswald. 2019. Fault Attack Countermeasures for Error Samplers in Lattice-Based Cryptography. Cryptology ePrint Archive, Report 2019/206. (2019). https://eprint.iacr.org/2019/206.

[7] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential Power Analysis. In *Advances in Cryptology—CRYPTO '99*. Santa Barbara, CA, 387–397.

[8] Itamar Levi, Davide Bellizia, and François-Xavier Standaert. 2019. Reducing a Masked Implementation's Effective Security Order with Setup Manipulations. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2019, 2 (Feb. 2019), 293–317.

[9] Francesco Regazzoni, Wang Yi, and François-Xavier Standaert. 2011. FPGA Implementations of the AES Masked Against Power Analysis Attacks. In *Proceedings of 2nd International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*. Darmstadt, Germany, 1–11.

[10] Falk Schellenberg, Dennis R.E. Gnad, Amir Moradi, and Mehdi B. Tahoori. 2018. An Inside Job: Remote Power Analysis Attacks on FPGAs. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*. Dresden, Germany, 1111–1116.

[11] Falk Schellenberg, Dennis R. E. Gnad, Amir Moradi, and Mehdi B. Tahoori. 2018. Remote Inter-chip Power Analysis Side-channel Attacks at Board-level. In *Proceedings of the International Conference on Computer-Aided Design*. New York, NY, USA, 114:1–114:7.

[12] Tobias Schneider and Amir Moradi. 2016. Leakage Assessment Methodology. *Journal of Cryptographic Engineering* 6, 2 (June 2016), 85–99.

[13] Souvik Sonar, Debapriya Basu Roy, Rajat Subhra Chakraborty, and Debdeep Mukhopadhyay. 2016. Side-Channel Watchdog: Run-Time Evaluation of Side-Channel Vulnerability in FPGA-Based Crypto-systems. Cryptology ePrint Archive, Report 2016/182. (2016). https://eprint.iacr.org/2016/182.

[14] Ji Sun, Ray Bittner, and Ken Eguro. 2011. FPGA Side-Channel Receivers. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. Monterey, CA, USA, 267–276.

[15] B. P. Welford. 1962. Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics* 4, 3 (Aug. 1962), 419–420.

[16] Xilinx 2015. *Virtex-5 Family Overview*. Xilinx. https://www.xilinx.com

[17] Bohan Yang, Vladimir Rožić, Nele Mentens, Wim Dehaene, and Ingrid Verbauwhede. 2016. TOTAL: TRNG On-the-fly Testing for Attack Detection Using Lightweight Hardware. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*. Dresden, Germany, 127–132.

[18] Mark Zhao and G. Edward Suh. 2018. FPGA-Based Remote Power Side-Channel Attacks. In *Proceedings of IEEE Symposium on Security and Privacy*. San Francisco, CA, US, 805–820.

[19] Kenneth M. Zick, Meeta Srivastav, Wei Zhang, and Matthew French. 2013. Sensing Nanosecond-Scale Voltage Attacks and Natural Transients in FPGAs. In *Proceedings of the 21th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. Monterey, CA, USA, 101–104.