A Tale of Two Shares: Why Two-Share Threshold Implementation Seems Worthwhile—and Why it is Not

Cong Chen, Mohammad Farmani, and Thomas Eisenbarth

Worcester Polytechnic Institute, Worcester, MA, USA
{cchen3,mfarmani,teisenbarth}@wpi.edu

Abstract. This work explores the possibilities for practical Threshold Implementation (TI) with only two shares in order for a smaller design that needs less randomness but is still first-order leakage resistant. We present the first two-share Threshold Implementations of two lightweight block ciphers—Simon and Present. The implementation results show that two-share TI improves the compactness but usually further reduces the throughput when compared with first-order resistant three-share schemes. Our leakage analysis shows that two-share TI can retain perfect first-order resistance. However, the analysis also exposes a strong second-order leakage. All results are backed up by simulation as well as analysis of actual implementations.

Keywords: Threshold Implementation, Paired t-test, Lightweight Cryptography, FPGA

1 Motivation

Protecting cryptographic hardware against side channel analysis is a difficult task and usually incurs significant area overheads. Especially masking schemes aimed at hardware have been found to be flawed or prone to implementation errors that leave the countermeasure at least partially insecure [13, 20, 23].

Threshold Implementation (TI) has become a popular masking scheme for hardware implementations in the recent years, due to several advantages over competing schemes. Unlike secure logic styles [32, 20], it does not require a change of the design flow. TI is fairly simple to apply to a wide range of ciphers, and its implementation is not very error-prone, if a known set of requirements and best practices is followed. Another advantage is that TI actually keeps the promise of reliable first-order side-channel resistance. It also provides good protection against higher-order attacks [24, 6].

However, like most other masking schemes, TI incurs large area and time overheads, and often consumes huge amounts of randomness for remasking, which can make practical application cumbersome. So far the best results have an area overhead of approximately three while consuming at least two times the combined plaintext and key size of randomness per encryption. Such overheads—the significant increase in area as well as the need for a high-performance random

number generator—make TI an expensive choice, too expensive for a broad range of practical applications. Reparaz et al. [27] generalized TI to provide protection against higher-order attacks. The work mentioned the feasibility of reducing the number of shares to $d+1$, where d is the desired protection order, suggesting that two shares are sufficient for first-order side channel protection. A first evaluation of using $d+1$ shares for AES was performed by De Cnudde et al. in [15].

Our contribution In this work we explore the practical implications of reducing the number of shares of threshold implementations to only two shares (2-TI). Such a reduction of shares enables implementations that only incur an area overhead of two and at the same time can also reduce the need of minimally required randomness by a factor of two, making the incurred cost more bearable and thus allowing side channel protection for a much wider range of applications. Reducing the number of shares is easily possible by applying the non-completeness requirement of TI at the bit-level rather than the state-level, as done by prevailing implementations.

While the feasibility of this approach has already been discussed in [27] and recently been practically verified in [15], this work explores the practical aspects, the benefits—and ramifications—of applying threshold implementation with only two shares to modern ciphers. Our case study focuses on applying 2-TI on two lightweight block ciphers, Present [7] and Simon [2]. Lightweight ciphers are usually a good target for TI, as the algebraic depth of their nonlinear functions is usually low. Low algebraic depth allows for cheap and effective masking while keeping the need for additional randomness low. In fact, our designs do not require remasking during the round functions, while a comparable masked implementation of AES requires more than 8,000 fresh random bits during one block encryption [15].

Our study shows that two-share TI is first order secure and also reduces the size of the sequential logic in hardware implementations. The 2-TI-conversion of nonlinear functions is more cumbersome and usually requires at least one additional pipeline stage, with negative impact on implementation size and/or performance. However, we also expose a strong second-order leakage in both of the designs and argue that this is inherent to two-share TI implementations. We show that these leakages exist both in the theoretical model and can also be quickly exposed by leakage detection tests. We validate the exploitability of the observed leakages by side channel key recovery attacks.

The remaining work is structured as follows: Relevant terminologies and methods are explained in Section 2. The theoretical discussion of two-share TI is given in Section 3 and two practical implementations of Simon and Present are introduced in Section 4 and 5. Sections 6 and 7 present implementation results and the outcome of the leakage analysis and we conclude at Section 8.

2 Preliminaries

2.1 Lightweight Cryptography

For many embedded applications, area and hence power or energy minimal implementations of cryptography are highly desirable. This has led to a rich literature on hardware-minimal crypto cores, which often rely on the numerous proposed "lightweight" block cipher designs, such as Present, Katan, or Simon and Speck. These lightweight ciphers as well as the area-minimal implementations share one common characteristic: *Serialization*.

Serialized implementations are very common for minimizing area of hardware implementations at the expense of increased run time. Area-critical functions are identified and broken into subfunctions that can be applied repeatedly, in an iterative manner, to achieve the same outcome. A typical example for block ciphers is the S-box layer, which due to its high nonlinearity usually is difficult to minimize in hardware. A classical area-optimized implementation of an S-box based cipher only features a single S-box, which is iteratively applied to different parts of the intermediate state. All modern block ciphers support this *vertical* type of serialization by using a single S-box (unlike DES which uses 8 different S-boxes). Similar techniques are also applied to decrease the size of large S-boxes (or in general functions of great algebraic complexity), by breaking them into subfunctions that are concatenated. Examples include implementations that compute the AES S-box by exploiting tower field representations by Canright [9] or the Present S-box into mappings of algebraic degree 2, which eases side-channel protection and decreases the size, at the cost of doubling the computation time [26]. We will refer to this serialization as *horizontal*. While vertical serialization is determined by the cipher at implementation time (usually determined by the number of S-boxes), the exploitable horizontal serialization is determined by the algebraic complexity of the nonlinear layer.

Typical vertical serialization parameters for hardware minimal implementations are ranging from data path sizes of 8 bit for AES, 4 bit for Present down to 1 bit for e.g. Simon or Katan. That is, as little as one bit of the cipher state are updated per cycle. Serial data paths increase the latency of the crypto core significantly. However, they also allow to reduce the combinational logic of the crypto core to low single-digit percentages of the entire design [29, 14]. That means, in applications where the latency is not critical, the area of a cipher is almost entirely determined by the registers storing the key and state. As a result, significant area-improvements can only be achieved by breaking the memory barrier, for example by externalizing key storage (cf. Ktantan [14]), or, for FPGAs, hiding state and key in dedicated bulk memory such as block RAMs [19] or shift registers [1]. Since the remainder of the work uses Present and Simon for proof-of-concept implementations, we provide more details on these two ciphers here.

2.2 Present

Present is a hardware-oriented block cipher proposed in 2007, optimized for low area footprint [7]. It is a substitution-permutation network featuring a 4×4 bit S-box and a permutation layer consisting only of bit shifts, making it low cost in hardware. It features a block size of 64 bits and a key size of 80 or 128 bits, and has 31 rounds.Present has been optimized for many application scenarios, but the area-minimal implementations with a 4-bit data-path. It has also been standardized as a lightweight cryptographic block cipher as ISO/IEC 29192-2:2012. Each round of Present cipher consists of three steps including a key-addition layer, a substitution layer which is a non-linear function, and a permutation layer. In the first step, the round key which is consisted of left most significant 64 bits of the key is xored with the 64-bit current state. In the next step, the Present S-box is used which is a non-linear 4-bit to 4-bit function shown in the following table in hexadecimal notation.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

The substitution layer can be performed with 16 parallel S-box or using only one S-box 16 times which depends on the application requirement. In the last step, the permutation is applied to all the 64-bit data which is just a rewiring.

At the same time, the key is updated in the key schedule part. The key can be 80-bit or 120-bit; however we use 80-bit key in this paper. In each round the 64 left most bits of the current key, $k_{79}k_{78}k_{77}...k_{17}k_{16}$, is used in addroundkey. After using the round key, the 80-bit key register is updated by shifting, using S-box, and xoring with round-counter. More details about the specification of the Present is provided in [7].

2.3 Simon

Simon is a lightweight block cipher proposed by NSA in 2013 [2]. Simon implements a Feistel structure that accepts two n-bit words as input plaintext, with $n \in \{16, 24, 32, 48, 64\}$. For each input size $2n$, Simon has a set of allowable key sizes ranging from 64 bits to 256 bits. The number of rounds in Simon ranges from 32 to 72 rounds. Simon128/128, which can be seen as a drop-in replacement for AES-128, accepts 128 bits of plaintext at a word size of 64 bits and 128 bits of key. It generates a ciphertext after 68 rounds. The Simon128/128 parameter set will be used throughout this work, though the implementation strategies apply to other parameter sets in a natural way..

We denote the input words of round i as l_i and r_i. Then the output words are given as:

$$\begin{aligned} r_{i+1} &= l_i \\ l_{i+1} &= r_i + l_i^2 + (l_i^1 * l_i^8) + k_i \end{aligned} \tag{1}$$

The upper index in l_i^s indicates left circular shift by s bits. The addition and the multiplication are in GF(2) and equivalent to bitwise XOR and AND operations, respectively. Given the initial key words k_0 and k_1 (and possibly k_2 and k_3, depending on the key size), which are also used as first round keys, the subsequent round keys are computed as:

$$\begin{aligned} k_{i+2} &= k_i + k_{i+1}^{-3} + k_{i+1}^{-4} + c_i \quad \text{Two and Three Words} \\ k_{i+4} &= k_i + k_{i+1} + k_{i+1}^{-1} + k_{i+3}^{-3} + k_{i+3}^{-4} + c_i \quad \text{Four Words} \end{aligned} \tag{2}$$

where c_i is a round constant.

2.4 Masking

Masking is a common technique to prevent side channel leakage [10]. Sensitive states of a cryptographic implementation are split into shares by adding randomness. In an additive masking scheme, a variable x is split into s shares x_i with $i \in \{0, 1, \ldots, s-1\}$ by choosing $x_{i>0}$ uniformly at random and $x_0 = x + \sum_{i=1}^{s-1} x_i$. These shares are then processed separately, ensuring that the sensitive state is never presented in the system, and—more importantly—that processed states are independent of the secret.

2.5 Threshold Implementation

Threshold Implementation (TI) was proposed by Nikova et al [25] as a side-channel countermeasure to address the common problem of *glitches* that resulted in leakage for many other theoretically sound countermeasure techniques when applied to hardware. The original proposal only deals with protection against first-order side-channel leakages. Threshold Implementation has found widespread adoption in the academic community: several implementations of symmetric [26, 24, 5, 6, 31] and even asymmetric crypto algorithms [11, 28] have been successfully protected with TI. Recently, TI has been expanded to protect against higher-order attacks as well [4], though potential pitfalls of the scheme in the multivariate setting have been pointed out and fixed in [27].

TI combines a set of three requirements with a constructive description of how to convert an algorithm into a side-channel resistant implementation in the presence of glitches. Sensitive states are converted into a shared representation by applying an additive Boolean masking, i.e., adding randomness. Functions $F(\cdot)$ are converted meeting the requirements of correctness, uniformity, and non-completeness.

– **Uniformity** requires all intermediate states (shares) to be uniformly distributed. Uniformity is intended to ensures the mean leakages to be state-independent, a key requirement to thwart first-order DPA. To ensure uniformity in a circuit it suffices to ensure uniformity for the output share of each function, as well as for the inputs of the circuit.
– **Non-Completeness** requires subfunctions f_i of a shared function F to be independent of at least one input share for first-order SCA resistance. That is, a function $F(x)$ shall be split into subfunctions $f_i(x_{j\neq i})$. This requirement was updated in [4] to require any d subfunctions to be independent of at least one input share to achieve d-th order SCA resistance. Non-completeness ensures that the final circuit is not affected by glitches. Since glitches can only occur in subfunctions f_i, and each subfunction has insufficient knowledge to reconstruct a secret state (since it has no knowledge of at least one share x_i), no leakage can be caused by glitches.
– **Correctness** simply states that applying the subfunctions to a valid shared input must always yield a valid sharing of the correct output.

In the classic approach, a function of algebraic degree t can be implemented using at least $t+1$ input shares for first order side-channel resistance, and $td+1$ for d-th order resistance [4, 27]. In practice, virtually all implementations try to keep the number of shares low, i.e. for first order-protected designs at or close to 3. As a consequence, implementations of algebraically more complex functions need to be broken into algebraically simpler subfunctions. The described TI conversion always ensures correctness and non-completeness. Uniformity can be either achieved by using more input shares or by adding randomness during the computation. As a result, many of the published implementations, in order to reduce the size of the circuit, consume lots of randomness, up to thousands of bits per encrypted block.

2.6 Leakage Detection

A side channel leakage detection method based on Welch's t-test has been recently gaining popularity due to its simplicity, efficiency and reliability. The test procedures have been well studied in [12] and [30] and is often referred to as Test Vector Leakage Assesment (TVLA) test. Unlike other attacks or leakage models used for key recovery, TVLA only returns a confidence level to reject the leakage-free hypothesis and fail the device under test. Essentially, a t-statistic is calculated using two sets of leakage samples as:

$$t = \frac{\mu_A - \mu_B}{\sqrt{(\sigma_A^2/N_A) + (\sigma_B^2/N_B)}} \tag{3}$$

where A and B denote the two sets and N_j denotes the number of traces in set $j \in \{A, B\}$. μ_j and σ_j are the sample mean and sample variance respectively. The two sets of measurements are obtained with either fixed versus random plaintext (in a *non-specific t-test*) or random versus random plaintext (in a *specific t-test*). In our work we use the *non-specific t-test* since it does not depend

on any intermediate value and power model. When the value of t exceeds a certain threshold, the null hypothesis can be rejected with a small Type I error probability p. In this paper, we follow the threshold of ± 4.5 used in [18] and [22].

An improved methodology based on paired t-test was suggested in [16]. The test uses matched pairs from the two sets of measurements. The advantage of this methodology is that common noise to both measurements can be rejected, making the test much more robust to slow changes of operating points in long measurement campaigns. When n such pairs of measurements are obtained, we have n difference measurements $D = L_A - L_B$ where L_A is a random variable representing samples from set A while L_B from set B. The paired difference cancels the noise variation and makes it easier to detect nonzero population difference. Now, the null hypothesis becomes mean difference $\mu_D = 0$ instead of $\mu_A = \mu_B$. Let $\bar{D}$ and s_D^2 denote the sample mean and sample variances of the paired differences D_1, ..., D_n. The paired t-test statistic is calculated as:

$$t_p = \frac{\bar{D}}{\sqrt{\frac{s_D^2}{n}}}, \tag{4}$$

The null hypothesis of non-leakage is also rejected if $|t_p|$ exceeds the threshold of 4.5.

With respect to higher order leakage detection, the original traces should be preprocessed as explained in [30]. For example in a second order t-test, the traces - at each sample points independently - are mean free squared beforehand. Usually, the global mean of all samples at each time point is used. However, as suggested in [16], a moving average which is the average of neighboring traces around each trace is used instead to mitigate the environmental effects. In our experiments, we apply both tests, the classic TVLA test as well as the paired T-test, the latter one with moving averages for higher-order analysis.

3 Threshold Implementation with Two Shares

While the constructive approach by Nikova et al. allows to implement any d-th order algebraic functions in a straightforward way, actual implementations requiring to share functions of degree greater than 2 have put significant effort into keeping the number of shares as close as possible to three, which is perceived as the minimum possible to implement nonlinear functions, until [27]. [1] In particular, [21] discussed the efficient implementation of 4-bit S-boxes with three shares. Similarly, the current TIs of AES utilize the algebraic structure of the AES S-box and four [24] or variable with up to five shares [6] to implement the S-box on a small area.

A natural question is: *Why to stop at three shares?* If small area is desirable, using similar techniques as the ones used by the above papers could enable TIs

[1] It should be noted that [31] also proposed a two-share TI version of Simon, with the requirement of manually preventing glitches for two parts of the equation.

with just two shares, further reducing the area footprint as well as the need for randomness. This approach was already discussed in [27]. The approach is straightforward for the linear operations of an implementation, and has already been widely used in several TIs for those parts [6, 11, 3]. The simplest nonlinear operation is a simple two-input and: $c = ab$ which can be processed with two shares as

$$c_0 = a_0 b_0 \qquad c_1 = a_1 b_1 \qquad c_2 = a_0 b_1 \qquad c_3 = a_1 b_0 \tag{5}$$

This equation is in violation of the common interpretation of the non-completeness requirement, since c_2 and c_3 mix inputs from shares with different indices. However, non-completeness is not violated as long as a and b are statistically independent.

Equation (5) suggests a 4-share output, which is undesirable for a minimal implementation. To keep the number of shares low, the four shares c_i can be recombined in the next cycle, e.g. $c'_0 = c_0 + c_2$ and $c'_1 = c_1 + c_3$. However, since the recombination would violate non-completeness, it must happen after a register-stage in the next clock cycle. In other words, a pipelining stage becomes necessary, increasing the register count and the delay of the output. The share proliferation gets worse for higher-degree algebraic functions, as stated in [27]. However, hardware-minimal implementations break higher-order algebraic functions into degree-minimal building blocks anyway, making share proliferation a theoretical concern only.

To also ensure uniformity and thus gain an implementable basic nonlinear building block, we implement $z = ab + c$ in two pipeline stages as

$$z'_0 = a_0 b_0 + c_0 \qquad z'_1 = a_1 b_1 + c_1 \qquad z_0 = z'_0 + a_0 b_1 \qquad z_1 = z'_1 + a_1 b_0 \tag{6}$$

Note that z'_i and z_i are computed in separate cycles. Conveniently, the z'_i and z_i are uniform. Furthermore, this computation order only needs to store 2 intermediate states (unlike eq. (5)). However, this assumes that the inputs are available in two subsequent clock cycles, which is a valid assumption in many serialized implementations. Either way, the resulting pipelining of the nonlinear function increases area overhead of that function, and also introduces a latency according to the number of pipeline stages needed. Most of this latency can be hidden if the data path of the implementation is small enough.

3.1 Potential Pitfalls

Share rotation In [26] it was suggested to rotate the shares in every step to achieve increased side channel resistance. With two shares, this is highly dangerous: if s_0 overwrites s_1, the resulting leakage is likely to depend on both shares, hence has a direct dependence on the secret itself. In general, any register updates must be handled with great care.

Increased Higher-order leakage The observed higher order leakage can be explained by the significant dependende of the variance on the value of the share

Table 1. Comparison of leakage for a 2-sharing (S_2) and 3-sharing (S_3) of a bit x in a Hamming weight model. The 2-sharing (S_2) shows a leakage in the variance $\sigma(S_2)$.

x	$S_2(x)$	$S_3(x)$	$wt(S_2)$	$wt(S_3)$	$\mu(S_2)$	$\mu(S_3)$	$\sigma(S_2)$	$\sigma(S_3)$
0	$\{00, 11\}$	$\{000, 011, 101, 110\}$	$\{0, 2\}$	$\{0, 2, 2, 2\}$	1	$3/2$	**2**	1
1	$\{01, 10\}$	$\{001, 010, 100, 111\}$	$\{1, 1\}$	$\{1, 1, 1, 3\}$	1	$3/2$	**0**	1

x. For a simple example we compare a 2-sharing S_2 and a 3-sharing S_3 of a bit x into $S_2(x) = \langle x_0, x_1 \rangle$ and $S_3(x) = \langle x_0, x_1, x_2 \rangle$ respectively. We further assume a Hamming weight ($wt(\cdot)$) leakage on the shares. Table 1 lists the possible states and the resulting means and variances for both sharings.

As proper TI sharings of x, the mean leakage $\mu(S_i)$ is independent of the value of x. However, the variance of S_2 depends on x, in particular $\text{var}(S_2(x = 0)) = 2 \neq 0 = \text{var}(S_2(x = 1))$. This is not true for the 3-sharing S_3, where the variances in both cases are identical as well. This is a strong indication why 2-sharings may have a strong second-order leakage. This was also observed for partial 2-share implementations in [3] and will be demonstrated for full 2-share implementations in the analysis of our reference implementations in Section 7.

4 Application to Simon

Threshold Implementations of Simon with three shares have been proposed in [31] to counteract first-order side channel attacks. Moreover, their bit-serialized implementation only consumes 87 slices on Spartan-3 xc3s50 FPGA which renders it the smallest threshold implementation of a block cipher. The authors also discussed how the requirement of *non-completeness* shuts the door on a two-share hardware implementation of Simon but not on software implementations.

In this section, we at first apply serialization technique in order to realize a two-share TI Simon on hardware. The leakage detection analysis and implementation results will be presented in Sections 6 and 7.

4.1 Simon with Two Shares

We follow the notation used in [31] to describe the cipher. The input plaintext is initially split into two shares as:

$$\begin{aligned} r[a]_0 &= m[p][1] \\ l[a]_0 &= m[p][2] \\ r[b]_0 &= m[p][1] + r_0 \\ l[b]_0 &= m[p][2] + l_0 \end{aligned} \tag{7}$$

Where r and l represents the two input words, a and b denote two shares of the variables and subscript i indicates the round of encryption. $m[p][1]$ and $m[p][2]$

are two fresh random values that mask the plaintext in the very beginning of the algorithm and no more random numbers are needed for the rest operations. Then, the round function is denoted as:

$$\begin{aligned} r[a]_{i+1} =& l[a]_i \\ l[a]_{i+1} =& r[a]_i + l[a]_i^2 + l[a]_i^1 * l[a]_i^8 + l[a]_i^1 * l[b]_i^8 + k[a]_i \\ r[b]_{i+1} =& l[b]_i \\ l[b]_{i+1} =& r[b]_i + l[b]_i^2 + l[b]_i^1 * l[b]_i^8 + l[b]_i^1 * l[a]_i^8 + k[b]_i \end{aligned} \tag{8}$$

Where the superscripts $1, 2, 8$ on $l[*]_i$ represent left circular shift by corresponding numbers of bits. (Notice that both addition and multiplication are in GF(2)). Obviously, the computations of $l[a]_{i+1}$ and $l[b]_{i+1}$, if directly mapped into combinational circuits, are not *non-complete* since the two shares $l[a]_i^8$ and $l[b]_i^8$ are present in the same circuit and glitches may still cause leakage. We can serialize the above equations by enforcing them being executed in two steps rather than one. That is, we first compute the intermediate values $l[a]_{i+1,int}$ and $l[b]_{i+1,int}$ using only half of the terms in the equations as follows:

$$\begin{aligned} l[a]_{i+1,int} =& r[a]_i + l[a]_i^2 + l[a]_i^1 * l[a]_i^8 \\ l[b]_{i+1,int} =& r[b]_i + l[b]_i^2 + l[b]_i^1 * l[b]_i^8 \end{aligned} \tag{9}$$

Then, the round outputs can be further calculated as:

$$\begin{aligned} l[a]_{i+1} =& l[a]_{i+1,int} + l[a]_i^1 * l[b]_i^8 + k[a]_i \\ l[b]_{i+1} =& l[b]_{i+1,int} + l[b]_i^1 * l[a]_i^8 + k[b]_i \end{aligned} \tag{10}$$

The serialization not only retains both *correctness* and *uniformity* but achieves *non-completeness* as well. In Equation (9), the inputs $r[a]_i$, $l[a]_i^2$, $r[b]_i$ and $l[b]_i^2$ are all uniform and therefore the output intermediates are also uniform. Each function is independent of one share of every input and hence is *non-complete*. Similarly, Equation (10) also satisfies the three requirements. Correctness can be easily proved by substituting $l[a]_{i+1,int}$ and $l[b]_{i+1,int}$ with Equation (9). The uniformity of inputs $k[a]_i$ and $k[b]_i$ makes the outputs uniform too. Moreover, each function is independent of one share of every input and thus the functions are *non-complete* as well. One may argue that $l[a]_i^1$ and $l[b]_i^8$ (or $l[b]_i^1$ and $l[a]_i^8$) are two shares of l_i with different rotations and may leak information of l_i. However, the multiplication between them is in GF(2) and is equivalent with bitwise AND operation. Further, in order to ensure the *non-completeness*, "Keep Hierarchy" property of synthesize tool (ISE with XST) is enabled to separate the LUTs for AND.

4.2 Round-based Implementation

Figure 1 depicts the structure of a FPGA implementation which contains two copies of the same data-path which consists of two registers L_j and R_j and

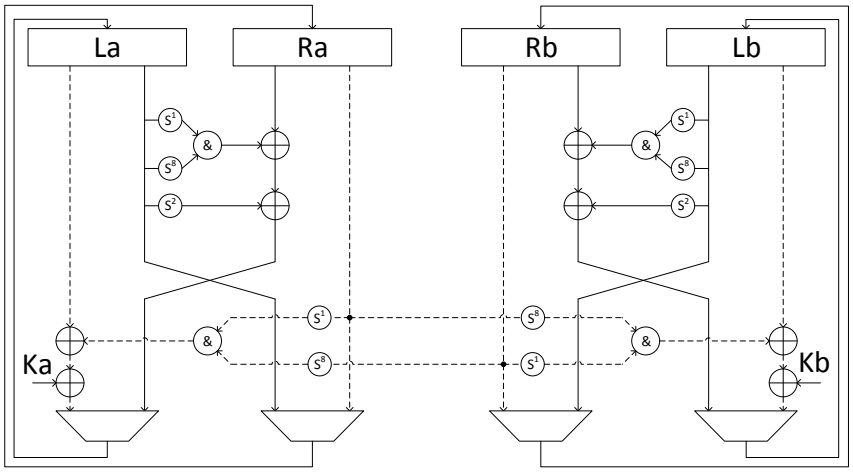


Fig. 1. Data-path of the Simon with Two Shares. Solid line: First clock cycle; Dashed line: Second clock cycle

the combinational circuits for round functions. Specifically, two clock cycles are taken to process each round operation. In the first clock cycle, the round inputs are evaluated with Equation (9) and then the intermediates are overwritten back into the registers as illustrated by the solid lines in the figure. Note that $r[j]_{i+1} = l[j]_i$ is stored in R_j while $l[a]_{i+1,int}$ is in L_j. Then, in the second clock cycle, Equation (10) is evaluated as shown by the dashed line but remember that since $l[j]_i$ is now stored in R_j and hence no extra buffer is needed for it.

The sharing of key schedule is not presented here since it consists of linear operations only and is trivial to implement.

4.3 Bit-serialized Implementation

In order to fairly compare with the bit-serialized 3-TI Simon introduced in [31] and achieve a even smaller size of Simon implementation, a bit-serialized 2-TI Simon is constructed as depicted in the Figure 2 (Only one share is shown).

Our design originates from the FIFO-based 3-TI bit-serialized in [31] but introduces new features in order for a 2-TI architecture.

First of all, the round function is adjusted according to Equation 9 and 10. (Note that both equations are evaluated in bits instead of the whole word in this case.) Therefore, as shown in the **LUT** part of Figure 2, a one-bit register is inserted to hold the intermediate value $l[a]_{i+1,int}$ so that $l[a]_i^8$ and $l[b]_i^8$ will not be combined to cause leakages mistakenly.

Second, due to the insertion of this register, it will take two clock cycles for **LUT** to perform round operation for each bit. However, by using pipeline technique, the overall throughput will not be scarified too much. In fact, the

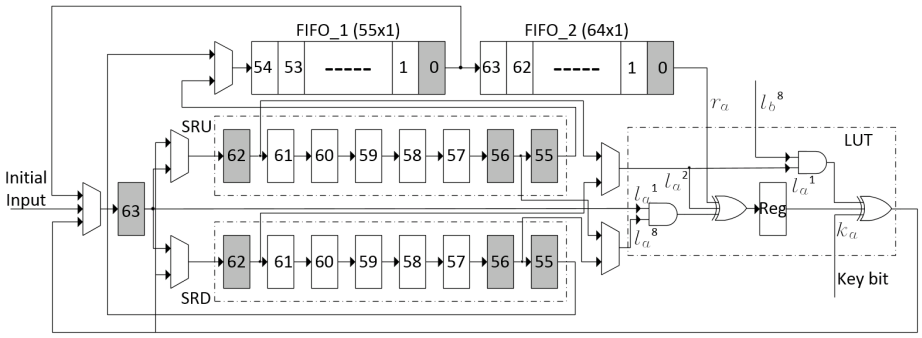


Fig. 2. Data-path of the bit-serialized 2-TI Simon

2-TI architecture processes all 64 bits within 65 clock cycles which is only one more than 3-TI in [31]. In order to achieve this, the FIFOs and shifted registers are designed to work as following.

- Initially, the 128-bit block is stored in register #63, Shifted Registers Up (SRU) #62 to #55, FIFO_1 and FIFO_2.
- Once Encryption started, the values are right shifted and in the mean time bits in register #63, #62 and #56 as well as bit 0 in FIFO_2 are fed into **LUT** for logic operation.
- The output will be written back to Shifted Registers Down (SRD). Note that the valid outputs are generated since the second clock cycle. And then, after 64 clock cycles, the first 63 output bits are stored in Shifted Registers Down (SRD) #62 to #55, FIFO_1 and FIFO_2. In the last (65th) clock cycle, the final output bit will be written in register #63. Therefore, the whole round operation is done within 65 clock cycles.

5 Application to Present

In this section, we apply two-share Threshold Implementation to the Present cipher. In [21], the authors presented the 3-TI Present S-box. To achieve this, they decomposed the non-linear S-box of degree 3 into the combination of two quadratic functions—G function—plus some linear functions, and then implement them with three shares. We follow their idea to use the same decomposition but then implement them with 2-TI while still retaining *uniformity*, *non-completeness*, and *correctness*. According to [21], the S-box of Present can be decomposed as:

$$S(X) = A(G(G(BX \oplus c)) \oplus d) \tag{11}$$

Where $G(.)$, A, B, and the constant vectors of c, d are given as follows:

$$
\begin{aligned}
G(x,y,z,w) =& (g_3, g_2, g_1, g_0) : \\
g_3 =& x + yz + yw \\
g_2 =& w + xy \\
g_1 =& y \\
g_0 =& z + yw
\end{aligned} \tag{12}
$$

$$
A = \begin{bmatrix} 1\,0\,1\,0 \\ 0\,1\,0\,0 \\ 1\,0\,0\,0 \\ 1\,0\,1\,1 \end{bmatrix}, B = \begin{bmatrix} 1\,1\,0\,0 \\ 0\,1\,1\,0 \\ 0\,0\,1\,0 \\ 0\,1\,0\,1 \end{bmatrix}, c = \begin{bmatrix} 0\,0\,0\,1 \end{bmatrix}, d = \begin{bmatrix} 0\,1\,0\,1 \end{bmatrix} \tag{13}
$$

5.1 Present with Two Shares

A 2-sharing scheme of $G(.)$ can be expressed as follows:

$$
\begin{aligned}
G_0(x_0,y_0,z_0,w_0,x_1,y_1,z_1,w_1) =& (g_{03}, g_{02}, g_{01}, g_{00}) \\
g_{03} =& x_0 + y_0z_0 + y_0z_1 + y_0w_0 + y_0w_1 \\
g_{02} =& w_0 + x_0y_0 + x_1y_0 \\
g_{01} =& y_0 \\
g_{00} =& z_0 + y_0w_0 + y_0w_1
\end{aligned} \tag{14}
$$

$$
\begin{aligned}
G_1(x_0,y_0,z_0,w_0,x_1,y_1,z_1,w_1) =& (g_{13}, g_{12}, g_{11}, g_{10}) \\
g_{13} =& x_1 + y_1z_0 + y_1z_1 + y_1w_0 + y_1w_1 \\
g_{12} =& w_1 + x_0y_1 + x_1y_1 \\
g_{11} =& y_1 \\
g_{10} =& z_1 + y_1w_0 + y_1w_1
\end{aligned} \tag{15}
$$

The above sharing satisfies both *correctness* and *uniformity* when the input shares are uniformly distributed. However, *non-completeness* is not fulfilled since two shares of the same inputs are fed into the same functions in some of the above equations.

As before, we serialize the computations into two steps in order to achieve *non-completeness* as illustrated in the following equations.

$$
\begin{aligned}
G_0^1(x_0,y_0,z_0,w_0) =& (g_{03}^1, g_{02}^1, g_{01}^1, g_{00}^1) \\
g_{03}^1 =& x_0 + y_0z_0 + y_0w_0 \\
g_{02}^1 =& w_0 + x_0y_0 \\
g_{01}^1 =& y_0 \\
g_{00}^1 =& z_0 + y_0w_0
\end{aligned} \tag{16}
$$

$$
\begin{aligned}
G_0^2(x_1, y_0, z_1, w_1, g_{03}^1, g_{02}^1, g_{01}^1, g_{00}^1) =& (g_{03}^2, g_{02}^2, g_{01}^2, g_{00}^2) \\
g_{03}^2 =& g_{03}^1 + y_0 z_1 + y_0 w_1 \\
g_{02}^2 =& g_{02}^1 + x_1 y_0 \\
g_{01}^2 =& g_{01}^1 \\
g_{00}^2 =& g_{00}^1 + y_0 w_1
\end{aligned} \tag{17}
$$

$$
\begin{aligned}
G_1^1(x_1, y_1, z_1, w_1) =& (g_{13}^1, g_{12}^1, g_{11}^1, g_{10}^1) \\
g_{13}^1 =& x_1 + y_1 z_1 + y_1 w_1 \\
g_{12}^1 =& w_1 + x_1 y_1 \\
g_{11}^1 =& y_1 \\
g_{10}^1 =& z_1 + y_1 w_1
\end{aligned} \tag{18}
$$

$$
\begin{aligned}
G_1^2(x_0, y_1, z_0, w_0, g_{13}^1, g_{12}^1, g_{11}^1, g_{10}^1) =& (g_{13}^2, g_{12}^2, g_{11}^2, g_{10}^2) \\
g_{13}^2 =& g_{13}^1 + y_1 z_0 + y_1 w_0 \\
g_{12}^2 =& g_{12}^1 + x_0 y_1 \\
g_{11}^2 =& g_{11}^1 \\
g_{10}^2 =& g_{10}^1 + y_1 w_0
\end{aligned} \tag{19}
$$

The superscript indicates the level of the circuit. Until now, we achieved a *correct*, *non-complete* and *uniform* two-share implementation of $G(.)$. the conversion of the remaining linear operations is discussed next.

5.2 Hardware Implementation

As depicted in Figure 3, in order to provide the *non-completeness* to the design, we use registers to separate the two parts of the G. The second part of the shares (G_0^2 and G_1^2) use not only the outputs of the first part of the shares (G_0^1 and G_1^1) but also some of their inputs as well (depicted in Figure 3). One 6-bit register and two 4-bit registers are used before the second part of the G module, to store the inputs x_0, x_1, z_0, z_1, w_0, and w_1; and the outputs of the first part of the G module, respectively.

In Figure 4, the S-box architecture is depicted which includes two G modules, and functions $BX+c_0$ and $AX+d_0$ for the first share as well as functions $BX+c_1$ and $AX+d_1$ for second share in which $c_0+c_1 = c$ and $d_0+d_1 = d$. Furthermore, due to *non-completeness*, we use another row of registers in between two $G(.)$ functions in the S-box. One may argue that registers should also be inserted between non-linear functions (e.g. $G(.)$) and linear functions (e.g. $AX + d_0$), since when they are merged the two shares of certain variables may be combined again which fails the *non-completeness* requirement. While this is true in general

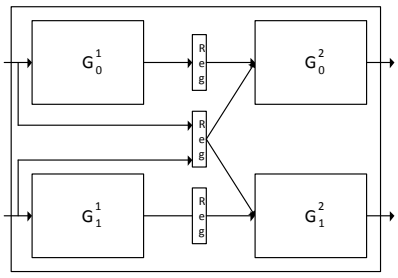


Fig. 3. Hardware architecture of the 2-share G module

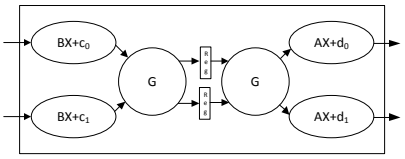


Fig. 4. Hardware architecture of the 2-share S-box module

cases, our design avoids this problem as G_0^2 and G_1^2 are both independent of one share of the inputs and hence any linear combination of $g_{13}^2, g_{12}^2, g_{11}^2, g_{10}^2$ or $g_{03}^2, g_{02}^2, g_{01}^2, g_{00}^2$ still satisfies *non-completeness*.

Figure 5 shows the whole Present cipher with two shares. The design includes two control inputs namely `key_load` and `data_load`. If `key_load` is high, at the rising edge of the clock signal, the 80-bit input key shares-Key A and Key B- are copied to the registers Key A and Key B respectively. When the `data_load` signal is high, at the rising edge of the clock signal, 64 right-most significant bits of the input shares (`data_in A[63:0]`, `data_in B[63:0]`) are copied to state registers. It is worth mentioning that when the `data_load` is set, i.e. loading new two shares of plaintext into the state registers results in a reset of the state machine. That why this design does not have a reset signal. When the two-share keys and two-share plaintexts are loaded, both `key_load` and `data_load` must be set to zero. After that, it takes 31 rounds in order to `Data_out A` and `Data_out B` have a valid ciphertexts. In each round, the S-box and permutation operations respectively operate the inputs to update the state registers for the next round. Considering the hardware design, each $G(.)$ function needs one cycle and then every S-box needs four clock cycles to compute table lookup. According to the Figure 5, each 64-bit input stored in the State register needs to use S-box 16 times. Hence, it needs 4 clock cycles for the first S-box due to its latency, plus 15 clock cycles for other 15 S-boxes in pipeline, also one more clock cycle for the permutation operation. Therefore, we need 20 cycles for each round of the Present cipher. Hence, we define another control signal, 'counter', in which it updates the state registers and Key registers after each 20 cycles. After each cycle of these 20 cycles, the state registers are shifted to the right by 4 bits and the four most significant bits of the state registers are replaced by the outputs of substitution and permutation network. The Present cipher has 31 rounds, hence a full encryption of a 64-bit input takes 620 clock cycles. We also design an unprotected Present cipher to show the area overhead of the protected Present versus unprotected one as well as its impact on maximum frequency and throughput. The comparison results are shown in Table 2.

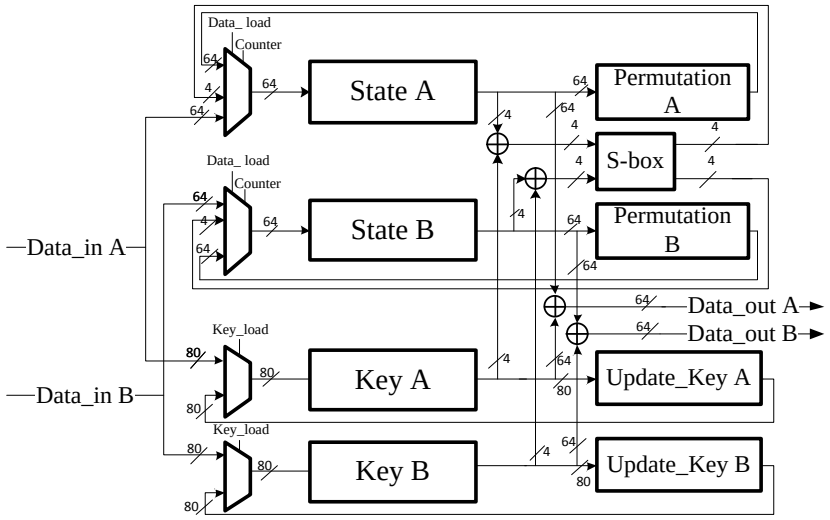


Fig. 5. Hardware architectures of the 2-shares Present Cipher.

6 Implementation Results

Table 2 summarizes the overhead and performance of two-share implementations of both ciphers. Note that we only implement Simon128/128 and Present64/80 as an example to show the advantage of two-share scheme. All the designs are implemented in Verilog and synthesized for Virtex-5 (xc5vlx50) or Spartan-3 (xc3s50) using XST.

For round-based Simon, we have three different implementations: unprotected, 2-TI and 3-TI. In terms of slice registers used, two-share TI implementation costs twice as much as the unprotected one and one third less than the 3-TI implementation. This is not surprising since increasing by one share will consume one more copy of registers to store the new share. Similarly, number of LUTs also increases. However, each round operation in 2-TI costs double clock cycles and therefore the throughput is greatly reduced compared with the other two designs.

We also implement bit-serialized 2-TI Simon to compare with the currently smallest block cipher designs for FPGAs, as given in [1], as well as its first-order protected 3-TI version from [31]. As shown in Table 2, our 2-TI design reduces the area overhead when compared to the 3-TI by about 13%, i.e., cannot quite reach the optimal reduction of 33% due to the pipelining overhead and the unaffected control logic. Nevertheless, this yields the smallest first-order protected block cipher design for FPGAs with the same parameters as AES-128.

Table 2. Implementation results of two-share Simon and Present.

Design	Slice (Regs)	Slice (LUTs)	Max. Frequency (MHz)	Throughput (Mbps)
Present on Virtex 5				
3-TI Present	466 (3.0x)	715 (3.1x)	397.289	45.567
2-TI Present	370 (2.4x)	742 (3.2x)	490.252	50.61
Present	154 (1x)	234 (1x)	394.563	40.73
Round-based **Simon** on Virtex 5				
3-TI Simon	777 (2.8x)	1302 (2.8x)	414	779
2-TI Simon	520 (1.9x)	1169 (2.5x)	382	360
Simon	272 (1x)	473 (1x)	421	792
Bit-serialized **Simon** on Spartan 3				
3-TI Simon [31]	61 (2.0x)	160 (2.2x)	109.4	3.21
2-TI Simon	55 (1.8x)	135 (1.9x)	91.1	2.64
Simon [1]	30 (1x)	72 (1x)	91.4	2.69

With respect to Present, we have three implementations: Unprotected, Regular 3-TI, and the new 2-TI Present. In terms of slice registers used, regular 3-TI implementation used more than three times of the unprotected one. This is because we should use extra registers to guarantee the *non-completeness* of first-order resistant three-share Present cipher. Also, two-share implementation costs more than two times of unprotected Present because of the same reason mentioned before. Moreover, it is worth mentioning that the 2-TI first order resistant implementation uses less registers than 3-TI. For example, we use extra registers in $G(.)$ function as explained in Section 5. These registers help reducing the critical path, which explains the speed-up and resulting increase in throughput for 2-TI Present.

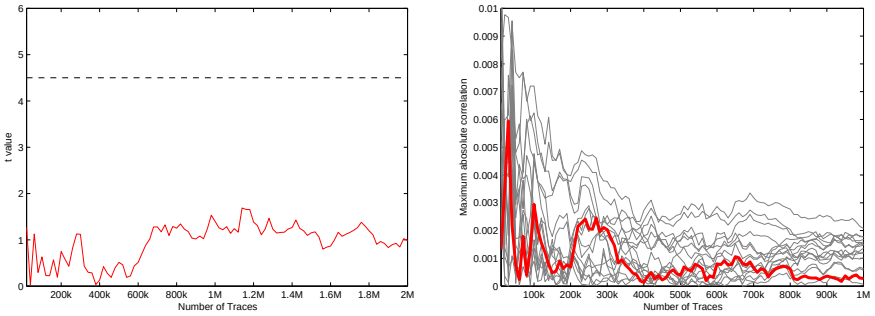
7 Leakage Analysis

In this section, we extend the discussion of a strong second-order leakage of two-share TI scheme, which was already described in Section 3.1, using simulation based leakage and the measurements from our reference implementations.

7.1 Theoretical Analysis

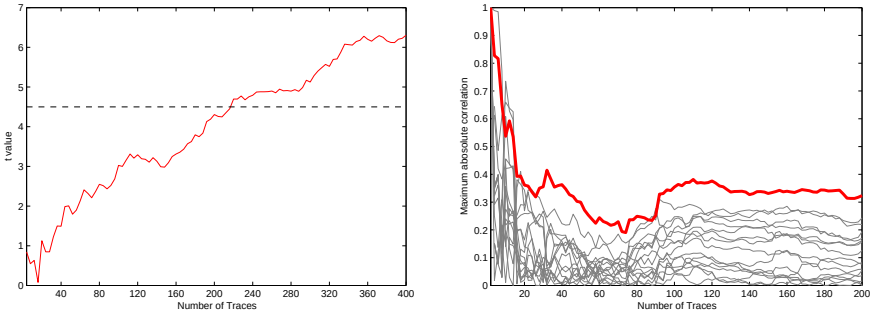
First we discuss the strong second-order leakage of two-share TI scheme using two-share Present S-box look-up as a target, namely the key-dependent inter-

mediate value $y = S(x \oplus k)$ where x, y, k are 4-bit input plaintext, S-box output and sub-key receptively.



(a) 1st order t-test (b) 1st order CPA

Fig. 6. First-order leakage analysis of synthetic data. Left: first-order paired t-test. Right: first-order CPA; Red line corresponds to the correct key guess



(a) 2nd order t-test (b) 2nd order CPA

Fig. 7. Second-order leakage analysis of synthetic data. Left: second-order paired t-test. Right: second-order CPA; Red line corresponds to the correct key guess

Synthetic samples and leakage model First, we generate noise free synthetic leakage samples of the 2-TI Present S-box based on Hamming weight model. As shown in Section 5, a 2-TI S-box processes two shares (4 bits for each share) in parallel and hence we use the Hamming weight of both output shares (8 bits in total) as the synthetic leakage samples. Further, in order for a second order analysis, the synthetic data should be center-and-then-squared. With respect to the leakage model, we use the Hamming weight of the regular S-box output which equals the bitwise XOR between the two output shares in the 2-TI S-box.

First-order analysis We perform first-order *non-specific paired t-test* on the synthetic data and attempt to exploit any leakage using classic CPA as well. For this purpose, 1 million synthetic leakage samples for random input plaintext are generated as well as another 1 million for fixed inputs. The result of t-test using the 2 million samples is shown in Figure 6(a) where the t value is less than 2 as the number of traces (synthetic samples) increases to 2 million. Then, a classic first-order CPA is performed on the 1 million samples associated with the random inputs using the above-mentioned leakage model. The results in Figure 6(b) shows the correct key cannot be distinguished from the wrong key hypotheses with as much as 1 million samples and the attacks fail.

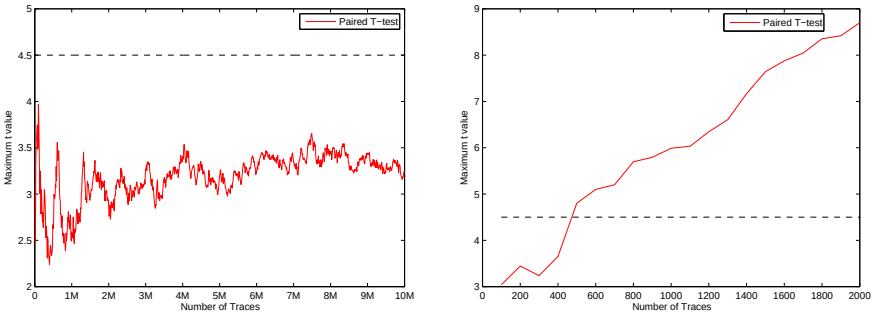
Second-order analysis Then, we proceed with second-order *non-specific paired t-test* and CPA. For this purpose, 200 synthetic leakage samples for random input plaintext are generated as well as another 200 for fixed inputs.Figure 7(a) shows that t value exceed 4.5 with only a couple of hundreds of samples while classic CPA can recover the correct key with less than a hundred samples as shown in Figure 7(b).

In summary, the theoretical analyses also show the first-order resistance of 2-TI scheme but reveals a strong second-order leakage. This strong second-order leakage is caused by the differing variances, as pointed out in Section 3.1. Note that we use perfect Hamming weight model for synthetic data without adding any noise. Hence, the CPA with a Hamming weight model can efficiently recover the key because it captures the leakage well. In fact, CPA on a perfect Hamming weight leakage is comparable to a profiled attack, in the absence of noise. But in the real world, actual leakages are more complex and CPA with Hamming weight model will not be as efficient as in this synthetic scenario. In the following we will conduct analysis on practical implementations to show this.

7.2 Practical Analysis

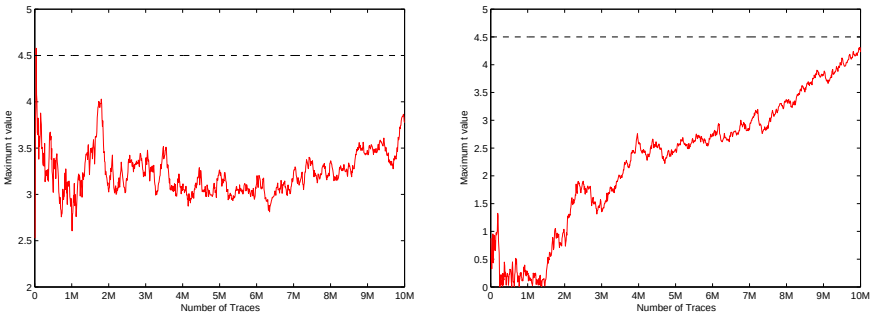
Next, we discuss the leakage analysis results for the two-share implementations of round-based Simon and Present. First, we apply the *non-specific* paired t-test method from [16] to detect any data-dependent leakage. Fixed (F) and random (R) measurements are interleaved using the FRRF pattern. Also, leakage detection tests are performed on round-based 3-TI Simon in order to compare with 2-TI and show the first-order leakage resistance of two-share scheme. Then, classic CPA is performed in order to exploit the second-order leakage detected by t-test and the results comply with the simulations in Section 7.1.

The analyzed implementations are ported into a Virtex-5 xc5vlx50 FPGA on the SASEBO-GII board clocked at 3 MHz. Measurements are taken using a Tektronix DPO-5104 oscilloscope which collects measurements with sample rate of 100 MS/s. The oscilloscope features a *FastFrame* functionality that can capture encryptions in bulk and thus 10 million measurements for each implementation can be taken in several hours.



(a) 1st order t-test (b) 2nd order t-test

Fig. 8. Leakage detection results for the two-share implementation of Simon for first order (left) and second order (right) leakage over the number of traces. Note that the dimensions change for both axes.



(a) 1st order t-test (b) 2nd order t-test

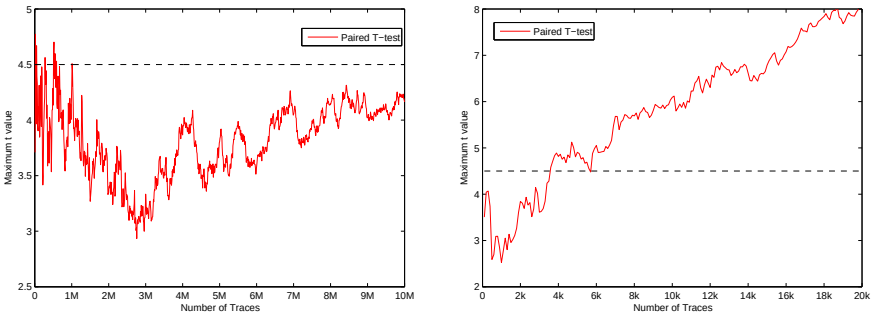
Fig. 9. Leakage detection results for the three-share implementation of Simon for first order (left) and second order (right) leakage over the number of traces.

Round-based 2-TI Simon For two-share Simon implementation, 10 million measurements are collected, yielding 5 million fixed-random pairs. Each measurement contains 5000 time samples, covering the 68 rounds of Simon. The first-order paired t-test is performed using $n = 5000, 10000, 15000, ...$ pairs. Figure 8(a) shows the first order t-test result on the two-share Simon. The maximum absolute t value across the 5000 time samples remains below the threshold of 4.5 with 10 million traces. We conclude that the two-share Simon implementation is resistant against first-order DPA and thus a validly implemented threshold implementation.

The results of the second order paired t-test are shown in Figure 8(b). The step size is reduced to $n = 100, 200, ...$ to magnify the relevant area: The t value of the second order analysis grows beyond 4.5 with about 500 traces. That is, a second order leakage is detectable with just hundreds of traces.

Round-based 3-TI Simon In order to practically compare the performance of 2-TI and 3-TI in resisting first-order and second-order leakage, the paired t-test is also applied to 10 million FRRF measurements from a round-based 3-TI Simon. Figure 9(a) shows similar result as in Figure 8(a) and the t value is below the threshold of 4.5. The comparison shows again that the first-order resistance of 2-TI is solid as a 3-TI. However, 3-TI exhibits resistance against second-order analysis as shown in Figure 9(b) and the t value is still below 4.5 with 10 million traces. That is, given more than 1000x as many measurements as for the 2-TI case, the leakage is just barely detectable. The results comply with the simulation analyses in Section 3.1 and Section 7.1 and validate the weakness of 2-TI.

2-TI Present As before, 10 million traces are captured for the two-share Present implementation, and then analyzed using paired t-test. The first order t-statistic is still below 4.5 with 10 million measurements, as shown in Figure 10(a). The second order t-statistics exceeds the threshold with about 6000 traces as shown in Figure 10(b). Again, the results suggest that two-share TI holds the promise of first order resistance, but fares terribly on the second order resistance.



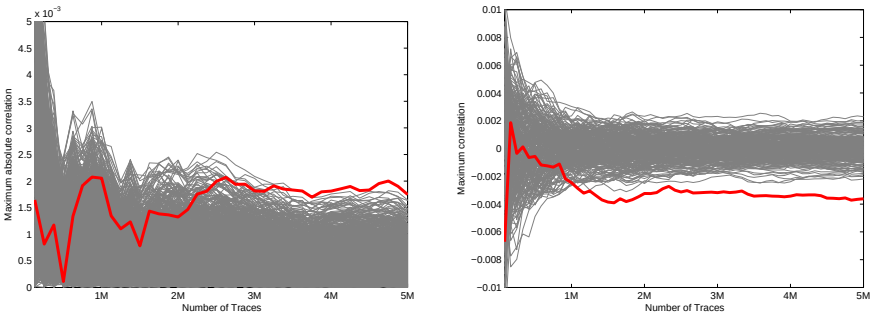
(a) 1st order t-test (b) 2nd order t-test

Fig. 10. Leakage detection results for the two-share implementation of Present for first order (left) and second order (right) leakage over the number of traces. Note that the dimensions change for both axes.

Exploiting the Uncovered Leakages In order to practically exploit this strong second-order leakage, a classic CPA [8, 10, 6] is performed on the measurements (center-and-then-squared) associated with the 5 million random plaintexts.

For *2-TI Simon*, the targeted operations occurred in the first clock cycle of the third round of encryption where shared values in registers L_a and L_b overwrite R_a and R_b respectively (see Figure 1). The leakage model used is Hamming distance between registers L and R as in a plain or unprotected implementation. The reason why third round is chosen is because of the weak non-linearity of single Simon round operation (only one AND) and attacking third round would relieve

the effect of "ghost peaks" [8]. Moreover, in order to reduce the computational complexity, we follow the divide-and-conquer approach and only attack the most significant four bits in L and R which are dependent on 10 bits in k_0 and 4 bits in k_1. Therefore, 2^{14} key hypotheses are required for the attack. To further reduce the complexity, we assume the knowledge of the relevant 4 bits in k_1 is known and only 10 bits in k_0 are aimed at to recover. Figure 11(a) shows the max correlation for each key hypothesis over the number of traces. The practical second-order attack successfully recovers the correct key with more than 3 million measurements even though ghost keys still exist. Note that these results can be significantly improved by using a profiled attack, predicting more bits, and by using a pruning technique as e.g. done in [17], which is always an option for ciphers with a low algebraic depth per round. Nevertheless, the results validate the second-order leakage of two-share TI detected by the t-test and it can be practically exploited.



(a) 2nd-order CPA of two-share Simon (b) 2nd-order CPA of two-share Present

Fig. 11. Second-order CPA. Max correlation for each key hypothesis over the number of traces.

We also performed the same second-order CPA on 5 million random traces (center-and-then-squared) on *2-TI Present*, targeting at the S-box output to exploit the leakage. Recall our 2-TI Present in which the 64-bit state registers are right rotated by 4 bits per clock cycle so that the least significant nibble is continuously fed into the S-box look-up and output is written back to the most significant nibble after 4 clock cycles. Therefore, a Hamming distance leakage occurs between consecutive output nibbles. In this attack, we use the Hamming distance power model between the first two consecutive S-box outputs which depends on the least significant key byte and thus 2^8 key hypotheses are required. The max correlations per key hypothesis over number of traces are shown in Figure 11(b) and the results show that correct key can be successfully recovered with more than 1 million traces which demonstrates the practical exploitability of detected leakage.

The results from both validate our simulation analyses for the idealized case from Section 3.1 and Section 7.1, which suggests strong second-order leakage. The difference in sensitivity for the two implementations stems from their differing design strategies: 2-TI Simon is round based and does not use pipelining. Hence, it maximizes the leakage for the fixed-vs-random test: the entire state that is processed per cycle is constant in the fixed case and varies in the other case. For 2-TI Present, the implementation is serialized, with a 4-bit datapath, hence, a much smaller part of the implementation is updated per cycle, making the leakage less pronounced.

Moreover, unlike the theoretical analysis results in Section 7.1 where the number of traces needed for successful second-order t-test and CPA are of the same order magnitude, a lot more traces are needed for practical second-order CPA with Hamming distance model to exploit the leakage detected by t-test with only hundreds to thousands of traces. This is mainly because: 1) Practical implementation don't leak a perfect Hamming weight or Hamming distance leakage; 2) Noises also render the practical attacks inefficient.

While two-share TI shows potential in preventing first order leakage with less overhead, its poor performance on second order leakage resistance compared with three-sharing makes it less worthwhile.

8 Conclusion

This work presents the first practical threshold implementations using only two shares. We showed that lightweight ciphers have several features making them good targets for threshold implementations. Furthermore, we explain how using two shares can actually yield smaller cipher implementations that need less randomness and still show perfect first order resistance. While moving to two shares makes implementing the nonlinear functions of a cipher more cumbersome, resulting in either a loss in throughput, increase in circuit size, or even both, it allows to reduce the overhead of the sequential part of the implementation by only doubling the state and key size. Since the area of low-area crypto implementations usually depends mainly on the sequential part, significant improvements are possible. In fact, the presented bit-serialized two-share implementation of Simon is the smallest side-channel protected 128-bit block cipher implementation for FPGAs. To this end, we presented the first two-share threshold implementations of Simon and Present, which feature perfect first-order resistance.

However, these findings are of limited practical impact, as two-share TI features strong second-order leakage. Hence, on one hand, the results highlight that provable resistance against a "low" order of attack might be meaningless in practice. On the other hand, the previously observed feature that three-share TI not only keeps the promised first-order resistance, but also fails gracefully for higher order analysis, is undervalued and may deserve further analysis.

Acknowledgments. This work is supported by the National Science Foundation under grant CNS-1261399 and grant CNS-1314770.

References

1. Aysu, A., Gulcan, E., Schaumont, P.: SIMON Says: Break Area Records of Block Ciphers on FPGAs. Embedded Systems Letters, IEEE 6(2), 37–40 (June 2014)
2. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK Families of Lightweight Block Ciphers. IACR Cryptology ePrint Archive 2013, 404 (2013)
3. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Trade-Offs for Threshold Implementations Illustrated on AES. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34(7), 1188–1200 (July 2015)
4. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-Order Threshold Implementations. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014, Springer LNCS, vol. 8874, pp. 326–343 (2014)
5. Bilgin, B., Daemen, J., Nikov, V., Nikova, S., Rijmen, V., Van Assche, G.: Efficient and First-Order DPA Resistant Implementations of Keccak. In: Francillon, A., Rohatgi, P. (eds.) Smart Card Research and Advanced Applications, pp. 187–199. Springer LNCS (2014)
6. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: A More Efficient AES Threshold Implementation. In: Pointcheval, D., Vergnaud, D. (eds.) Progress in Cryptology –AFRICACRYPT 2014, Springer LNCS, vol. 8469, pp. 267–284 (2014)
7. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings. pp. 450–466. Springer Berlin Heidelberg, Berlin, Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-74735-2_31
8. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.J. (eds.) Cryptographic Hardware and Embedded Systems — CHES 2004, Springer LNCS, vol. 3156, pp. 135–152 (2004)
9. Canright, D.: A Very Compact S-Box for AES. In: Rao, J.R., Sunar, B. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2005: 7th International Workshop, Edinburgh, UK, August 29 – September 1, 2005. Proceedings, pp. 441–455. Springer Berlin Heidelberg, Berlin, Heidelberg (2005), http://dx.doi.org/10.1007/11545262_32
10. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Advances in Cryptology – CRYPTO'99. pp. 398–412. Springer (1999)
11. Chen, C., Eisenbarth, T., von Maurich, I., Steinwandt, R.: Masking Large Keys in Hardware: A Masked Implementation of McEliece. In: Selected Areas in Cryptography — SAC 2015. Springer LNCS (August 2015), preprint available at http://eprint.iacr.org/924
12. Cooper, J., DeMulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Rohatgi, P.: Test Vector Leakage Assessment (TVLA) methodology in practice. In: International Cryptographic Module Conference (2013), http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf

13. Coron, J.S., Prouff, E., Rivain, M.: Side Channel Cryptanalysis of a Higher Order Masking Scheme. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings. pp. 28–44. Springer Berlin Heidelberg, Berlin, Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-74735-2_3
14. De Canniere, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN–A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Cryptographic Hardware and Embedded Systems–CHES 2009, pp. 272–288. Springer (2009)
15. De Cnudde, T., Reparaz, O., Bilgin, B., Nikova, S., Nikov, V., Rijmen, V.: Masking AES with $d + 1$ Shares in Hardware. In: Gierlichs, B., Poschmann, Y.A. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2016: 18th International Conference. pp. 194–212. Springer Berlin Heidelberg (2016), http://dx.doi.org/10.1007/978-3-662-53140-2_10
16. Ding, A.A., Chen, C., Eisenbarth, T.: Simpler, Faster, and More Robust T-test Based Leakage Detection. In: Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers. pp. 163–183. http://dx.doi.org/10.1007/978-3-319-43283-0_10
17. Eisenbarth, T., Kasper, T., Moradi, A., Paar, C., Salmasizadeh, M., Shalmani, M.T.M.: On the Power of Power Analysis in the Real World: A Complete Break of the Keeloq Code Hopping Scheme. In: Advances in Cryptology–CRYPTO 2008, pp. 203–220. Springer (2008)
18. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A Testing Methodology for Sidechannel Resistance Validation. Non-Invasive Attack Testing Workshop (2011), http://www.cryptography.com/public/pdf/a-testing-methodology-for-side-channel-resistance-validation.pdf
19. Kavun, E.B., Yalcin, T.: RAM-Based Ultra-Lightweight FPGA Implementation of PRESENT. In: Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on. pp. 280–285. IEEE (2011)
20. Kirschbaum, M., Popp, T.: Evaluation of a DPA-Resistant Prototype Chip. In: Computer Security Applications Conference, 2009. ACSAC '09. Annual. pp. 43–50 (Dec 2009)
21. Kutzner, S., Nguyen, P., Poschmann, A., Wang, H.: On 3-Share Threshold Implementations for 4-Bit S-boxes. In: Prouff, E. (ed.) Constructive Side-Channel Analysis and Secure Design, Springer LNCS, vol. 7864, pp. 99–113 (2013)
22. Leiserson, A.J., Marson, M.E., Wachs, M.A.: Gate-Level Masking under a Path-Based Leakage Metric. In: Batina, L., Robshaw, M. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2014, Springer LNCS, vol. 8731, pp. 580–597 (2014)
23. Moradi, A., Mischke, O.: How Far Should Theory Be from Practice? In: Prouff, E., Schaumont, P. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2012: 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings. pp. 92–106. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-33027-8_6
24. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In: Paterson, K.G. (ed.) Advances in Cryptology — EUROCRYPT 2011, Springer LNCS, vol. 6632, pp. 69–88 (2011)
25. Nikova, S., Rechberger, C., Rijmen, V.: Threshold Implementations Against Side-Channel Attacks and Glitches. In: Ning, P., Qing, S., Li, N. (eds.) Information and Communications Security, Springer LNCS, vol. 4307, pp. 529–545 (2006)

26. Poschmann, A., Moradi, A., Khoo, K., Lim, C.W., Wang, H., Ling, S.: Side-Channel Resistant Crypto for less than 2,300 GE. Journal of Cryptology 24(2), 322–345 (2011)
27. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating Masking Schemes. In: Advances in Cryptology–CRYPTO 2015, pp. 764–783. Springer LNCS (2015)
28. Reparaz, O., Roy, S.S., Vercauteren, F., Verbauwhede, I.: A Masked Ring-LWE Implementation. In: Cryptographic Hardware and Embedded Systems–CHES 2015, pp. 683–702. Springer (2015)
29. Rolfes, C., Poschmann, A., Leander, G., Paar, C.: Ultra-Lightweight Implementations for Smart Devices–Security for 1000 Gate Equivalents. In: Smart Card Research and Advanced Applications, pp. 89–103. Springer (2008)
30. Schneider, T., Moradi, A.: Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In: Güneysu, T., Handschuh, H. (eds.) CHES. Lecture Notes in Computer Science, vol. 9293, pp. 495–513. Springer (2015), http://dblp.uni-trier.de/db/conf/ches/ches2015.html#SchneiderM15
31. Shahverdi, A., Taha, M., Eisenbarth, T.: Silent Simon: A Threshold Implementation under 100 Slices. In: Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on. pp. 1–6 (May 2015)
32. Tiri, K., Verbauwhede, I.: A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In: Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1. pp. 10246–. DATE '04, IEEE Computer Society, Washington, DC, USA (2004), http://dl.acm.org/citation.cfm?id=968878.969036