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. In this work, we explore 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 gains in compactness while loses in throughput compared with three-share schemes. Moreover, the leakage analyses show that two-share TI retains perfect first-order resistance but is shadowed by a strong second-order leakage, making it less worthwhile.

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 [11, 16, 19].

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 [28, 16], 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 [20, 5].

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.

Our contribution In this work we explore the possibility of further 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 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 all prevailing implementations.

While the feasibility of this approach has already been discussed in [23], this work is the first one to explore 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[6] and Simon[2]. Lightweight ciphers are usually a good target for TI, as the algebraic depth of their nonlinear functions is usually quite low.

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.

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. Section 6 presents implementation results and the outcome of the leakage analysis and we conclude at Section 7.

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-mininal 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:

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. Typical examples 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 used 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 [7] 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 [22]. We will refer to this serialization as *horizontal*. While vertical serialization is determined by the cipher at design 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 the cipher state are updated per cycle. This of course increases the latency of the crypto core significantly. However, it also allows to reduce the combinational logic of the crypto core to low single-digit percentages of the entire design [25, 12]. 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 [12]), or, for FPGAs, hiding state and key in dedicated bulk memory such as block RAMs [15] 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 [6]. 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.

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.

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

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, $k_79k_78k_77...k_17k_16$, 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 [6].

2.3 Simon

Simon is one of the two lightweight block ciphers [2] introduced by NSA in 2013. Designed as a Feistel structure, Simon accepts two n-bit words as input plaintext and n could be 16, 24, 32, 48 and 64. For each input size, Simon has a set of allowable key sizes ranging from 64 bits to 256 bits. The number of rounds in Simon ranges from 32 rounds to 72 rounds. For example, Simon128/128 accepts 128 bits of plaintext at a word size of 64 bits and 128 bits of key (two words). It generates a ciphertext after 68 rounds.

Let's denote input words of round i as l_i and r_i, the output words are:

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

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 as bitwise XOR and AND operation respectively. Also, assuming that the input words of the key, which are also the first round keys, are k_0 and k_1 (and possibly k_2 and k_3, depending on the key size), the next round key is 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 [8]. 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 present 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 [21] 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 [22, 20, 4, 5, 27] and even asymmetric crypto algorithms [9, 24] have been successfully protected with TI. Recently, TI has been expanded to protect against higher-order attacks as well, though potential pitfalls of the scheme in the multivariate setting have been pointed out [23].

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 adding randomness by applying an additive Boolean masking. 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 [3] 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.

The paper states that 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. In practice, virtually all implementations try to keep the number of shares at or close to 3, thereby requiring implementations of algebraically more complex functions 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.

2.6 Leakage Detection

Developed by Cryptography Research Inc., Test Vector Leakage Assessment (TVLA) methodology [10] based on Welch's t-test has been recently gaining popularity in detecting potential side channel leakage due to its efficiency and reliability. The test procedures have been well studied in [10] and [26]. 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*). Usually, *non-specific t-test* is more favorable 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 [14] and [18].

In [13], an improved methodology based on paired t-test was suggested due to its robustness against environmental impact on the sample collections. Indeed, today's side channel measurement campaign may take days to collect tens or even hundreds of millions traces and the environmental noise such as temperature fluctuation should not be neglected. In a paired t-test, matched pairs which share the same noise variation are selected from the two sets of measurements. For example, in a *non-specific t-test*, the fixed (F) and random (R) input data are fed into the device according to the sequence $FRRFFRRF...FRRFFRRF$ such that we can always construct a matched pair FR or RF. Even though the sequence is deterministic, the predecessor and successor for each measurement are equally from the either F or R set so that the false-positive bias can be removed. When n such pairs of measurements are constructed, 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 [26]. 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 [13], a moving average which is the average of neighboring traces around each trace is used instead to mitigate the environmental effects.

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. In particular, [17] 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 [20] or variable with up to five shares [5] 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 with just two shares, further reducing the area footprint as well as the need for randomness. This approach was already discussed in [23]. The approach is straightforward for the linear operations of an implementation, and has already been widely used in several TIs for those parts [5, 9]. 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 [23]. However, hardware-minimal implementations break higher-order algebraic functions into degree-minimal building blocks anyway, making share proliferation a purely theoretical concern.

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 [22] 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 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.The following table lists the possible states and the resulting means and variances for both sharings.

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

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, which we will actually demonstrate in the analysis of the reference implementations in Section 6.

4 Application to Simon

Threshold Implementations of Simon with three shares have been proposed in [27] to counteract first-order side channel attacks. Moreover, their parallel 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 Section 6.

4.1 Simon with Two Shares

First of all, we follow the notations used in [27] 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 other 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

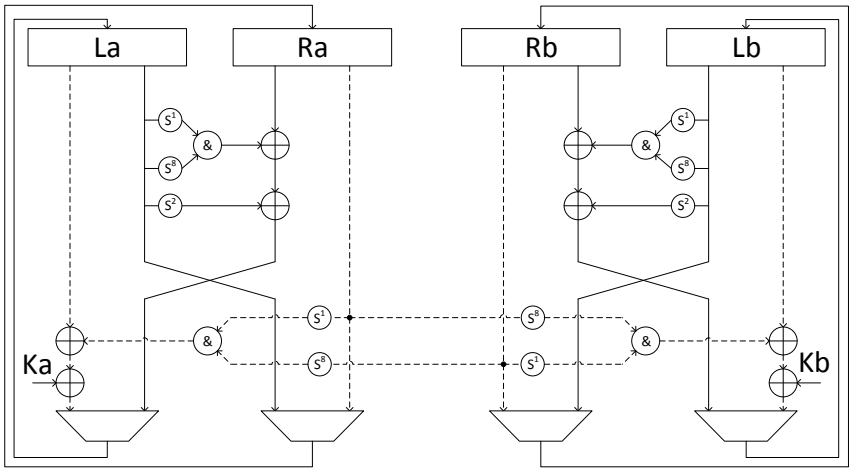


Fig. 1. Data-path of the Simon with Two Shares.

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 FPGA 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 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 dotted line but remember that since $l[j]_i$ is now stored in R_j and hence no extra buffer is needed for it. Note that here we only present a round-based design which processes the whole block each clock cycle while [27] proposed bit-serialized designs which process one bit per clock cycle. It is predictable that our design will be larger, however, our goal is to show that two-share scheme can reduce the overhead compared with three-share TI as demonstrated in Section 6.1. The two-share scheme can also be applied to the bit-serialized design and further decrease the overhead.

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

5 Application to Present

In this section, we apply two-share Threshold Implementation to Present cipher. In [17], the authors presented the 3-TI Present S-box. To achieve this, they need to decompose 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 3-TI. We follow their idea to use the same decomposition but then implement them with 2-TI while still retain the *uniformity*, *non-completeness*, and *correctness*. According to [17], 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 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:

$$\begin{aligned} G_1(x_0, y_0, z_0, w_0, x_1, y_1, z_1, w_1) =& (g_{13}, g_{12}, g_{11}, g_{10}) \\ g_{13} =& x_0 + y_0z_0 + y_0z_1 + y_0w_0 + y_0w_1 \\ g_{12} =& w_0 + x_0y_0 + x_0y_1 \\ g_{11} =& y_0 \\ g_{10} =& z_0 + y_0w_0 + y_0w_1 \end{aligned} \tag{14}$$

$$\begin{aligned} G_2(x_0, y_0, z_0, w_0, x_1, y_1, z_1, w_1) =& (g_{23}, g_{22}, g_{21}, g_{20}) \\ g_{23} =& x_1 + y_1z_0 + y_1z_1 + y_1w_0 + y_1w_1 \\ g_{22} =& w_1 + x_1y_0 + x_1y_1 \\ g_{21} =& y_1 \\ g_{20} =& z_1 + y_1w_0 + y_1w_1 \end{aligned} \tag{15}$$

Obviously, 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.

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

$$\begin{aligned} G_0^2(x_0, y_1, z_0, w_0, g_{03}^1, g_{02}^1, g_{01}^1, g_{00}^1) =& (g_{03}^2, g_{02}^2, g_{01}^2, g_{00}^2) \\ g_{03}^2 =& g_{03}^1 + y_1 z_0 + y_1 w_0 \\ g_{02}^2 =& g_{02}^1 + x_0 y_1 \\ g_{01}^2 =& g_{01}^1 \\ g_{00}^2 =& g_{00}^1 + y_1 w_0 \end{aligned} \tag{17}$$

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

$$\begin{aligned} G_1^2(x_1, y_0, z_1, w_1, g_{13}^1, g_{12}^1, g_{11}^1, g_{10}^1) =& (g_{13}^2, g_{12}^2, g_{11}^2, g_{10}^2) \\ g_{13}^2 =& g_{13}^1 + y_0 z_1 + y_0 w_1 \\ g_{12}^2 =& g_{12}^1 + x_1 y_0 \\ g_{11}^2 =& g_{11}^1 \\ g_{10}^2 =& g_{10}^1 + y_0 w_1 \end{aligned} \tag{19}$$

Where the superscript indicates the level of the circuit. Until now, we achieved a *correct*, *non-complete* and *uniform* two-share $G(.)$.

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. We try to divide the G

function in two parts that it needs less registers between two parts. The second part of the shares use not only the outputs of the first part but also some of the inputs as well. Hence, we divide the two shares of the G function wisely such that it requires the least number of inputs of the first part as inputs for the second part. Three 4 bits registers are used before the second part. Furthermore, due to *non-completeness*, we use another row of registers in between two $G(.)$ functions in the S-box as shown in Figure 2.

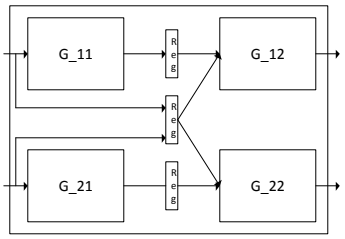


Fig. 2. Hardware architectures of 2-share G module

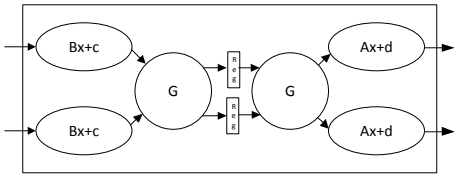


Fig. 3. Hardware architectures of 2-share S-box module

Figure 4 shows the whole Present cipher with two shares. After loading the key in key registers, the 2 shares of the input are stored in 64-bits state registers, then the S-box and permutation operations respectively operate the input to update the state registers for the next round. Regarding the Present algorithm, the result will be ready after 31 rounds.

Considering the hardware design, each $G(.)$ function needs one cycle and then every S-box needs three clock cycles to compute table lookup. According to the Figure 4, each 64-bit input stored in the State register needs to use S-box 16 times, plus one more clock cycle for the permutation operation. Therefore, we need 20 cycles for each round of the Present cipher. The Present cipher has 31 round and thus leads to 620 clock cycles to encrypt a 64-bit input.

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 will be shown in Table 1.

6 Practical Results

6.1 Implementation Results

Table 1 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 using XST.

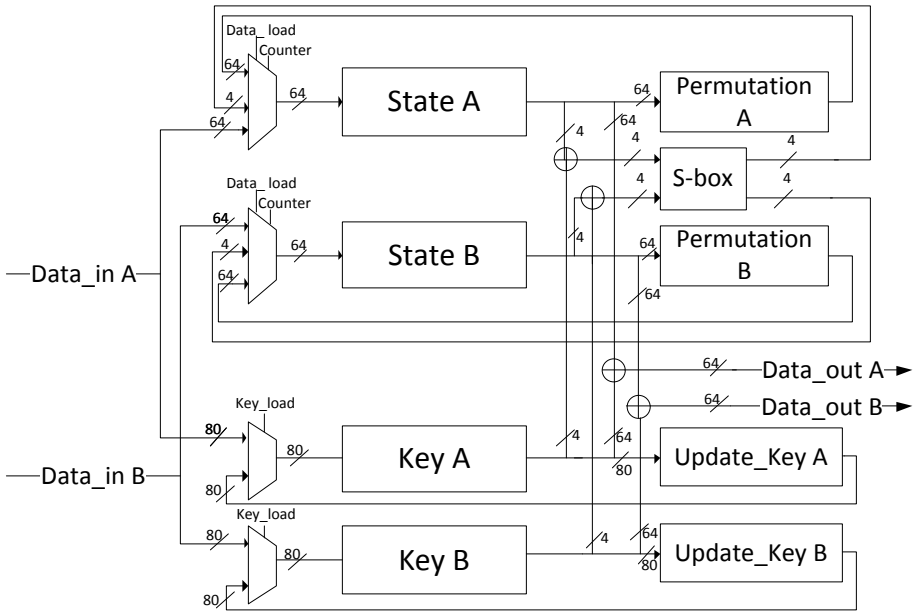


Fig. 4. Hardware architectures of the 2-shares Present Cipher.

For 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. Again, we only use round-based designs to show the advantage of 2-TI and thus the area is not optimal compared with the bit-serialized implementation in [27]. Also, each round operation costs double clock cycles and therefore the throughput is greatly reduced compared with the other two designs.

With respect to Present, we only have two implementations. Two-share implementation costs more resources due to the doubling of inputs as well as other modules in Present cipher; however the results show that the resources used are more than double of the unprotected Present. This is because we should use extra registers to guarantee the *non-completeness* of two-share Present cipher. For example, we use extra registers in $G(.)$ function as explained in Section 5.

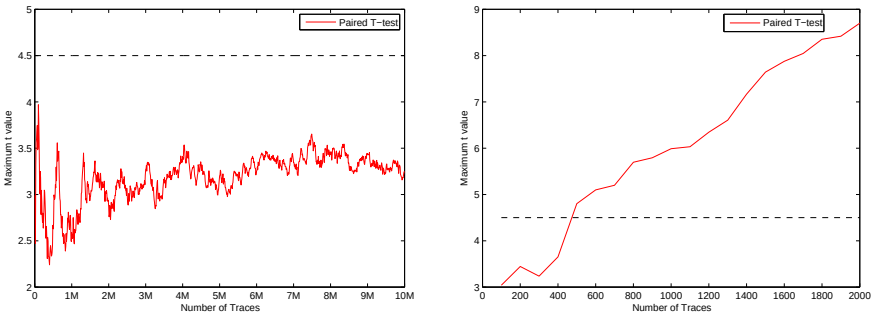
6.2 Leakage Analysis

Next, we discuss the leakage detection results for the two-share implementations of Simon and Present. We apply the *non-specific* paired t-test method

Design	Slice (Regs)	Slice (LUTs)	Max. Frequency (MHz)	Throughput (Mbps)
3-TI Simon	777	1302	414	779
2-TI Simon	520	1169	382	360
Unprotected Simon	272	473	421	792
2-TI Present	362	742	490.252	50.61
Unprotected Present	154	234	394.563	40.73

Table 1. Implementation results of two-share Simon and Present.

from [13]. Fixed (F) and random (R) measurements are interleaved using the FRRF pattern. For second-order analysis, local averages are computed using a sliding window of 100 traces. 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 *Fast-Frame* functionality that can capture encryptions in bulk and thus 10 million measurements for each implementation can be taken in several hours.



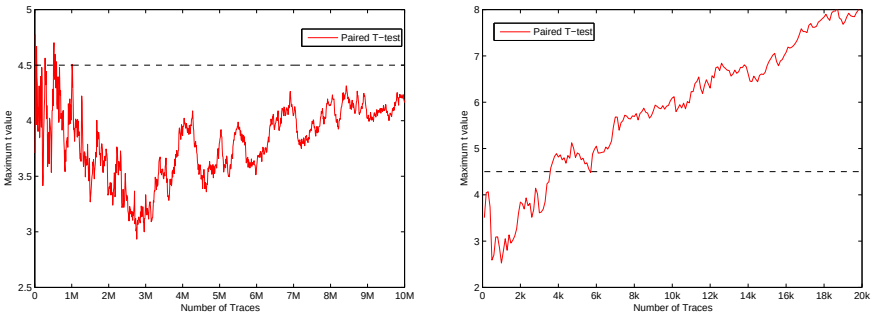
(a) 1st order t-test (b) 2nd order t-test

Fig. 5. 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.

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 5(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 5(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.



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

Fig. 6. Leakage detection results for the two-share implementation of Present for first order (left) and second order (right) leakage over the number of traces.

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 6(a). The second order t-statistics exceeds the threshold with about 6000 traces as shown in Figure 6(b). Again, the results suggest that two-share TI holds the promise of first order resistance, but fares terribly on the second order resistance.

The results from both validate our simulation analysis for the idealized case from Section 3.1, 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.

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.

7 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 implementations usually depends mainly on the combinational part, significant improvements are possible. 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 little practical impact, as two-share TI features a glaring 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.

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.: Higher-order threshold implementations. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology ?ASIACRYPT 2014, Springer LNCS, vol. 8874, pp. 326–343 (2014)
4. 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)
5. 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)
6. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: Cryptographic Hardware and Embedded Systems - CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings, chap. PRESENT: An Ultra-Lightweight Block Cipher, pp. 450–466. Springer Berlin Heidelberg, Berlin, Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-74735-2_31
7. Canright, D.: Cryptographic Hardware and Embedded Systems – CHES 2005: 7th International Workshop, Edinburgh, UK, August 29 – September 1, 2005. Proceedings, chap. A Very Compact S-Box for AES, pp. 441–455. Springer Berlin Heidelberg, Berlin, Heidelberg (2005), http://dx.doi.org/10.1007/11545262_32

8. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Advances in CryptologyCRYPTO?9. pp. 398–412. Springer (1999)
9. 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
10. 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
11. Coron, J.S., Prouff, E., Rivain, M.: Cryptographic Hardware and Embedded Systems - CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings, chap. Side Channel Cryptanalysis of a Higher Order Masking Scheme, pp. 28–44. Springer Berlin Heidelberg, Berlin, Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-74735-2_3
12. De Canniere, C., Dunkelman, O., Knežević, M.: Katan and ktantana family of small and efficient hardware-oriented block ciphers. In: Cryptographic Hardware and Embedded Systems-CHES 2009, pp. 272–288. Springer (2009)
13. Ding, A.A., Chen, C., Eisenbarth, T.: Simpler, faster, and more robust t-test based leakage detection. accepted at COSADE 2016 (2016), preprint available at http://ia.cr/2015/1215
14. 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
15. 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)
16. 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)
17. 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)
18. 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)
19. Moradi, A., Mischke, O.: Cryptographic Hardware and Embedded Systems – CHES 2012: 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings, chap. How Far Should Theory Be from Practice?, pp. 92–106. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-33027-8_6
20. 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)
21. 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)

22. 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)
23. 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)
24. 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)
25. 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)
26. Schneider, T., Moradi, A.: Leakage assessment methodology - a clear roadmap for side-channel evaluations. In: Gneysu, 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
27. 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)
28. 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