More Efficient Private Circuits II Through Threshold Implementations

Thomas De Cnudde  
KU Leuven, ESAT-COSIC  
and iMinds, Belgium  
Email: thomas.decnudde@esat.kuleuven.be

Svetla Nikova  
KU Leuven, ESAT-COSIC  
and iMinds, Belgium  
Email: svetla.nikova@esat.kuleuven.be

Abstract—Since the introduction of Private Circuits at CRYPTO 2003, several works have attempted its implementation in hardware. Only very recently was an implementation of this masking scheme shown to survive state-of-the-art leakage detection tests. The overhead introduced to achieve the provable security was significant. Similarly, the implementational aspect of Private Circuits II, the tamper-resistant extension of Private Circuits presented at EUROCRYPT 2006, was only recently presented at RECONFIG 2015. It however relied on a combinational PC-I implementation, which is susceptible to both glitches and early evaluation.

In this work, we evaluate a recently proposed Private Circuits implementation and its corresponding Threshold Implementation side by side and give a full comparison in an equal and fair setting. In succession, we take the smallest resulting masking scheme as basis for a new approach towards a secure PC-II implementation. In addition to quantifying the resource overhead of PC-II, our work provides detailed instructions on how to achieve PC-II in FPGAs.

Keywords: Threshold Implementations, Private Circuits, DPA, FA, Masking, Countermeasure.

I. INTRODUCTION

The presence of symmetric cryptography in embedded systems is necessary to achieve confidentiality and integrity for both the users and their data. While modern ciphers offer strong, computationally unbreakable security on the algorithmic level, many of their implementations are susceptible to physical attacks. Physical attacks exploit characteristics of the implementation itself in order to retrieve secret keys with much more ease compared to cryptanalytic or brute-force attacks.

Two well known classes of physical attacks are Side-Channel Analysis (SCA) and Fault Attacks (FAs). In SCA, a passive adversary observes e.g. the time duration [1], power consumption [2] or electromagnetic emanation [3] of the cryptosystem in an attempt to obtain information about intermediate values of the computed algorithm. Popular approaches to mount such attacks are Differential Power Analysis (DPA) [2] and Correlation Power Analysis (CPA) [4]. FAs on the other hand, requires an active adversary, that is, the adversary tampers with the device and exploits its faulty behavior [5]. An attack that illustrates the power of FAs is Differential Fault Analysis (DFA) [6]. SCA and FA can be combined to form even more powerful attacks [7]–[11].

Countermeasures are required to harden embedded systems against such attacks. Similar to the classification of the attacks, countermeasures have been predominantly researched separately. For SCA, a popular and widely implemented countermeasure is masking [12], [13]. It provides provable security [14] by randomizing the computed intermediates on the algorithmic level in order to decouple secret values from their side channels. Countermeasures against FAs generally rely on some form of redundancy. Either a given computation is repeated (time redundancy) or the same operation is computed in parallel (area redundancy) in order to check whether any fault was injected [15].Appending error correction or detection codes to the intermediate values form another option for increasing the system’s security [16]. Another approach is to rely on ad hoc countermeasures, e.g. shielding parts of an Integrated Circuit (IC) to prevent optical attacks [17]. In case tampering is detected, an alarm can be triggered to withhold the faulty ciphertexts from being output.

An interesting approach towards provable resistance against both passive and active attacks is Private Circuits II (PC-II) [18]. It is an extension of Private Circuits (PC-I) [19], on which it relies to obtain protection from passive attacks.

A practical drawback of Private Circuits I and II is that ideal gates are required, i.e. gates that evaluate only once and in the right order. In contrast, Threshold Implementations (TI) [20]–[23], another popular masking scheme, does not have this requirement, rendering them especially interesting for applications in standard CMOS logic, where gates leak from their non-ideal behavior through glitches and early evaluation [24].

Our first research question is how these two approaches compare when considering FPGA platforms. From these results we move on to describe how to securely implement Private Circuits II in FPGAs. Our results are applied on the lightweight PRESENT block cipher [25].

A. Related Work.

Several works have considered the implementational aspects of Private Circuits. While the introduction of Private Circuits and Private Circuits II date from 2003 and 2006 respectively, their most notable implementation-related aspects have only recently been published. We now give a brief overview of these different advances.

In several works, Park et al. proposed optimizations for Private Circuits [26], [27]. The resulting implementations were later shown to have security flaws [28]. At ICCD 2015,
Roy et al. [29], pointed out the reasons for the existent side-channel leakage. Undesired circuit optimization by the design flow tools, glitches and early propagation of signal values all had a noticeable impact on the security of the masked implementations. By registering the output of every gate, a fully sequential implementation of Private Circuits was established as a secure but expensive solution.

At CRYPTO 2015, Reparaz et al. [30] related Private Circuits and Threshold Implementations to formalize theoretical foundations of hardware masking schemes. Additionally, a strategy for secure Private Circuits implementations in the presence of glitches is given by inheriting the non-completeness property from TI. Their strategy provides a cheaper and more efficient PC-I implementation compared to the fully sequential realization of [29].

A first move towards PC-II implementations was made by Rakotomalala et al. [31] at RECONFIG 2015. In summary, the authors implement PC-II with tamper-resistance against reset attacks on an FPGA. Their design is based on a manually coded, fully combinational PC-I implementation followed by a PC-II encoding using FPGA specific primitives. In addition, an assessment of the security against critical path violations is provided.

B. Contribution.

Our main contribution is a novel approach towards PC-II. Instead of relying on PC-I as base for PC-II, we substitute this first transformation in the chain by the Threshold Implementation masking scheme, which provides security without the need for ideal gates. This path leads to the first PRESENT implementation resisting both side-channel and fault attacks.

During the elaboration of our method, we provide several auxiliary contributions. We perform the SCA evaluation of the more efficient PC-I implementation proposed by Reparaz et al. that emerged from reasoning with the non-completeness property on the intermediate values appearing in the gadgets [30]. This leads to a direct area comparison of PC-I and TI, and follows the theoretical comparison made at CRYPTO 2015 [30]. While applying the PC-II extensions over TI, we detail every step taken to provide a guide for future PC-II implementations.

C. Paper Organization.

In Section 2, we give the necessary theoretical background w.r.t. PC-I, PC-II and TI. We compare PC-I and TI applied on the PRESENT S-box in Section 3. Based on the area results, we choose TI to mask the full PRESENT implementation and we give the result of the leakage detection test. We detail our approach to PC-II and contemplate on the cost of our implementations in Section 4. We draw conclusions and propose directions for future work in Section 5.

II. PRELIMINARIES

We now give an overview of the relevant background. Due to the vast information given in the original Private Circuit works [18], [19], we only provide the information that we use throughout this paper. Before recounting the masking schemes, we provide the notation and definitions we employ.

Lower-case characters represent elements in $\mathbb{GF}(2^m)$. These elements are split in $s$ shares $a = (a_1, a_2, ..., a_s)$ using Boolean masking. The sharing is said to be uniform if, say the first, $s-1$ shares are assigned from a uniform random distribution and the final share is chosen to satisfy $a_s = a \oplus_{i=1}^{s-1} a_i$. Upper-case characters denote functions $F : \mathbb{GF}(2^m) \to \mathbb{GF}(2^m)$. A function $F(a) = b$ is shared into component functions $F = (F_1, F_2, ..., F_s)$ such that functional equivalence or correctness is maintained, i.e. $b = \bigoplus_{i=1}^s F_i(a)$.

We adopt the $d$-probing adversary model. In this model the adversary is allowed to probe $d$ wires in a circuit per clock cycle. Each probed wire reveals information about all values that occur on that wire during a computation, even temporary or intermediate ones. We modify the $d$-probing adversary of [19] slightly by not allowing adaptive probes, i.e. we do not allow the attacker to move probes within a clock cycle nor over clock cycle boundaries. While this might appear to reduce the security, the ability of our adversary to observe all intermediate values on a wire results in a more flexible model. This keeps the security of our implementation unaffected from effects such as glitches in CMOS.

A. Private Circuits

A Private Circuit [19] distinguishes between two transformations to achieve $d$-probing security. Both transformations retain the correctness of the original circuit.

A first, stateless transformation considers a circuit $C$ as a directed acyclic graph where vertices are Boolean gates and the edges are wires. A second, stateful transformation allows the inclusion of memory elements, or registers and extends the graph to contain cycles as long as every cycle traverses at least one register. The addition of an initial state $s_0$ and external input and output wires completes the definition. We elaborate on the latter transformation, which we refer to as $T_{ISW}^{(d)}$, since it is more practical in the context of cryptosystems.

The transformation of a stateful circuit $C[s_0]$ to a $d$-probing secure circuit $C'[s_0]$ is achieved through the following steps:

1) Input Encoding. An input bit $a$ of the unprotected circuit $C$ is transformed into a set of input bits $a$ to $C'$ by means of uniform Boolean masking.

2) Gates Encoding. The $T_{ISW}^{(d)}$ construction relies on replacing the universal NOT and AND gates in $C$ to NOT and AND gadgets in $C'$. In addition, we describe the well-known XOR gadget since it leads to more efficient implementations.

1) This stronger adversary can be accommodated by adopting the Refreshing Layer of [30], we however do not consider this approach due to the resulting increase in implementation cost.

2) Throughout the paper, we describe $T_{ISW}^{(d)}$ using $s = 2d + 1$ shares, but $s = d + 1$ shares are sufficient after a small modification [19].

3) $T_{ISW}^{(d)}$ can achieve a higher efficiency when applied on larger Field multiplications instead of single AND gates [32].
• **NOT Gate.** Transforming the NOT gate in $C$ to a NOT gadget is achieved by inverting any uneven number of shares of the input $a = (a_1, a_2, \ldots, a_s)$.

• **AND Gate.** The AND operation $c = F(a, b) = ab$ in $C$ is transformed into an AND gadget which performs the following sequential steps:

  a) First, random bit values $r_{i,j}$ with $i \neq j$ and $1 \leq i \leq 2d + 1$ are generated

  b) Then, $T_{r_{i,j}} = (r_{i,j} \oplus a_i b_j) \oplus a_j b_i$ with $i \neq j$ and $1 \leq i \leq 2d + 1$ are computed

  c) Now, the final output bits are computed as $c_i = a_i b_j \oplus (\oplus_{j \neq i} r_{i,j})$, with $1 \leq i \leq 2d + 1$ and $1 \leq j \leq 2d + 1$

• **XOR Gate.** The XOR gate $c = F(a, b) = a \oplus b$ is trivially transformed to an XOR gadget that implements $c_i = a_i \oplus b_i$, $i \in \{1, \ldots, s\}$ on the individual shares

3) **Register Encoding.** Each register value $x$ in $C$ is stored in $C'$ in its encoded form $E_{2d}(x)$. After passing through a circuit of gadgets, the next state of the registers is still encoded and stored at the next clock cycle.

4) **Output Decoding.** An output bit is reconstructed from the bit shares by unmasking $c = \oplus c_i$.

### B. Private Circuits II

At EUROCRYPT 2006, Private Circuits were extended to protect against an adversary capable of modifying values anywhere in a circuit, on an unbounded number of individual wires between logical gates [18]. In contrast to other FA countermeasures [33] no part of the circuit needs to be completely free from tampering.

Private Circuits II come in two styles:

1) Tamper resistance against an unbounded number of adaptive reset-only wire faults

2) Tamper resistance against a bounded number $t$ of arbitrary wire faults (set, reset or toggle) per clock cycle

In both constructions, Private Circuits are used as an essential building block to achieve (optionally) self-destructing circuits which in addition to fault attacks, also resist probing attacks. Much like PC-I, the constructions rely on circuit transformations. However, two independent transformations are now carried out: one for the circuit and one for the data. The starting point is the transformation $T_{ISW}^{(d)}$ or PC-I as described earlier, where $2d + 1$ is the lower bound on the number of used shares.

An overview of both PC-II constructions is now given.

1) **Tamper Resistance Against Unbound Reset Attacks on Wires:** Reset faults on wires are the only type of fault the adversary is allowed to inject. In each clock cycle, any number of wires can be set to zero.

Two transformations $T_{1}^{(d)}$ and $T_{2}^{(d)}$ are applied to provide the tamper-resistance. $T_{1}^{(d)}$ converts a circuit to an encoded private circuit in two steps. In the first step, $T_{ISW}^{(d)}$ is applied to obtain a private circuit. Afterwards, a Manchester encoding is applied to all the data: bit 0 is mapped to the pair of bits 01 and bit 1 is mapped to 10. After this step, all wires and registers are doubled. The circuit acting on this encoded data is transformed by the following steps:

1) **Input Encoding.** All input data passes through an encoding gadget which outputs the pair (Input, $\neg$Input) using a NOT gate.

2) **Gates Encoding.** The gates of the $T_{ISW}^{(d)}$ transformed circuit are replaced by the $T_{1}^{(d)}$ gadgets listed in Table I.

3) **Error Cascading.** Before values are registered or output, its wires go through an error cascading gadget. This way, detected errors will propagate and erase the data in the circuit and at the output. An error is detected when the invalid encoding 00 (denoted as $\perp$) occurs in a wire pair. Note that the circuit is so designed that the other invalid encoding 11 can not arise from reset attacks alone. This self-destruction is achieved by following the structure shown in Figure 1, where the Error Cascading gadget is listed in Table I. We recall that this stage is optional [18].

4) **Output Decoding.** The final output of the circuit is decoded by ignoring the second (negated) wire in the encoded bit.

| Table I  
| TRUTH TABLES FOR THE GADGETS OF $T_{ISW}^{(d)}$ |
| Input $a$ | Input $b$ | Output $c = ab$ |
| 01         | 01         | 01              |
| 01         | 10         | 01              |
| 10         | 01         | 01              |
| 10         | 10         | 10              |
| ...        | ...        | 00              |

| Input $a$ | Output $c = a \oplus b$ |
| 01         | 01         |
| 01         | 10         |
| 10         | 01         |
| 10         | 10         |
| ...        | 00         |

| Input $a$ | Input $b$ | Output $c = \neg a$ |
| 00         | 00         |
| 01         | 10         |
| 10         | 01         |
| 11         | 11         |

| Input $a$ | Input $b$ | Output $c$ | Output $d$ |
| 01         | 01         | 01       | 01       |
| 01         | 10         | 01       | 10       |
| 10         | 01         | 10       | 01       |
| 10         | 10         | 10       | 10       |
| ...        | ...        | 00       | 00       |
Random Number Generator (PRNG) circuit required, this is achieved by encoding the outputs of a Pseudo Random Number Generator (PRNG) circuit \( C \) with seed \( D \) and \( n \) output bits. This circuit is then instantiated \( 2d + 1 \) times and given different values for the seeds. The \( n \) encoded randomness gates are then replaced by the XOR of all outputs of the \( 2d + 1 \) PRNG outputs.

2) Tamper Resistance Against General Attacks on Wires:
In this more general model, the adversary can set the values in any of the circuit wires to 0 or 1, as well as toggle its value. A limit \( e \) on the number of newly targeted wires per clock cycle is imposed but the attacker is still allowed to release the perturbations without restrictions on the number or time of release. Hence, persistent or permanent faults are only counted on their introduction in the circuit.

The transformation \( T_2^{(d)} \) follows a similar approach to the reset-only case.

The first step consists of applying \( T_1^{(d)} \) and is followed by encoding each bit and introducing an error cascading stage. The second step replaces all encoded randomness by a PRNG circuit similar to the reset-only case. These actions are formalized as:

1) **Input Encoding.** The encoder transforms bit value 0 to \( 0^{2se} \) and 1 as \( 1^{2se} \), where \( s = 2d + 1 \) and \( e \) is the limit on faults the attacker can inject. All other values are considered invalid (⊥). Furthermore, a special value \( \bot^s \) is defined as \( 0^{se}1^{se} \).

2) **Gates Encoding.** The gates of the \( T_1^{(d)} \) transformed circuit are replaced by the \( T_1^{(d)} \) gadgets of which the truthtables are listed in Table II. Their implementation is of the form OR of ANDs of the input wires or their NOTs, or the NOT of such a circuit. The NOT gates used are reversible, so that faults occurring on the output side of a NOT gate propagate to the input side.

3) **Error Cascading.** This stage is the same as the one used in the reset-only model but with the error cascading gadget described in Table II to detect and propagate \( \bot^s \) values.

4) **Output Decoding.** The final output can be decoded by ignoring all but the first of the \( 2se \) wires.

---

![Figure 1. Error Cascading Stage](image)

<table>
<thead>
<tr>
<th>Table II</th>
<th>TruthTables for the gadgets of ( T_1^{(d)} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND Gadget</strong></td>
<td>Input ( a )</td>
</tr>
<tr>
<td>( 0^{2se} )</td>
<td>( 0^{2se} )</td>
</tr>
<tr>
<td>( 0^{2se} )</td>
<td>( 1^{2se} )</td>
</tr>
<tr>
<td>( 1^{2se} )</td>
<td>( 0^{2se} )</td>
</tr>
<tr>
<td>( 1^{2se} )</td>
<td>( 1^{2se} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><strong>XOR Gadget</strong></td>
<td>Input ( a )</td>
</tr>
<tr>
<td>( 0^{2se} )</td>
<td>( 0^{2se} )</td>
</tr>
<tr>
<td>( 0^{2se} )</td>
<td>( 1^{2se} )</td>
</tr>
<tr>
<td>( 1^{2se} )</td>
<td>( 0^{2se} )</td>
</tr>
<tr>
<td>( 1^{2se} )</td>
<td>( 1^{2se} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><strong>NOT Gadget</strong></td>
<td>Input ( a )</td>
</tr>
<tr>
<td>( 0^{2se} )</td>
<td>( 1^{2se} )</td>
</tr>
<tr>
<td>( 1^{2se} )</td>
<td>( 0^{2se} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><strong>Error Cascading Gadget</strong></td>
<td>Input ( a )</td>
</tr>
<tr>
<td>( 0^{2se} )</td>
<td>( 0^{2se} )</td>
</tr>
<tr>
<td>( 0^{2se} )</td>
<td>( 1^{2se} )</td>
</tr>
<tr>
<td>( 1^{2se} )</td>
<td>( 0^{2se} )</td>
</tr>
<tr>
<td>( 1^{2se} )</td>
<td>( 1^{2se} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

C. **Threshold Implementations**

Threshold Implementation (TI) is another way to achieve provable security against \( d \)-th order DPA attacks. By making minimal assumptions on the underlying hardware, unbounded security is attained even in the presence of glitches [23]. While Private Circuits act (by convention) on single logic gates, e.g. AND, XOR and NOT gates, Threshold Implementations can be seen as acting on whole functions. For this reason, more flexibility w.r.t. the randomness consumption and a more efficient secure implementation can be accomplished.

The transformation of a circuit \( C \) to a Threshold Implementations [20]–[23], [34] \( C^\ell \) is obtained as follows.

1) **Input Encoding.** Each value (or wire\(^5\)) is split into \( s \) shares\(^6\) using uniform Boolean masking.

2) **Function Encoding.** The function \( y = F(x) \) is implemented as a set of component functions \( F = (F_1, F_2, ..., F_s) \) where each \( F_i \) acts on a subset of the vector of input shares \( x \). The shared function \( F \) is required to satisfy the following properties:

- Correctness.

---

\(^5\)The Private Circuits literature addresses probing and faulting using wires. In TI, a single wire translates to an \( n \)-bit bus when the values reside in GF(2\(^n\)).

\(^6\)As with Private Circuits, security can also be achieved using \( s = d + 1 \) shares.
• $d^{th}$-order non-completeness. Any combination of up to $d$ component functions $F_i$ of $F$ is independent of at least one input share
• Uniform sharing of functions. The security of cascaded nonlinear functions relies on this property. The outputs of the component functions should be constructed to satisfy uniformity, just as the inputs do. Several ways to achieve this property have been proposed [34]–[39]

3) Output Decoding. Similar to $T_{SW}^{(d)}$, the output value is reconstructed from the shares by unmasking $c = \bigoplus c_i$.

III. THE MASKING PROCESS

In this section, a comparison is made between Private Circuits and Threshold Implementations. We implement the PRESENT S-box using both masking schemes and apply leakage detection tests to assess their security. We limit the discussion to first-order secure masking with leakage detection tests to assess their security. We limit the discussion to first-order secure masking with $s = 2d+1$ shares.

The PRESENT S-box performs the substitution $S(x)$ given in Table III.

A. Masking with Threshold Implementations

A Threshold Implementation of PRESENT was proposed by Poschmann [35]. To implement the S-box $S(x)$, a decomposition $S(x) = F(G(x))$ is used to reduce the algebraic degree from three to two. The two new S-boxes $G$ and $F$ are given in Table III and their (unmasked) equations are listed below, with $x = (x_3, x_2, x_1, x_0)$ where $x_3$ is the Most Significant Bit (MSB).

$$G(x_3, x_2, x_1, x_0) = (g_3, g_2, g_1, g_0)$$

$$g_3 = x_0 \oplus x_1 \oplus x_2$$

$$g_2 = 1 \oplus x_1 \oplus x_2$$

$$g_1 = 1 \oplus x_3 \oplus x_1 \oplus x_0x_2 \oplus x_0x_1$$

$$g_0 = 1 \oplus x_0 \oplus x_2x_3 \oplus x_1x_3 \oplus x_1x_2$$

$$F(x_3, x_2, x_1, x_0) = (f_3, f_2, f_1, f_0)$$

$$f_3 = x_3 \oplus x_1 \oplus x_0 \oplus x_3x_0$$

$$f_2 = x_3 \oplus x_1x_0$$

$$f_1 = x_2 \oplus x_1 \oplus x_3x_0$$

$$f_0 = x_1 \oplus x_2x_0$$

For the shared version of these equations, we refer to the original work [35].

In order to guarantee the uniformity at the input of $F$, the evaluation of the nonlinear functions $G$ and $F$ needs to be separated by registers. The total S-box is then computed in two clock cycles.

B. Masking with Private Circuits

To our knowledge, no PC-I implementation of the PRESENT S-box has been published so far. Therefore, we explore different possibilities to approach its masking.

Before we delve into the full S-box, we describe how to secure an AND gate using PC-I when non-ideal gates are used. This approach was proposed by Reparaz et al. [30]. Security can be obtained while still keeping the integrity of PC-I w.r.t. the operation order by inserting registers. In [29], a secure AND gadget was proposed by inserting registers behind every gate of a manually encoded PC-I AND. The resulting impact on the area and performance is significant.

A different approach for preventing leakage from glitches and early propagation is obtained by reasoning on the non-completeness property of TI. When investigating the different outputs $c_{1\leq i\leq 3}$ in the PC-I AND gadget, it becomes clear that only one layer of registers is required to introduce non-completeness to the scheme.

The AND gadget is then executed in the following two steps:

**Step 1:**

$$c_{1,reg} = a_1b_1 \oplus r_{1,2} \oplus r_{1,3}$$

$$c_{2,reg} = a_2b_2 \oplus a_2b_1 \oplus r_{1,2} \oplus a_1b_2 \oplus r_{2,3}$$

$$[a_3b_3]_{reg} = a_3b_3$$

$$[r_{3,1}]_{reg} = a_3b_1 \oplus r_{1,3} \oplus a_1b_4$$

$$[r_{3,2}]_{reg} = a_3b_2 \oplus r_{2,3} \oplus a_2b_4$$

**Step 2:**

$$c_{1, opt} = c_{1,reg}$$

$$c_{2, opt} = c_{2,reg}$$

$$c_{3, opt} = [a_3b_3]_{reg} \oplus [r_{3,1}]_{reg} \oplus [r_{3,2}]_{reg}$$

Here, we opt for a first-order implementation with pipelining in mind. Therefore, the term $a_3b_3$ gets computed in the first clock cycle to avoid storing the individual bits $a_3$ and $b_3$ separately.

In addition to the advantage of a smaller area, the AND gadget evaluation drops from the 9 clock cycles of [29] to 2. It was shown that PC-I AND gadget can be generalized to nonlinear operations in larger fields [32]. Similarly, the described partitioning of operations can be applied to secure multiplications in larger fields.

We now describe some possible approaches towards a PC-I PRESENT S-box. The first possibility is to use the Algebraic Normal Form (ANF) of the S-box and substitute every AND gate with a PC-I AND gadget. The ANF of $y = S(x)$ is given below, where $x_3$ and $y_3$ denote the MSB.

$$y_3 = x_0x_1x_2 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \oplus x_1x_2 \oplus x_0 \oplus x_1 \oplus x_3 \oplus 1$$

$$y_2 = x_0x_1x_3 \oplus x_0x_2x_3 \oplus x_0x_1 \oplus x_0x_3 \oplus x_1x_3 \oplus x_2 \oplus x_3 \oplus 1$$

$$y_1 = x_0x_1x_2 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \oplus x_1x_3 \oplus x_2x_3 \oplus x_1 \oplus x_3$$

$$y_0 = x_1x_2 \oplus x_0 \oplus x_2 \oplus x_3$$

A total of 23 AND gadgets and 23 XOR gadgets are required when no specific optimizations are applied.

A second possible approach is to use the decomposition $S(x) = F(G(x))$. The required gates reduce to 9 AND...
Table III
4-BIT TO 4-BIT SUBSTITUTION OF THE PRESENT S-BOX [25].

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>S(x)</td>
<td>C</td>
<td>5</td>
<td>6</td>
<td>B</td>
<td>9</td>
<td>0</td>
<td>A</td>
<td>D</td>
<td>3</td>
<td>E</td>
<td>F</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>G(x)</td>
<td>7</td>
<td>E</td>
<td>9</td>
<td>2</td>
<td>B</td>
<td>0</td>
<td>4</td>
<td>D</td>
<td>5</td>
<td>C</td>
<td>A</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>6</td>
<td>F</td>
</tr>
<tr>
<td>F(x)</td>
<td>0</td>
<td>8</td>
<td>B</td>
<td>7</td>
<td>A</td>
<td>3</td>
<td>1</td>
<td>C</td>
<td>4</td>
<td>6</td>
<td>F</td>
<td>9</td>
<td>E</td>
<td>D</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

and 19 XOR gadgets. We choose this approach for our implementation as it requires less resources.

Now our PC-I PRESENT S-box is chosen, we proceed with a comparison of the side-channel security.

C. Measurement Setup

We perform the evaluation of the security by loading our designs separately onto a SASEBO-G board [40]. During synthesis, we select the Xilinx “Keep Hierarchy” option to avoid optimizations over boundaries of the individual shares, we keep all other options to their default value. The SASEBO-G features two Xilinx Virtex-II Pro FPGA devices: an XC2VP7 to hold our cryptographic implementations and an XC2VP30 for controlling the communication between the board, the measurement PC and other equipment. We provide a stable clock of 3MHz to our designs and measure the instantaneous power consumption as the voltage drop over a 1Ω resistor between the ground lines of the crypto FPGA core and the board. We acquire the power traces with a Tektronix DPO 7254C oscilloscope at a sample rate of 1GHz/s. We submit our designs to leakage detection tests [41]–[43], since they provide a very powerful way to test for side-channel resistance.

To show that the security of our designs uniquely stem from a sound masked implementation, we employ the following steps during our evaluation. We first bias the uniformity of the input shares by turning off the PRNG responsible for the initial sharing. In that case, we expect that the leakage detection test will return t-values beyond the confidence interval of ±4.5. We then proceed with the evaluation by turning on the PRNG of our designs. The lack of bias, and thus the correct application of the properties of the masking schemes, is then exclusively responsible for any increase in resistance.

a) PRNG Off: Figure 2 shows the result of the leakage detection tests for the biased PC-I and TI masked S-boxes. With 100k traces, it is clear that the t-value goes beyond the confidence interval of ±4.5 and we conclude that our measurement setup is sound.

b) PRNG On: Figure 3 shows the results of the leakage detection tests with the activated PRNG. While clear leaks are present in the second-order t-test, no first-order t-values fall outside the confidence interval for both the PC-I and TI masked S-box. We conclude that both designs achieve the targeted first-order security with 25 million traces.

Since both masking schemes achieve similar security levels, we compare their implementation cost in Table IV to choose our base transformation for PC-II. The numbers are obtained with Xilinx ISE 10.1.
The resulting TI S-box results in a smaller number of slices compared to the PC-I implementation. The number of flip flops is also lower since only two stages of registers are required, compared to four stages for the PC-I S-box. A direct result from the difference in the gate encoding for Private Circuits versus the function encoding in Threshold Implementations is manifested in both a smaller number of LUTs and a complete elimination of any randomness consumption for TI. TI is clearly the more efficient approach for our goal, hence we proceed by masking the PRESENT cipher with TI in the style of Poschmann [35].

We follow the same approach as with our previous security evaluations for the PRESENT-TI:

c) PRNG Off: The result of the leakage detection tests for the biased PRESENT-TI is shown in Figure 4. With 100k traces, the t-value goes beyond the confidence interval of ±4.5 and we again conclude that our measurement setup is sound.

d) PRNG On: The results of the leakage detection tests on PRESENT-TI with the activated PRNG is shown in Figure 3. Again clear leaks are present in the second-order t-test while no first-order t-values fall outside the confidence interval. Our PRESENT-TI implementation achieves the targeted first-order security with 100 million traces and provides us with a solid foundation for a PC-II implementation.

### IV. APPLYING PC-II

We proceed by applying the remaining PC-II transformations on the PRESENT-TI. While we allow an attacker to only inject one fault \((e = 1)\), the explained process can be extended to any number of fault injections for the general attack model.

#### A. Encoding

Transforming every wire into a wire pair is straightforward. All wires and registers are simply duplicated. In contrast to masking, this duplication has to be applied to all wires, including wires responsible for the control of the data flow. These can be omitted when masking a circuit as they are independent of the plaintext and key data. They can however be faulted to prematurely output ciphertexts of which the key can be trivially retrieved. This is exemplified in Figure 6: activating the ready signal at the start of an encryption would make a key retrieval straightforward for an attacker.

In addition to registers, the shared S-box and the shared permutation Layers, two adders, several multiplexers and comparators are present. An example can be found in the structure of the PRESENT-TI control logic, which is shown in Figure 7. The internal signals of e.g. the adders have to undergo the duplication too. We achieve this by expressing all operations in the Hardware Description Language of the

### Table IV

<table>
<thead>
<tr>
<th></th>
<th>PC-I</th>
<th>TI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slices</td>
<td>107</td>
<td>29</td>
</tr>
<tr>
<td>Number of Slice Flip Flops</td>
<td>166</td>
<td>48</td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
<td>96</td>
<td>57</td>
</tr>
<tr>
<td>Consumed Random Bits</td>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>Number of Clock Cycles</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
PRESENT-TI using atomic AND, OR, XOR and NOT gates. The substitution of these gates by the gadgets from Table I or II then becomes trivial.

B. Gate Transformation

All our elementary gates can now be substituted by their representative gadgets. Since each AND, OR and XOR gadget has four inputs and two outputs, we can not pack a single gadget in one Look-up Table (LUT) of our targeted Virtex-II Pro FPGA. This is possible with more recent technologies, as was done for the Virtex-5 in [31]. Therefore, each gadget requires two LUTs.

In Section 2, it was noted that reversible NOT gates are required for PC-II in the general attack model. Since the reason for this is described in the original work [18], we only give the main idea: by using reversible NOT gates, they can be considered as part of an atomic AND gate. By mapping one gadget output to a single LUT, the whole LUT function can be considered as atomic. With the original assumption that attacks are only possible on wires, we can thus omit the reversible NOT gate requirement.

C. Error Cascading

Error Cascades are nonlinear gadgets that forward the input unless an invalid encoding is detected at one of the inputs. In that case, both its encoded outputs are made invalid ($\perp$).

We provide a toy example to explain how the Error Cascading Stage affects the security. Assume we have a two shared 1-bit value $s = s_1 \oplus s_2$ such that $s_1 = s \oplus r$ and $s_2 = r$, where $r$ is a random bit drawn from a uniform distribution. These two values are passed through an Error Cascade, of which the function is given in Table II. Its underlying circuit is of the form OR of ANDs of all the shares and therefore breaks the non-completeness property.

For the Error Cascading Stage to be effective, it should pass all pairs of wires as shown in Figure 1. This creates a combinational circuit that nonlinearly relates every share, and invalidates the non-completeness property. When using CMOS, glitching and early propagation of values will cause the circuit to leak. Since this stage is stated to be optional [18], we will omit its implementation.

D. Resource Comparison

Table V lists the resources required for our implementations on the Virtex-II Pro 7. Since it has only 4928 slices available, our PC-II designs are too large to fit in the FPGA.

While the PC-I and PC-II version of [31] have the same netlist, and result in a comparable resource consumption, this is not the case for our PRESENT. Instead, we notice a significant overhead for the PC-II implementations compared to the TI version. The first reason is that we did not manually code our TI. The FPGA tools can group several gates in a single LUT to reduce the total number of used slices. The second reason is the architecture of the LUTs. We were not able to just change the LUT configuration to change the single output gates to double output gadgets. A third reason is that we can not use the standard multiplexers, but instead have to create PC-II style multiplexers from LUTs.

The resources for the reset-only and the first-order general attack implementation are exactly the same, as the only change required to go from one PC-II version to the other can be obtained from reconfiguring each LUT to the gadget of either Table I or II.

In order to provide Gate-equivalent figures we would need to synthesize our code, while ensuring that the PC-II assumptions hold. This means that the NOT gates for the General-Attack model should be reversible, or at least, be incorporated in a single cell as in our FPGA specific way. Furthermore, the gadgets should strictly follow the OR of ANDs structure that is required for PC-II [18]. Because it is difficult to satisfy these

![Figure 7. Control Structure for PRESENT-TI](image-url)
requirements using a standard-cell library, we have chosen not to provide Gate-equivalent figures.

E. Fault Attack Simulation

An attractive target wire to inject a fault on is the ready signal of our implementation. The ready signal asserts the output only when the cipher has finished the right number of rounds. By validating this signal at the start of an encryption using fault injection, intermediate states of the cipher become observable and successful cryptanalysis becomes possible.

We now simulate this fault injection on the ready signal on our unprotected and PC-II enhanced PRESENT-TI. Figure 8 shows traces of several signals. The clk signal represents the clock, the start signal indicates the start of the encryption and the ready signal indicates when the output is available. We show the decoded input, key and output shares in decoded_inp, decoded_key and decoded_out respectively. The ready signal in this example is stuck-at-1, i.e. every intermediate state is observable at the output of the cipher. By having the plaintext and the first intermediate result, the key can be obtained without much effort.

In Figure 9 we show the same simulation applied on our PC-II enhanced PRESENT-TI. Since the ready signal and the state values pass through an AND gadget (see Figure 6), the value of its decoded output becomes zero if a fault is present at its inputs. The invalid signal 01 on the encoded ready signal is detected, and the countermeasure succeeds in making the fault unexploitable.

V. Conclusion

In this paper, we considered the implementational aspect of Private Circuits I and II. Previous work has shown that applying PC-I or PC-II is not trivial and prone to many subtle mistakes that can lead to insecure designs. As the traps and pitfalls of PC-I were covered in the work of Roy et al. [29], we direct our focus on PC-II.

In an incremental approach, we first masked the PRESENT S-box with the Private Circuits and the Threshold Implementations masking schemes. Their security was evaluated using leakage detection tests.

After evaluation of the area, randomness and number of clock cycles the different S-boxes require, we chose the most lightweight one to protect PRESENT-80. Threshold Implementations was the leanest and comes with the advantage that no randomness is consumed during an encryption. We implemented the PRESENT-80 TI of Poschmann et al. [35] as base for our subsequent PC-II transformations. After confirmation of its security in our measurement setup, we clarified the process of applying PC-II. Since this process is very subtle, we described every step.

Since our considered PRESENT-80 implementation does not consume any randomness during its execution, we propose the investigation of applying PC-II to a circuit depending on a PRNG as future work.

Acknowledgements

This work was supported in part by NIST with the research grant 60NANB15D346 and in part by the Research Council KU Leuven (OT/13/071). Thomas De Cnudde is funded by a research grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

References


[31] H. Rakotomalala, X. T. Ngo, Z. Najm, J. Danger, and S. Guilley, “Private circuits II versus fault injection attacks,” in International Conference on ReConFigurable Computing and FPGAs, ReConFig 2015, Riviera Maya,


