J. Cryptol. (2011) 24: 322–345
DOI: 10.1007/s00145-010-9086-6

Journal of
CRYPTOLOGY

Side-Channel Resistant Crypto for Less than 2,300 GE

Axel Poschmann
Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore
aposchmann@ntu.edu.sg

Amir Moradi
Horst Görtz Institute for IT Security, Ruhr University Bochum, Bochum, Germany
moradi@crypto.rub.de

Khoongming Khoo and Chu-Wee Lim
DSO National Laboratories, 20 Science Park Drive, Singapore, Singapore
kkhoongm@dso.org.sg; lchuwee@dso.org.sg

Huaxiong Wang and San Ling
Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, Singapore
hxwang@ntu.edu.sg; lingsan@ntu.edu.sg

Received 1 September 2009
Online publication 27 October 2010

Abstract. A provably secure countermeasure against first order side-channel attacks was proposed by Nikova et al. (P. Ning, S. Qing, N. Li (eds.) International conference in information and communications security. Lecture notes in computer science, vol. 4307, pp. 529–545, Springer, Berlin, 2006). We have implemented the lightweight block cipher PRESENT using the proposed countermeasure. For this purpose we had to decompose the S-box used in PRESENT and split it into three shares that fulfill the properties of the scheme presented by Nikova et al. (P. Lee, J. Cheon (eds.) International conference in information security and cryptology. Lecture notes in computer science, vol. 5461, pp. 218–234, Springer, Berlin, 2008). Our experimental results on real-world power traces show that this countermeasure provides additional security. Post-synthesis figures for an ASIC implementation require only 2,300 GE, which makes this implementation suitable for low-cost passive RFID-tags.

Key words. Side-channel attacks, Countermeasures, Secret sharing, Lightweight, ASIC.

1. Introduction

1.1. *Motivation*

Increasingly, everyday items are enhanced to pervasive devices by embedding computing power, and their interconnection leads to Mark Weiser's famous vision of *ubiquitous computing* (ubicomp) [45], which is widely believed to be the next paradigm in information technology. Pervasiveness requires mass deployment, which in turn implies harsh cost constraints on the used technology. The cost constraints imply in particular for application-specific integrated circuits (ASICs) that power, energy, and area requirements must be kept to a minimum. Even Moore's law needs to be interpreted contrarily here: rather than doubling of performance, the price for constant computing power halves each 18 months. This interpretation leads to interesting conclusions, because many foreseen applications require a minimum amount of computing power, but at the same time have extremely tight cost constraints (e.g. RFID in tetra packs). As a consequence, these applications are not realized yet, simply because they do not pay off. Moore's law however halves the price for a constant amount of computing power every 18 months, and consequently enables such applications after a certain period of time. Therefore, a constant or even increasing demand for the cheapest (read lightweight) solutions can be foreseen.

The mass deployment of pervasive devices promises many benefits such as lower logistic costs, higher process granularity, optimized supply chains, or location-based services among others. Besides these benefits, there are also risks inherent in pervasive computing, since many foreseen applications are security sensitive. With the widespread presence of embedded computers in such scenarios, security is a striving issue, because the potential damage of malicious attacks also increases. An aggravating factor is that pervasive devices are usually not deployed in a controlled but rather in a hostile environment, i.e., an adversary has physical access to or control over the devices. This adds the whole field of physical attacks to the potential attack scenarios. Most notable are the *side-channel attacks*, especially *simple*, *differential* and *correlation power analyses* [4,20].

In practice, a complete—i.e., including the analog part—low-cost RFID tag might have between 1,000 and 10,000 GE[1] and for security components only 200–2,000 GE may be available [15]. A major chip manufacturer only considers side-channel resistant implementations for its next generation of security RFID-tags, which due to cost constraints have to be smaller than 3,000 GE. In this article we will show how to tackle this challenging task. We provide implementation details for five architectures and assess their level of side-channel resistance by evaluating real power traces obtained from a field programmable gate array (FPGA)-based side-channel standard platform.

1.2. *Related Work*

Though the topic of lightweight and side-channel resistant implementation is a pressing issue, only a few results that claim to be lightweight have been published so far.

[1] Gate equivalent is a measure for area requirements of integrated circuits (IC). It is derived by dividing the area of the IC by the area of a two-input NAND gate with the lowest driving strength.

Karpinskyy et al. present a masked implementation of mCrypton [22] that requires 6,929 GE for a 64 bit key and 7,446 GE for a more adequate 96 bit key [16]. However, the authors provide no evaluation of the side-channel resistance, which leaves an important question open. Regazzoni et al. have proposed an electronic design automatization (EDA) design flow and evaluation framework for differential power analysis (DPA)-resistant instruction set extensions for embedded processors in [33]. Though they also used PRESENT as an exemplary block cipher, they do not focus on lightweight ASIC implementations. Their proposed flow rather allows the simulation and evaluation of possible trade-offs for embedded processors in a very early design step. Furthermore, they use DPA-resistant logic styles, while we focus on algorithmic countermeasures.

1.3. *Our Work*

First we identify PRESENT [3] as the most suitable lightweight encryption algorithm and the masking scheme of Nikova et al. [27–29] as the most suitable countermeasure for our purposes. Since PRESENT uses an S-box $S(x)$ that is composed of four quadratic and cubic Boolean functions, but the masking scheme requires only quadratic Boolean functions, we first had to decompose the S-box into two S-boxes $F(x)$,$G(x)$ with algebraic degree 2, s.t. $S(x) = F(G(x))$. moderate, ranging from 2,282 GE for a moderately secured implementation up to 3,582 GE for a highly secure implementation. The simulated current consumption for all variants is in the range of single digit µA and thus well suited for passive RFID implementations. To speed up the search, we apply a series of tricks and exploit some properties of Boolean functions so that we can decrease the search space for all possible decompositions from 2^{88} to 2^{26}, which takes a few minutes on a standard PC. Building on these findings, we describe five lightweight hardware architectures that allow a smoothly scalable security-cost trade-off. As a result it turns out that the area requirements are surprisingly moderate, ranging from 2,282 GE for a moderately secured implementation up to 3,582 GE for a highly secured implementation. The simulated current consumption for all variants is in the range of single digit µA and thus well suited for passive RFID implementations. We substantiate our claims for all profiles by a complete side-channel evaluation based on real-world power traces that we obtain from a side-channel attack standard evaluation board (SASEBO). We use a variety of different power models—e.g., Hamming weight (HW) of the S-box input, HW of the S-box output, and (partial) Hamming distance of the state register—to show that our most secure proposal is resistant against first order DPA attacks even if an attacker is capable of measuring 5,000,000 power traces. Furthermore, we use 100,000 measurements to show that if an attacker is not able to profile the device, even our smallest proposal (2,282 GE) provides first order DPA resistance. Our results are the first published first order DPA-resistant implementations that come close to the often-cited 2,000 GE barrier. In practice, however, this barrier is not fixed but rather fuzzy; hence, our implementations are well suited for low-cost passive RFID-tags.

1.4. *Outline*

We first give a brief introduction to *differential power analysis* and countermeasures in the following section. We point out that, especially for low-power lightweight implementations, a strong need for DPA countermeasures exists. A general overview of countermeasures follows a more detailed description of the masking scheme presented in [27, 28], which we use for our experimental evaluation. In Sect. 3 we start with an explanation of why we chose PRESENT as the target algorithm and give a brief description of it. In the remainder of Sect. 3 we explain how to efficiently decompose the PRESENT S-box into two S-boxes and share it using six S-boxes. Based on these findings, in Sect. 4 we propose five different hardware architectures of a serialized PRESENT and mount DPA attacks on its real-world power traces in Sect. 5. Finally we conclude this article in Sect. 6.

2. Introduction to DPA

Side-channel cryptanalysis has emerged as a serious threat for smart cards and other types of pervasive devices performing cryptographic operations. It was demonstrated in a number of publications that side-channel attacks are an extremely powerful and practical tool for breaking unprotected (or insufficiently protected) implementations of cryptosystems. These attacks exploit the fact that the execution of a cryptographic algorithm on a physical device leaks information about sensitive data (e.g., secret keys) involved in the computations.

2.1. *History*

Though these attacks were already discovered accidentally in 1943 [40], it took more than 50 years for the first publication of power analysis attacks to appear in 1999 [20]. Many sources of side-channel information have been discovered in recent years, including the timing characteristics of a cryptographic algorithm [19], as well as deliberately introduced computational faults [1], but most notable are power analysis attacks [20], which evolved to a unique scientific sub-field with an ever-increasing number of publications.

In the context of power analysis, contrary to mathematical cryptanalyses which mostly require pairs of plain- and ciphertexts, knowing either the input or the output of the cipher would be adequate to mount a key-recovery attack. Measuring and evaluating the power consumption captured from a cryptographic device allows for exploiting information-dependent leakage and combining with the knowledge about the plaintext or ciphertext in order to extract the secrets. A *simple power analysis* (*SPA*), which relies on visual inspection of power traces, e.g., measured from an embedded microcontroller of a smart card, aims at recovering a secret by means of (ideally) a single power trace while a *differential power analysis* (*DPA*) utilizes statistical methods and evaluates several power traces belonging to known/chosen different plain- or ciphertexts to recover the secrets [20]. In a *correlation power analysis* (*CPA*), which is a general form of DPA, measurements are compared to estimations obtained by means of a theoretical model which fits to the characteristics of the target implementation [4]. A DPA/CPA requires no knowledge about the concrete implementation of the cipher and can hence be applied to any unprotected black box implementation.

2.2. *Lightweight Implementations and DPA*

Power optimization techniques are an important tool for lightweight implementations of specific pervasive applications. On the one hand they also strengthen implementations against side-channel attacks, because they lower the power consumption (the signal), which decreases the signal-to-noise ratio (SNR). However, on the other hand power saving techniques also *weaken* the resistance against side-channel attacks. One consequence of the power minimization goal is that in the optimal case only those parts of the data path are active that process the relevant information. Furthermore, the width of the data path, i.e., the amount of bits that are processed at one point in time, is reduced by serialization. This however implies that the algorithmic noise is reduced to a minimum, which reduces the amount of required power traces for a successful side-channel attack. Even worse, the serialized architecture allows the adversary a divide-and-conquer approach, which further reduces the complexity of a side-channel attack. Summarizing, it can be concluded that lightweight implementations greatly enhance the success probability of a side-channel attack. The practical side-channel attack [7] on *KeeLoq* applications [18] impressively underlines this observation.

2.3. *Countermeasures*

Several schemes have been proposed to protect cryptographic implementations against power analysis (DPA). A DPA countermeasure aims at preventing a dependency between the power consumption of a cryptographic device and intermediate values of the executed algorithm [24]. *Hiding* and *masking* are among the most common countermeasures on either the hardware or the software level. The goal of *hiding* methods is to increase the noise factor [47] or to equalize the power consumption values [41] independently of the processed data, thereby decreasing the SNR.

Alternative DPA-resistant logic styles, such as SABL [41] and adiabatic logic techniques [17], require a full-custom design flow and cannot be used with semi-custom design tools. As a consequence it is not possible to evaluate the effectiveness of a countermeasure on an FPGA, but manufacturing of an ASIC is required. This is not only an expensive but also a time-consuming task. Moreover, those DPA-resistant logic styles which can be used with semi-custom design tools, e.g., WDDL [42] and MDPL [32], have strong data-dependent leakage which makes them vulnerable to straightforward DPA attacks [31,37]. Therefore, we have focused on countermeasures at the algorithmic level.

2.4. *Masking*

Masking countermeasures rely on randomizing key-dependent intermediate values processed during the execution of the cipher and can be employed on either an algorithmic level [30] or a cell level [32]. Masking is in fact a $(2,2)$ secret sharing scheme [2, 35], where both shares of the secret are required to proceed.

Algorithmic masking schemes often are combined with randomizing the order of the operations, i.e., *shuffling* [13], when applied on microprocessor-based platforms. In spite of the increased efforts required by an adversary, e.g., higher order attacks [44] or sophisticated power models [25], none of the techniques proposed so far can perfectly counteract a key recovery by means of DPA in practice. In short, currently there exists no perfect protection against DPA attacks. However, applying appropriate countermeasures makes the attacker's task more difficult and expensive. Chari et al. have shown in [6] that up to nth order DPA attacks can be prevented by using n masks. Following this direction, Nikova et al. extend the idea of masking with more than two shares in [27] to prevent those attacks which use sophisticated power models, e.g., counting the glitches occurring when the inputs of a complex combinational circuit change. They show that nonlinear functions implemented in such a way achieve provable security against first order DPA attacks and also resist higher order attacks that are based on a comparison of mean power consumption. Estimations of a hardware implementation of these ideas are presented in [28], but until now no real-world implementation of this scheme has been published. For a detailed description and theoretic foundations of their scheme we refer to their article in this special issue [29]. We chose their scheme, because it is a promising candidate for our goal of a lightweight and side-channel resistant implementation. This conclusion has also been drawn in [26], where an overview of the suitability of DPA countermeasures for lightweight implementations is provided.

3. Shared Computation of the PRESENT S-box Using Quadratic S-boxes

Our goal is to provide a lightweight and side-channel resistant implementation of a symmetric encryption algorithm with a reasonable security level that is suitable for passive RFID-tags. A glance at the hardware figures from several algorithms (see Table 1) leads us directly to PRESENT as the most suitable algorithm.

3.1. *Algorithmic Description of PRESENT*

PRESENT [3] is an aggressively hardware-optimized block cipher that was designed for ultra-constrained devices, such as passive RFID-tags. It has a block size of 64 bits and specifies two key lengths: 80 and 128 bits, referred to as PRESENT-80 and PRESENT-128, respectively. In the following we focus on PRESENT-80, since this is often stated as an adequate security level for pervasive applications. PRESENT consists of 31 full rounds and a final key whitening. Each round comprises an XOR with the roundkey, a substitution layer, and a permutation layer. The substitution layer consists of 16 applications of the same S-box with the following truth table:

Table 1. Hardware implementation results of selected symmetric encryption algorithms.

Algorithm		Key size	Block size	Cycles/ block	Tech. [μm]	Area [GE]
Stream ciphers						
Trivium	[11]	80	1	1	0.13	2,599
Grain	[11]	80	1	1	0.13	1,294
Block ciphers						
PRESENT	[34]	80	64	547	0.18	1,075
SEA	[23]	96	96	93	0.13	3,758
mCrypton	[22]	96	64	13	0.13	2,681
ICEBERG	[23]	128	64	16	0.13	7,732
HIGHT	[14]	128	64	34	0.25	3,048
AES	[8]	128	128	1,032	0.35	3,400
AES	[12]	128	128	160	0.13	3,100
DESXL	[21]	184	64	144	0.18	2,168

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 permutation layer of PRESENT is a bit permutation that moves bit i to bit position $P(i)$ according to the following equation:

$$P(i) = \begin{cases} i \cdot 16 \bmod 63, & i \in \{0, \dots, 62\}, \\ 63, & i = 63. \end{cases}$$

The key schedule generates a 64 bit roundkey out of the 80 bit key state for every round. First the user-supplied key is stored in a key register K and represented as $k_{79}k_{78}\cdots k_0$. At round i the 64 bit round key $K_i = \kappa_{63}\kappa_{62}\cdots\kappa_0$ consists of the 64 leftmost bits of the current contents of register K. Thus at round i we have that

$$K_i = \kappa_{63}\kappa_{62}\cdots\kappa_0 = k_{79}k_{78}\cdots k_{16}.$$

After extracting the round key K_i, the key register $K = k_{79}k_{78}\cdots k_0$ is updated as follows.

1. $[\mathtt{k}_{79}\mathtt{k}_{78}\cdots\mathtt{k}_1\mathtt{k}_0] = [\mathtt{k}_{18}\mathtt{k}_{17}\cdots\mathtt{k}_{20}\mathtt{k}_{19}]$
2. $[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}] = S[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}]$
3. $[\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] = [\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] \oplus \mathtt{round\_counter}$

Note that the key schedule uses the same S-box as the data path. Further details of PRESENT can be found in [3].

3.2. *Decomposition of PRESENT S-box into Composition of Two Quadratic S-boxes*

Since the PRESENT S-box $S(x)$ has algebraic degree 3, we want to decompose the PRESENT S-box into a composition of two quadratic S-boxes $F(X)$ and $G(X)$ (refer to Remark 1 later for the motivation). Each of these quadratic decompositions then has to be split into three shares to apply the secret sharing countermeasure to the PRESENT algorithm. Figure 1 depicts our approach graphically

$$S(X) = F\big(G(X)\big) \quad \text{where } S, F, G : GF(2)^4 \to GF(2)^4. \tag{1}$$

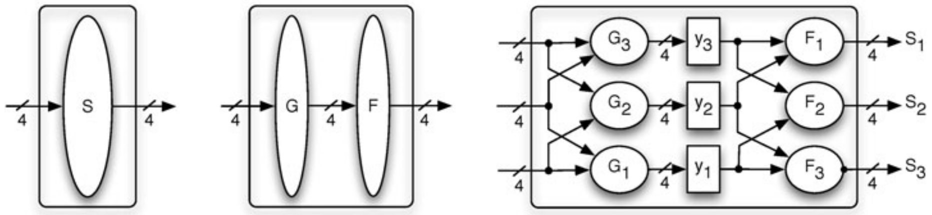

Fig. 1. The original PRESENT S-box S (*left*) is first decomposed into two S-boxes F and G (*center*) which then are split into three shares each (*right*).

Write the input and output of $G(X)$ as 4 bit vectors $X = (x, y, z, w)$ and $G(X) = (g_3(X), g_2(X), g_1(X), g_0(X))$. Each $g_i : GF(2)^4 \to GF(2)$ is a quadratic Boolean function whose algebraic normal form (ANF) is

$$g_i(x, y, z, w) = a_0 + a_1x + a_2y + a_3z + a_4w + a_{12}xy + a_{13}xz + a_{14}xw + a_{23}yz + a_{24}yw + a_{34}zw.$$

We do the same likewise for $F(X)$. From the ANF, there are 11 coefficients, each taking two possible values {0, 1}. Therefore, by looking at the eight output bits of F, G, there are at most $(2^{11})^8 = 2^{88}$ possibilities for the decomposition. A straightforward approach to search through this space for possible decompositions will take too long. We shall apply a few shortcuts to narrow down the search space.

Observation 1. *Since $S(X) = F(G(X))$ is a bijection, then $F(X)$ and $G(X)$ must also be bijections.*

Thus we can write the decomposition $S(X) = F(G(X))$ as

$$S(G^{-1}(X)) = F(X).$$

Now we just need to search through all possible quadratic functions $G(X)$ and compute $F(X) = S(G^{-1}(X))$ to see if it is also a quadratic function. In this way, we cut down the search space from 2^{88} to $(2^{11})^4 = 2^{44}$. This is doable on a workstation but we would like to cut down the complexity some more.

Note that we can rewrite $S(X) = F(G(X))$ as $S(X) = F'(G'(X))$ where $G'(X) = G(X) + G(0)$ and $F'(X) = F(X + G(0))$. Thus we can assume that $G(0) = 0$ and get the other decompositions directly by substituting 15 nonzero values for $G(0)$. Therefore, we only need to vary the 10 nonconstant coefficients in the ANF and the search space is reduced to $(2^{10})^4 = 2^{40}$.

We can also make use of the following well-known result on the balancedness of a vectorial Boolean function and each component output function. We say a vectorial function $G : GF(2)^n \to GF(2)^m$ is *balanced* if each $Y \in GF(2)^m$ has exactly 2^{n-m} preimages.

Proposition 1 [5, Proposition 2]. *Let $G : GF(2)^n \to GF(2)^m$; then $G(X)$ is balanced if and only if every linear combination of output bits is a balanced Boolean function, i.e., $\sum_{i \in I} g_i(X) : GF(2)^n \to GF(2)$ is balanced for every index set I where $G(X) = (g_{m-1}(X), \dots, g_0(X))$.*

To go through all bijective and quadratic $G(X)$ (and test if $S(G^{-1}(X))$ is quadratic), we use a 4-layer nested loop to vary the 4 components $g_i(X)$ of $G(X)$, while ensuring that $\sum_{i \in I} g_i(X)$ is balanced at each step. This allows us to further reduce the search space from 2^{40} to 2^{26}, which can be completed in a few minutes on a PC.

We find 141,120 decompositions $S(X) = F(G(X))$ with $G(0) = 0$. By varying $G(0)$ for 15 nonzero 4 bit vectors in $S(X) = F'(G'(X))$ as explained above, we get all $15 \times 141{,}120 = 2{,}116{,}800$ possible decompositions of $S(X) = F(G(X))$. We list in Table 2 one such example.

Table 2. Look-up tables of the quadratic S-boxes $F(X)$ and $G(X)$ for a decomposition of the PRESENT S-box $S(X) = F(G(X))$.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$G[x]$	7	E	9	2	B	0	4	D	5	C	A	1	8	3	6	F
$F[x]$	0	8	B	7	A	3	1	C	4	6	F	9	E	D	5	2

3.3. *Efficient Shared Computation of the Component Functions*

For secure implementation of the PRESENT S-box $S(X) = F(G(X))$ against side-channel attack, we need to split both the computation of $F(X)$ and $G(X)$ into shares. For efficiency, we split F, G into 3 input and output shares, which is the minimum number of shares required to satisfy the following properties for protection against first order side-channel attacks [28]. We shall explain the properties for F, noting that they must hold for G as well.

1. *Correctness and Noncompleteness* [28, Sects. 3.2, 3.3]. We decompose F as follows:

$$F(X_1 + X_2 + X_3) = F_1(X_2, X_3) + F_2(X_1, X_3) + F_3(X_1, X_2).$$

where $X_i \in GF(2)^4$ are the 3 input shares and $F_i : GF(2)^8 \to GF(2)^4$ are the 3 output shares of F. Correctness means the components F_1, F_2, F_3 sum up to the function F. Noncompleteness means each F_i is independent of the variable X_i. for G.

2. *Uniform* [28, Sect. 3.4]. For each unshared input, each shared output value must be equally likely: if we fix $X \in GF(2)^4$, then as we vary through all $(X_1, X_2, X_3) \in GF(2)^{12}$ with $X = X_1 + X_2 + X_3$, the output $(F_1(X_2, X_3), F_2(X_1, X_3), F_3(X_1, X_2)) \in GF(2)^{12}$ is uniformly distributed. Likewise for $G(X)$. In short,

$$(X_1, X_2, X_3) \mapsto \big(F_1(X_2, X_3), F_2(X_1, X_3), F_3(X_1, X_2)\big)$$

is a 12 bit permutation.

Write

$$F(x, y, z, w) = A_0 + A_1x + A_2y + A_3z + A_4w + A_{12}xy + A_{13}xz + A_{14}xw + A_{23}yz + A_{24}yw + A_{34}zw,$$

where $(x, y, z, w) \in GF(2)^4$ and each coefficient $A_i \in GF(2)^4$ is a 4 bit vector. We also denote the input share X_i by the 4 bit vector (x_i, y_i, z_i, w_i) for $i = 1, 2, 3$ and expand:

$$\begin{aligned}
&F(X_1 + X_2 + X_3)\\
&\quad = F(x_1 + x_2 + x_3, y_1 + y_2 + y_3, z_1 + z_2 + z_3, w_1 + w_2 + w_3)\\
&\quad = A_0 + A_1(x_1 + x_2 + x_3) + A_2(y_1 + y_2 + y_3) + A_3(z_1 + z_2 + z_3) + \cdots\\
&\qquad + A_{24}(y_1 + y_2 + y_3)(w_1 + w_2 + w_3) + A_{34}(z_1 + z_2 + z_3)(w_1 + w_2 + w_3)\\
&\quad = F_1(X_2, X_3) + F_2(X_1, X_3) + F_3(X_1, X_2),
\end{aligned}$$

and place the monomials in the expansion in different output shares F_i. For monomials involving two indices, it is obvious which F_i to place them in. For example, we must place monomials $y_1 w_2$ and $z_2 w_1$ in $F_3(X_1, X_2)$. For monomials involving just one index, e.g., x_1 or $y_2 w_2$, we adopt the convention that terms with index 1 (resp. 2, 3) are placed in F_3 (resp. F_1, F_2). The constant term is placed in F_1. For instance, F_1 is defined by

$$\begin{aligned}F_1(X_2, X_3) &= F_1(x_2, y_2, z_2, w_2, x_3, y_3, z_3, w_3)\\ &= A_0 + A_1 x_2 + A_2 y_2 + A_3 z_2 + A_4 w_2 + A_{12}(x_2 y_2 + x_2 y_3 + x_3 y_2)\\ &\quad + A_{13}(x_2 z_2 + x_2 z_3 + x_3 z_2) + A_{14}(x_2 w_2 + x_2 w_3 + x_3 w_2)\\ &\quad + A_{23}(y_2 z_2 + y_2 z_3 + y_3 z_2) + A_{24}(y_2 w_2 + y_2 w_3 + y_3 w_2)\\ &\quad + A_{34}(z_2 w_2 + z_2 w_3 + z_3 w_2).\end{aligned}$$

Remark 1. In the above construction, we see that the expanded quadratic terms can easily be placed into 3 noncomplete shares. This explains why we had to split $S(X)$ into quadratic $F(X)$, $G(X)$. In comparison, if we had wanted to split the cubic function $S(X)$ into 3 shares in the same way, it would not be possible because of the existence of cubic terms.

It turns out that exactly $3/7$ of the 2,116,800 decompositions automatically satisfy the uniformity condition for both F_1, F_2, F_3 and G_1, G_2, G_3 without the need for correction terms: thus we have $3/7 \times 2{,}116{,}800 = 907{,}200$ decompositions of the PRESENT S-box into quadratic shares which satisfy the correctness, noncompleteness, and uniformity conditions of [28].

Among these, we choose the decomposition that gives the most space-efficient hardware implementation. Note that an XOR takes twice as much resources to implement as an AND gate. So we shall give a weightage of 1 to each AND and 2 to each XOR gate in the shares F_i, G_i. Equivalently, we would like to find the pairs (F, G) where

$$\begin{aligned}&\text{Weighted Sum}\\ &\quad = 2 \times (\text{Sum of Hamming weight of constant terms of } F, G)\\ &\qquad + 6 \times (\text{Sum of Hamming weight of linear coefficients of } F, G)\\ &\qquad + 27 \times (\text{Sum of Hamming weight of quadratic coefficients of } F, G),\end{aligned}$$

is minimum. This is because each constant term uses an XOR, which gives a weightage of 2. Each linear term of F is expanded into 3 linear terms in F_i where 3 XORs give a weightage of $3 \times 2 = 6$. Each quadratic term is expanded into 9 quadratic terms in F_i where 9 XORs and 9 ANDs give a weightage of $9 \times 2 + 9 = 27$ (likewise for G).

We found 24 optimal decompositions (F, G) with minimum weighted sum 339. They all satisfy:

- Sum of Hamming weight of constant terms in $(F, G) = 3$
- Sum of Hamming weight of linear coefficients in $(F, G) = 15$

– Sum of Hamming weight of quadratic coefficients in $(F, G) = 9$

These functions will give a distributed implementation of the PRESENT S-box that is secure against side-channel analysis and uses a minimal number of gates. One such example that we used for implementation is given by the functions $G(X), F(X)$ (in look-up table format) in Table 2. Their algebraic normal forms (ANFs) are given by:

$$\begin{aligned}
&G(x, y, z, w) = (g_3, g_2, g_1, g_0),\\
&\quad g_3 = y + z + w,\ g_2 = 1 + y + z,\ g_1 = 1 + x + z + yw + zw,\\
&\quad g_0 = 1 + w + xy + xz + yz.\\
&F(x, y, z, w) = (f_3, f_2, f_1, f_0),\\
&\quad f_3 = y + z + w + xw,\ f_2 = x + zw,\ f_1 = y + z + xw,\\
&\quad f_0 = z + yw.
\end{aligned}$$

Their output shares (F_1, F_2, F_3) and (G_1, G_2, G_3) can be calculated by the formulas in this section. The ANFs of the six output shares are listed in the Appendix A. In the following sections we will use these output shares to implement five different lightweight hardware architectures of PRESENT and attack them by DPA.

4. Hardware Architectures

This section is dedicated to the description of the different hardware profiles that we will attack in the next section. For this purpose we first introduce the design flow used before we detail the hardware architectures and finally summarize the implementation results.

4.1. *Design Flow*

In the last section we decomposed the PRESENT S-box $S(x)$ into two S-boxes $F(x)$ and $G(x)$ with algebraic degree 2 and split them into three shares (F_1, F_2, F_3) and (G_1, G_2, G_3) (see the Appendix A for their ANFs). For the hardware implementation in *VHDL*, we used the Boolean minimization tool `BOOM II` [9,10] to obtain their 24 Boolean functions. For functional simulation we used *Mentor Graphics ModelSimXE 6.4b*, and *Synopsys DesignCompiler* version *A-2007.12-SP1* [38] was used to synthesize the designs to the *Virtual Silicon* (VST) standard cell library *UMCL18G212T3*, which is based on the *UMC L180 0.18 μm 1P6M* logic process and has a typical voltage of 1.8 V [43]. We used *Synopsys Power Compiler* version *A-2007.12-SP1* [39] to estimate the power consumption of our ASIC implementations. For synthesis and for power estimation we advised the compiler to keep the hierarchy and use a clock frequency of 100 kHz, which is a widely used operating frequency for RFID applications. Note that the wire load model used, though it is the smallest available for this library, still simulates the typical wire load of a circuit with a size of around 10,000 GE.

To substantiate our claims on the efficacy of the proposed countermeasures, we decided to implement the ASIC cores on a SASEBO to obtain and evaluate real-world power traces. The crypto core FPGA on SASEBO consists of an *XC2VP7 Virtex-II Pro* (Package FG456 with speed grade-5) from *Xilinx* [46]. For design synthesis, implementation, and configuration of SASEBO we used *Xilinx ISE v10.1.03 WebPACK*. For the power measurements on SASEBO we had to modify the finite state machine of the cryptographic core in order to implement a handshaking protocol, because the control FPGA and the crypto core FPGA have to be synchronized. This modification has no effect on the power measurements, but results in a more complex finite state machine and introduces some timing overhead. In a typical application scenario the cryptographic core would be part of an integrated ASIC, hence no such I/O communication is required. Consequently, in this section we discuss the implementation cost of a cryptographic core without these I/O overheads.

4.2. *Different Countermeasure Profiles for Different Security Levels*

To fully exploit the security-cost trade-off inherent in strengthening hardware against side-channel attacks, we propose five different profiles with different security levels. These profiles combine none, one, or many of the following countermeasure options:

Option 1: Sharing the data path
Option 2: Sharing the key schedule
Option 3: Randomly permuting the shares

As depicted in the left part of Table 3, profile 1 does not apply any of the countermeasures. Profile 2 shares the data path and profile 3 additionally permutes the shares. Profile 4 shares the data path and the key schedule and profile 5 applies all countermeasures.

The overall architecture of all variants is depicted in Fig. 2, where we left out further details such as the finite state machine and control and clock signals for the sake of clarification. As one can see, the core has two 4 bit wide inputs (`data_in`, `key`) and one 4 bit wide output (`data_out`) that is gated with an AND gate to prevent leakage. The unprotected implementation (*profile 1*, solid lines) consists of the `State` and the `Key` module, an XOR, a MUX, and a standard (i.e. not decomposed, not shared) PRESENT S-box. The `State` module comprises 16 4-bit wide clock-gated registers with two modes of operation: in serial mode it forwards 4 bits to the next stage, thus acting like a 4 bit wide shift register, and in parallel mode it performs the permutation layer of PRESENT within 1 clock cycle. The `Key` module comprises 20 4-bit wide

Table 3. Post-synthesis implementation results of different architectures of a serialized PRESENT-80. The power consumption was estimated at 100 KHz and a supply voltage of 1.8 V.

Profile	Sharing		Rand.	Cycles		Current		Area	
	Data [Y/N]	Key [Y/N]	Perm. [Y/N]	Total [clk]	Rel. [%]	Total [μA]	Rel. [%]	Total [GE]	Rel. [%]
1	N	N	N	547	100	1.34	100	1,111	100
2	Y	N	N	547	100	2.86	213	2,282	205
3	Y	N	Y	547	100	3.10	231	2,417	218
4	Y	Y	N	578	106	4.23	316	3,322	299
5	Y	Y	Y	578	106	5.02	375	3,582	322

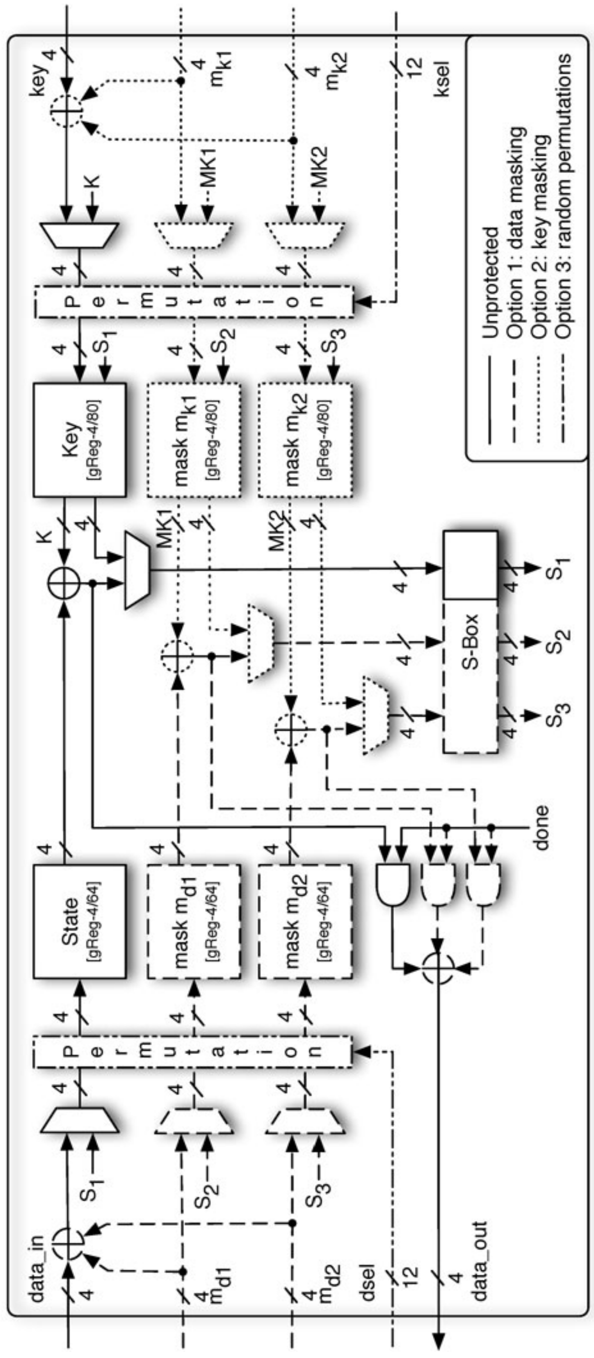

Fig. 2. Lightweight hardware architectures of five different profiles of a serialized PRESENT-80.

clock-gated registers and the serial mode is similar to the one previously described. In parallel mode the `Key` module performs the key schedule of PRESENT (61 bit left rotation, substitution of the 4 MSB by the S-box and adding the counter) in one cycle. One round of PRESENT requires 16 clock cycles to substitute the data state and 1 clock cycle for the key schedule and the permutation layer. Including the initialization phase, a total of $31 \times (16 + 1) + 20 = 547$ clock cycles are required to process one block of data.

If we make use of the first option (sharing the data path) we obtain *profile 2*. The additional hardware requirements for this profile are depicted in Fig. 2 by the dashed lines. For this profile we need two randomly generated masks (m_{d1} and m_{d2}), which are XORed to the data chunk during initialization. The unmasking step is performed by simply XORing all three shares yielding the output (`data_out`). The state of the masks also needs to be maintained, which leads to two more instantiations of the `State` module labeled `mask` m_{d1} and `mask` m_{d2}. Furthermore, the S-box is now replaced by a decomposed and shared S-Box module that contains a pipelining stage (see Fig. 1 for details). Note that in profile 2 the key schedule is not shared, thus it cannot use the shared S-box. Hence, additionally the standard PRESENT S-box has to be implemented, but the MUX can be omitted.

Profile 4 shares both the data path and the key schedule, hence it combines option 1 and option 2. In Fig. 2 the additional overhead due to option 2 is denoted by dotted lines. Similarly to profile 2, during initialization the key is XORed with two randomly generated masks (m_{k1} and m_{k2}). Contrary to the data masks, the key masks have to be removed (by simply XORing them to the data masks) in every round, which leads to two additional XOR gates. The state of the key masks is stored in two slightly modified instantiations of the `Key` module labeled `mask` m_{k1} and `mask` m_{k2}. It suffices to add the counter once, so in parallel mode `mask` m_{k1} and `mask` m_{k2} implement one round of the key schedule without the counter addition. In profile 4 also the key schedule uses the decomposed and shared S-box, which results in the following area and timing overhead: three MUXes are required to select the correct input (one for each share); due to the pipelining stage within the shared S-box, one additional cycle per round (31 in total) is required to wait for the result of the substitution step of the key schedule.

In order to further strengthen the implementation, we propose to randomly permute the shares in each round (option 3, dashed and dotted lines in Fig. 2). This countermeasure adds noise to the measurements, thus making an attack even harder, at the cost of additional mask bits per clock cycle. There are six possibilities to permute three inputs, so three bits are required to select the permutation. In order to make the probability of a permutation more uniform, we used the encoding of Table A.1 in the Appendix A. If we permute each of the 4 bits of a data chunk we need 12 randomly generated bits for one instantiation of the random permutation module. *Profile 5* combines data path and key schedule sharing and permutes all shares in every clock cycle. Therefore 24 random bits are required per clock cycle. *Profile 3* combines data path sharing and random permutation, so only 12 random bits are required per clock cycle.

4.3. *Performance Figures*

Table 3 summarizes the implementation figures of all five profiles. Columns 2 to 4 display if data masking, key masking, or random permutation, respectively, have been

Table 4. Breakdown of the post-synthesis implementation results of different architectures of a serialized PRESENT-80. *P* stands for Profile.

P	m_{d1} m_{d2}		m_{k1} m_{k2}		Rand. Perm.		S-box		FSM		State		Key		Other		Sum	Rel.
	%	GE	%	GE	%	GE	%	GE	%	GE	%	GE	%	GE	%	GE	GE	%
1	0	0	0	0	0	0	3	32	13	145	35	389	45	498	4	47	1,111	100
2	34	778	0	0	0	0	17	387	6	146	17	389	22	498	3	75	2,282	205
3	32	778	0	0	5	121	16	387	6	146	16	389	21	498	4	98	2,417	218
4	23	778	29	970	0	0	11	355	5	156	12	389	15	498	5	176	3,322	299
5	22	778	27	970	7	243	10	355	4	155	11	389	14	498	5	194	3,582	322

applied. Then the required clock cycles for processing one block, the average current consumption in μA, and the area footprint in GE are shown. For these measures we provide absolute figures and—in order to better highlight the overhead of each combination of countermeasures—also relative figures. For a detailed breakdown of the area requirements we refer to Table 4.

Profile 1 is an unprotected serialized PRESENT-80 implementation without any side-channel attack countermeasures. It has an area footprint of 1,111 GE of which 80% are required to store the key and the data state. Masking the data path with the secret sharing countermeasure (profile 2) adds the shared S-box component and two XOR gates. In addition to this, the mask states have to be stored, which mainly contributes to the area and power overhead. Adding the random permutation countermeasure leads to a moderate increment (121 GE) of the area requirements.

Each of profiles 1, 2, and 3 requires 547 clock cycles to process one block of 64 bits. For initialization 20 clock cycles are required to load the plaintext and key (and both data masks) into the ASIC. Then each of the 31 rounds requires 16 clock cycles to process all data chunks by the S-box, and 1 clock cycle for the permutation layer and the key schedule. It can be clearly seen that neither the sharing of the data path nor the random permutation of its shares leads to a timing overhead. However, if we also share the key schedule (as in profiles 4 and 5), one additional cycle per round is required. Therefore, these architectures require 578 cycles to process one block of 64 bits, which is a very moderate increment of 6% compared to profiles that do not share the key schedule. Note that it would also be possible to save this additional clock cycle per round at the cost of additional hardware resources, e.g., by implementing a second shared S-box. Since the sum of 578 clock cycles is still moderate for the application scenarios envisioned, we decided to save area at the cost of 31 additional clock cycles.

The power consumption was estimated at 100 KHz and a supply voltage of 1.8 V. The unprotected implementation (profile 1) requires 1.34 μA, and these figures increase up to 5.02 μA for profile 5. All estimated power figures are in the range of μA, thus we conclude that all implementations are suitable for passive RFID-tags.

5. Experimental Results

In order to practically investigate the resistance of the proposed schemes, they have been implemented on an FPGA-based platform, and actual power consumption traces have been analyzed. In the following subsections first the conditions and specification of the platform used and the measurement setup are introduced, and then practical results of different profiles are compared to validate the desired security levels.

5.1. *Measurement Setup*

Profiles are implemented on a circuit board SASEBO (side-channel attack standard evaluation board) which is particularly designed for side-channel attack experiments [36]. The profiles are implemented on an xc2vp7 Virtex-II Pro FPGA [46], i.e., the crypto FPGA of the target SASEBO, and the clock signal is provided by a 1.8432MHz oscillator.[2] Power traces are collected using a LeCroy WP715Zi 1.5 GHz oscilloscope at a sampling rate of 10 GS/s and by means of a differential probe which captures the voltage drop of a 1Ω resistor at VDD (3.3V) path.

5.2. *Side-Channel Resistance*

We first focus on the unprotected version, i.e., profile 1. According to the architecture presented in Fig. 2, each 4 bit nibble is processed separately at each clock cycle; a nibble of the state register is XORed by a roundkey part and processed by the S-box block; then while the state register and the key register are shifted by 4 bits, the S-box result (a nibble) is replaced by a 4 bit nibble of the state register. Figure 3 shows a measured trace of profile 1 and indicates which operation takes place at each clock cycle.

In order to find the leakage resources, several DPA attacks have been performed using different power models, e.g., Hamming weight (HW) of the S-box input, HW of the S-box output, and (partial) Hamming distance (HD) of the state register. According to Fig. 4, which shows the attack results on the 8th nibble of the roundkey at the first round

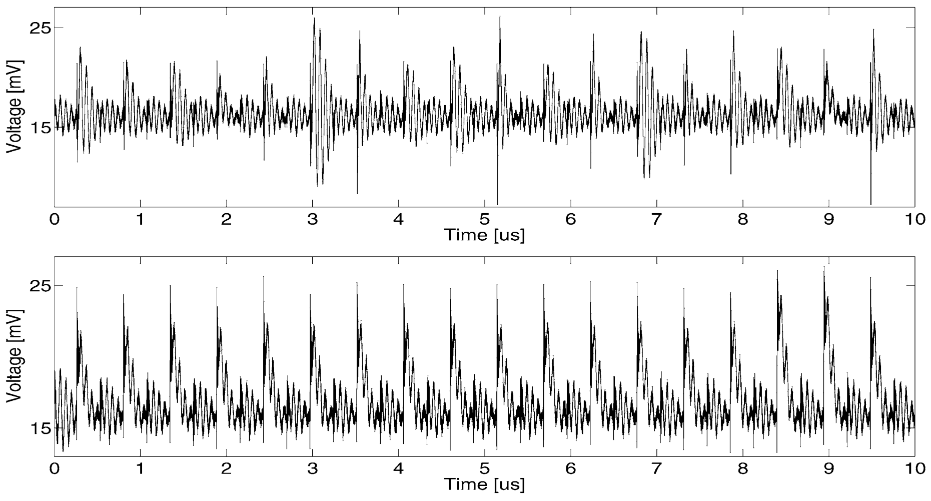

Fig. 3. Measured power traces of profile 1 (*top*), profile 2 (*bottom*).

[2] This frequency of operation is selected to prevent overlapping power peaks of consecutive clock cycles and hence to simplify the attacks.

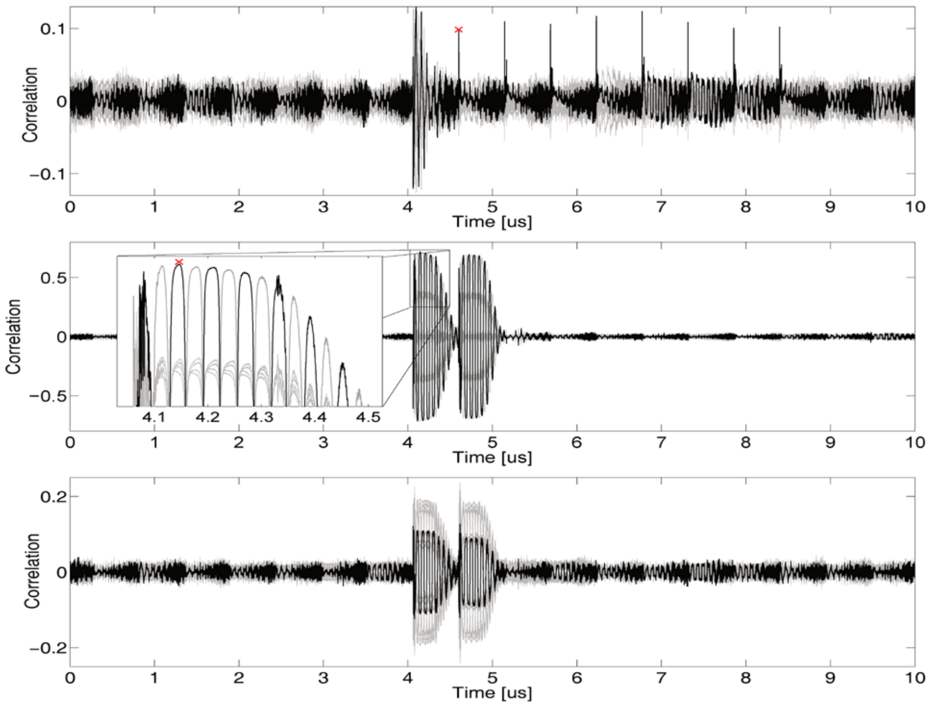

Fig. 4. DPA results on profile 1 using 10,000 traces estimating HD of a nibble of the state register (*top*), HW of the S-box input (*middle*), and HW of the S-box output (*bottom*).

of the cipher using 10,000 traces, the leakage of the S-box input is more obvious than those of other power models. Note that in all attacks the plaintexts are chosen randomly, and to perform the attack based on the HD of the state register, we gave a favor to the adversary by knowing the last nibble of the roundkey to estimate the HD.[3] Moreover, in order to have an estimation about the number of required traces, Fig. 5 shows the attack result over the number of traces for the best point of the first two attacks presented in Fig. 5.

In order to check the resistance of profile 2, the same attacks have been performed on 100,000 traces of the corresponding implementation. A measured trace is presented in Fig. 3. None of the power models leads to recovering (a part of) the secret key, e.g., Fig. 6 shows the attack results estimating HD of the state register and HW of the S-box input. Like previous attacks, both target revealing the 8th roundkey nibble.

As expected, since in profile 2 (supposing a fixed key) the key register holds the same values at the first round for all encryptions, and HD of the key register is fixed while rotating the key nibbles, one can profile the leakage of the key register and reduce the key entropy. To verify this we have measured and got an average of 100,000 traces of profile 2 separately for three different keys[4]: (i) all $(00)_h$ leading to HD of zero, (ii) all

[3] Indeed, this assumption is quite realistic since to attack the first nibble of the roundkey all values of the state register are known (as plaintext), and the other roundkey nibbles can be attacked one after another.

[4] As before, the plaintexts and masks have been selected randomly.

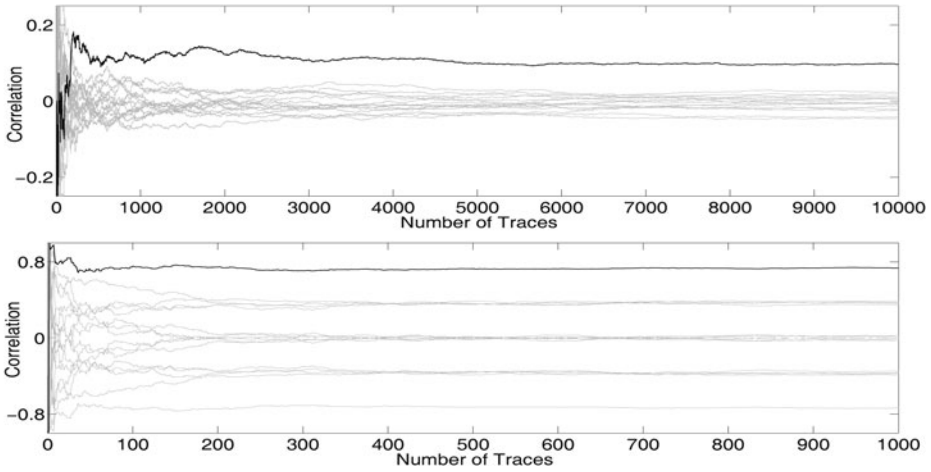

Fig. 5. DPA results over the number of traces for HD model on time instant of 4.6 μs (*top*) and HW model (S-box input) on time instant of 4.14 μs (*bottom*).

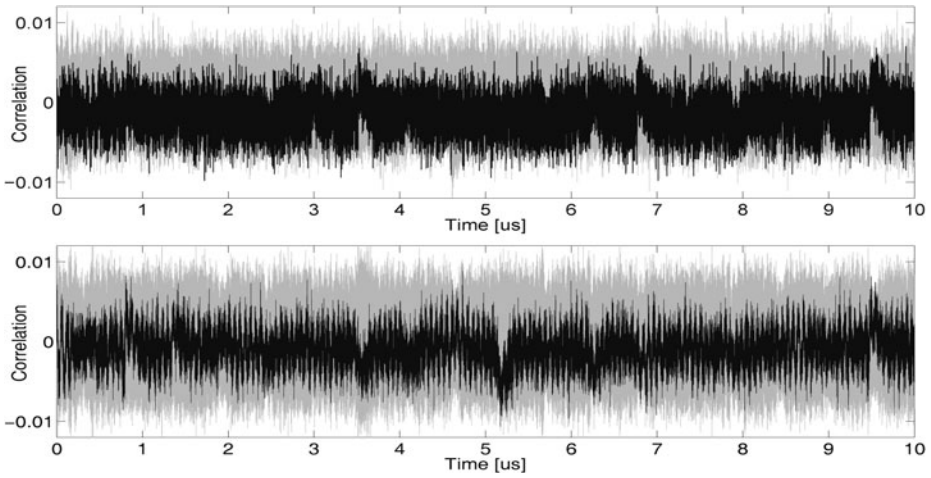

Fig. 6. DPA results on profile 2 using 100,000 traces estimating (*top*) HD of a nibble of the state register, (*bottom*) HW of the S-box input.

$(f0)_h$ which causes HD to be maximum, i.e., 80, and (iii) $(56789abcdef012345678)_h$ which makes HD to equal 40. Figure 7(a) presents a part of these mean traces. Obviously, they are sorted based on HD of the key register. Supposing that the adversary can detect HD of the key register, although the entropy loss would be very low, it can restrict the key space for the cases where the estimated HD is very low (say close to 0) or very high (close to 80). To overcome this vulnerability, profile 4 is proposed, where the key register is masked as well, and it is expected that profiling the target device by means of HD of the key register would not be possible. The same scenario is repeated on the same number of traces of profile 4. As Fig. 7(b) shows, the mean traces are very

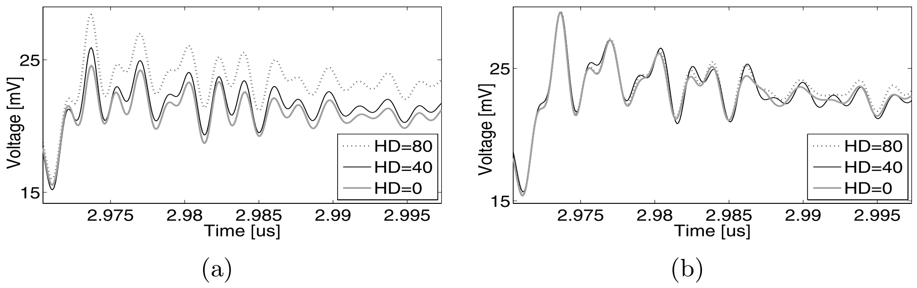

Fig. 7. Means of power traces of (**a**) profile 2, (**b**) profile 4.

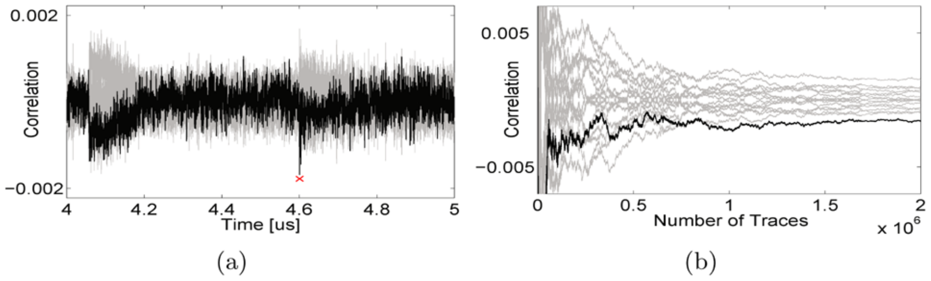

Fig. 8. DPA results on profile 4 estimating HW of the S-box input (**a**) using 2,000,000 traces, (**b**) over the number of traces on time instant of 4.6 µs.

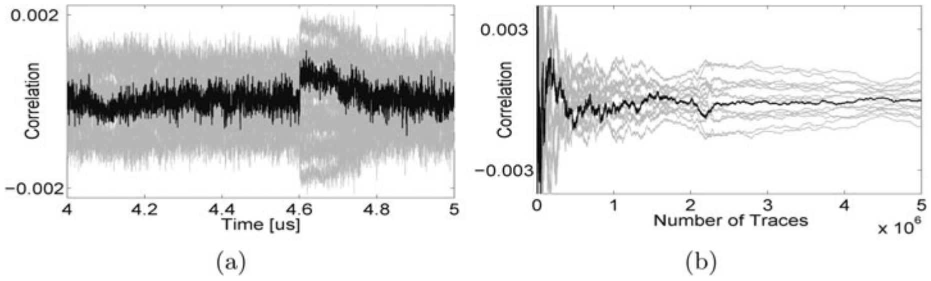

Fig. 9. DPA results on profile 5 estimating HW of the S-box input (**a**) using 5,000,000 traces, (**b**) over the number of traces on time instant of 4.6 µs.

close to one another, and profiling based on HD of the key register would not help in this regard.

In order to investigate the strength of profile 4 to resist DPA attacks, we have measured 2,000,000 traces and performed the former attacks. Figure 8(a) shows the attack result estimating HW of the S-box input when processing the 8th nibble at the first round. Obviously at some points, e.g., 4.6 µs, the correlation coefficient for the correct hypothesis is distinguishable among the others. As Fig. 8(b) represents the attack result on a time instant of 4.6 μs, around 1,000,000 measurements are required to detect the correct hypothesis.

To avoid such leakage, profile 5, which makes use of random permutation in addition to key and data sharing, is proposed. To evaluate the efficacy of this profile in comparison to other profiles, 5,000,000 traces are collected and the same attacks are performed. According to Fig. 9, which shows the attack results, the leakage of profile 4 is prevented, and the correct hypothesis is not distinguishable.

6. Conclusions

Attacker models for the upcoming age of ubiquitous computing have to extend classical attacker models and also take physical attacks into account, while implementations of cryptographic algorithms face fierce area and power constraints. Given a cost-driven deployment, Moore's law does not relax this situation, but further increases the demand for lightweight solutions. Unfortunately, nearly all side-channel countermeasures introduce power and area overhead which are proportional to the values of the unprotected implementation. Therefore, the *relative* difference of two protected cryptographic algorithms stays the same, but the *absolute* difference increases by the factor of the countermeasure overhead. This fact prohibits the implementation of a wide range of proposed countermeasures and also narrows down possible cipher candidates. In this article we have selected PRESENT as the encryption algorithm and a recently proposed secret-sharing-based masking scheme in combination with random permutation of the share inputs. In order to apply the masking scheme to PRESENT, we had to decompose the S-box $S(x)$ into two S-boxes $F(x)$, $G(x)$ of algebraic degree 2 and split them into three shares each (F_1, F_2, F_3 and G_1, G_2, G_3). We have defined four different profiles that combine a subset of these countermeasures (plus an unprotected version), thus yielding different levels of side-channel resistance. Their absolute area footprints—ranging from 2,282 GE to 3,582 GE—are surprisingly moderate, while the timing overhead is either 6% or none at all. According to practical side-channel investigations, masking the state register by means of two shares prevents the straightforward DPA attacks. However, since the key register is not protected, profiling the leakage of the key register may help on key space restriction. Masking the key register prevents such a leakage, but cannot protect against those attacks which make use of around 1,000,000 traces. However, the perfect resistance against first order attacks (at least using 5,000,000 measurements) is achieved by combining data masking, key masking, and random data and key permutations.

Acknowledgements

The research was supported in part by the Singapore National Research Foundation under Research Grant NRF-CRP2-2007-03. The authors would like to thank Akashi Satoh for his prompt and kind help in obtaining a SASEBO.

Appendix A

Listed below are the algebraic normal forms (ANFs) of the six output shares G_1, G_2, G_3, F_1, F_2, F_3 described in Sect. 3

$$G_1(x_2, y_2, z_2, w_2, x_3, y_3, z_3, w_3) = (g_{13}, g_{12}, g_{11}, g_{10}),$$
$$g_{13} = y_2 + z_2 + w_2, g_{12} = 1 + y_2 + z_2,$$
$$g_{11} = 1 + x_2 + z_2 + y_2w_2 + y_2w_3 + y_3w_2 + z_2w_2 + z_2w_3 + z_3w_2,$$
$$g_{10} = 1 + w_2 + x_2y_2 + x_2y_3 + x_3y_2 + x_2z_2 + x_2z_3 + x_3z_2 + y_2z_2 + y_2z_3 + y_3z_2;$$
$$G_2(x_1, y_1, z_1, w_1, x_3, y_3, z_3, w_3) = (g_{23}, g_{22}, g_{21}, g_{20}),$$
$$g_{23} = y_3 + z_3 + w_3, g_{22} = y_3 + z_3,$$
$$g_{21} = x_3 + z_3 + y_3w_3 + y_1w_3 + y_3w_1 + z_3w_3 + z_1w_3 + z_3w_1,$$
$$g_{20} = w_3 + x_3y_3 + x_1y_3 + x_3y_1 + x_3z_3 + x_1z_3 + x_3z_1 + y_3z_3 + y_1z_3 + y_3z_1;$$
$$G_3(x_1, y_1, z_1, w_1, x_2, y_2, z_2, w_2) = (g_{33}, g_{32}, g_{31}, g_{30}),$$
$$g_{33} = y_1 + z_1 + w_1, g_{32} = y_1 + z_1,$$
$$g_{31} = x_1 + z_1 + y_1w_1 + y_1w_2 + y_2w_1 + z_1w_1 + z_1w_2 + z_2w_1,$$
$$g_{30} = w_1 + x_1y_1 + x_1y_2 + x_2y_1 + x_1z_1 + x_1z_2 + x_2z_1 + y_1z_1 + y_1z_2 + y_2z_1;$$
$$F_1(x_2, y_2, z_2, w_2, x_3, y_3, z_3, w_3) = (f_{13}, f_{12}, f_{11}, f_{10}),$$
$$f_{13} = y_2 + z_2 + w_2 + x_2w_2 + x_2w_3 + x_3w_2,\ f_{12} = x_2 + z_2w_2 + z_2w_3 + z_3w_2,$$
$$f_{11} = y_2 + z_2 + x_2w_2 + x_2w_3 + x_3w_2,\ f_{10} = z_2 + y_2w_2 + y_2w_3 + y_3w_2;$$
$$F_2(x_1, y_1, z_1, w_1, x_3, y_3, z_3, w_3) = (f_{23}, f_{22}, f_{21}, f_{20}),$$
$$f_{23} = y_3 + z_3 + w_3 + x_3w_3 + x_1w_3 + x_3w_1,\ f_{22} = x_3 + z_3w_3 + z_1w_3 + z_3w_1,$$
$$f_{21} = y_3 + z_3 + x_3w_3 + x_1w_3 + x_3w_1,\ f_{20} = z_3 + y_3w_3 + y_1w_3 + y_3w_1;$$
$$F_3(x_1, y_1, z_1, w_1, x_2, y_2, z_2, w_2) = (f_{33}, f_{32}, f_{31}, f_{30}),$$
$$f_{33} = y_1 + z_1 + w_1 + x_1w_1 + x_1w_2 + x_2w_1,\ f_{32} = x_1 + z_1w_1 + z_1w_2 + z_2w_1,$$
$$f_{31} = y_1 + z_1 + x_1w_1 + x_1w_2 + x_2w_1,\ f_{30} = z_1 + y_1w_1 + y_1w_2 + y_2w_1.$$

Table A.1. Encoding of the random permutation. The input (A, B, C) is permuted to the output in the second column according to the value of SEL.

SEL	Output
000	(A, B, C)
001	(A, C, B)
010	(B, A, C)
011	(B, C, A)
100	(C, A, B)
101	(C, B, A)
110	(B, C, A)
111	(C, A, B)

References

[1] E. Biham, A. Shamir, Differential fault analysis of secret key cryptosystems, in *Advances in Cryptology—CRYPTO 1997*, ed. by B.S. Kaliski. Lecture Notes in Computer Science, vol. 1294 (Springer, Berlin, 1997), pp. 513–525

[2] G.R. Blakley, Safeguarding cryptographic keys, in *National Computer Conference* (1979), pp. 313–317

[3] A. Bogdanov, G. Leander, L. Knudsen, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, C. Vikkelsoe, PRESENT—an ultra-lightweight block cipher, in *Cryptographic Hardware and Embedded Systems—CHES 2007*, ed. by P. Paillier, I. Verbauwhede. Lecture Notes in Computer Science, vol. 4727 (Springer, Berlin, 2007), pp. 450–466

[4] E. Brier, C. Clavier, F. Olivier, Correlation power analysis with a leakage model, in *CHES 2004*. Lecture Notes in Computer Science, vol. 3156 (Springer, Berlin, 2004), pp. 16–29

[5] C. Carlet, Vectorial (multi-output) boolean functions for cryptography, in *Boolean Methods and Models* (Cambridge University Press, Cambridge, to appear)

[6] S. Chari, C.S. Jutla, J.R. Rao, P. Rohatgi, Towards sound approaches to counteract power-analysis attacks, in *Advances in Cryptology—CRYPTO 1999*, ed. by M. Wiener. Lecture Notes in Computer Science, vol. 1666 (Springer, Berlin, 1999), pp. 398–412

[7] T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, M.T.M. Shalmani, 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*. Lecture Notes in Computer Science, vol. 5157 (Springer, Berlin, 2008), pp. 203–220

[8] M. Feldhofer, J. Wolkerstorfer, V. Rijmen, AES implementation on a grain of sand. *Inf. Secur. IEE Proc.* **152**(1), 13–20 (2005)

[9] P. Fišer, J. Hlavička, BOOM—a heuristic boolean minimizer. *Comput. Inf.* **22**(1), 19–51 (2003)

[10] P. Fišer, J. Hlavička, Two-level boolean minimizer BOOM-II, in *Proceedings of 6th Int. Workshop on Boolean Problems—IWSBP'04* (2004), pp. 221–228

[11] T. Good, M. Benaissa, Hardware results for selected stream cipher candidates. State of the Art of Stream Ciphers 2007 (SASC 2007), Workshop Record, February 2007. Available via www.ecrypt.eu.org/stream

[12] P. Hämäläinen, T. Alho, M. Hännikäinen, T.D. Hämäläinen, Design and implementation of low-area and low-power AES encryption hardware core, in *DSD* (2006), pp. 577–583

[13] C. Herbst, E. Oswald, S. Mangard, An AES smart card implementation resistant to power analysis attacks, in *Applied Cryptography and Network Security—ACNS 2006*. Lecture Notes in Computer Science, vol. 3989 (Springer, Berlin, 2006), pp. 239–252

[14] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, S. Chee, HIGHT: a new block cipher suitable for low-resource device, in *Cryptographic Hardware and Embedded Systems—CHES 2006*, ed. by L. Goubin, M. Matsui. Lecture Notes in Computer Science, vol. 4249 (Springer, Berlin, 2006), pp. 46–59

[15] A. Juels, S.A. Weis, Authenticating pervasive devices with human protocols, in *Advances in Cryptology—CRYPTO 2005*, ed. by V. Shoup. Lecture Notes in Computer Science, vol. 3126 (Springer, Berlin, 2005), pp. 293–198

[16] L.F.A. Karpinskyy, M. Korkishko, Masked encryption algorithm mCrypton for resource-constrained devices, in *IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2007—IDAACS 2007* (2007), pp. 628–633

[17] M. Khatir, A. Moradi, A. Ejlali, M.T. Manzuri Shalmani, M. Salmasizadeh, A secure and low-energy logic style using charge recovery approach, in *International Symposium on Low Power Electronics and Design—ISLPED 2008* (ACM, New York, 2008), pp. 259–264

[18] N.N. Keeloq algorithm. Available via http://en.wikipedia.org/wiki/KeeLoq, November 2006

[19] P.C. Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, in *Advances in Cryptology—CRYPTO 1996*, ed. by N.I. Koblitz. Lecture Notes in Computer Science, vol. 1109 (Springer, Berlin, 1996), pp. 104–113

[20] P.C. Kocher, J. Jaffe, B. Jun, Differential power analysis, in *Advances in Cryptology—CRYPTO 1999*, ed. by M. Wiener. Lecture Notes in Computer Science, vol. 1666 (Springer, Berlin, 1999), pp. 388–397

[21] G. Leander, C. Paar, A. Poschmann, K. Schramm, New lightweight DES variants, in *Fast Software Encryption 2007—FSE 2007*. Lecture Notes in Computer Science, vol. 4593 (Springer, Berlin, 2007), pp. 196–210

[22] C. Lim, T. Korkishko, mCrypton—a lightweight block cipher for security of low-cost RFID tags and sensors, in *Workshop on Information Security Applications—WISA 2005*, ed. by J. Song, T. Kwon, M. Yung. Lecture Notes in Computer Science, vol. 3786 (Springer, Berlin, 2005), pp. 243–258

[23] F. Mace, F.-X. Standaert, J.-J. Quisquater, ASIC implementations of the block cipher sea for constrained applications, in *RFID Security—RFIDsec 2007, Workshop Record* (Malaga, Spain, 2007), pp. 103–114

[24] S. Mangard, E. Oswald, T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards* (Springer, Berlin, 2007)

[25] S. Mangard, N. Pramstaller, E. Oswald, Successfully attacking masked AES hardware implementations, in *Cryptographic Hardware and Embedded Systems—CHES 2005*. Lecture Notes in Computer Science, vol. 3659 (Springer, Berlin, 2005), pp. 157–171

[26] A. Moradi, A. Poschmann, Lightweight cryptography and DPA countermeasures: a survey, in *Workshop of Lightweight Cryptography—WLC'2010, Proceedings of Financial Cryptography*. Lecture Notes in Computer Science, vol. 6054 (Springer, Berlin, 2010), pp. 68–79

[27] S. Nikova, C. Rechberger, V. Rijmen, Threshold implementations against side-channel attacks and glitches, in *International Conference in Information and Communications Security—ICICS 2006*, ed. by P. Ning, S. Qing, N. Li. Lecture Notes in Computer Science, vol. 4307 (Springer, Berlin, 2006), pp. 529–545

[28] S. Nikova, V. Rijmen, M. Schläffer, Secure hardware implementations of non-linear functions in the presence of glitches, in *International Conference in Information Security and Cryptology—ICISC 2008*, ed. by P. Lee, J. Cheon. Lecture Notes in Computer Science, vol. 5461 (Springer, Berlin, 2008), pp. 218–234

[29] S. Nikova, V. Rijmen, M. Schläffer, Secure hardware implementations of non-linear functions in the presence of glitches. *J. Cryptol.* (2010). doi:10.1007/s00145-010-9085-7. Special issue on hardware and security

[30] E. Oswald, S. Mangard, N. Pramstaller, V. Rijmen, A side-channel analysis resistant description of the AES S-box, in *Fast Software Encryption—FSE 2005*. Lecture Notes in Computer Science, vol. 3557 (Springer, Berlin, 2005), pp. 413–423

[31] T. Popp, M. Kirschbaum, T. Zefferer, S. Mangard, Evaluation of the masked logic style MDPL on a prototype chip, in *CHES 2007*. Lecture Notes in Computer Science, vol. 4727 (Springer, Berlin, 2007), pp. 81–94

[32] T. Popp, S. Mangard, Masked dual-rail pre-charge logic: DPA-resistance without routing constraints, in *Cryptographic Hardware and Embedded Systems—CHES 2005*. Lecture Notes in Computer Science, vol. 3659 (Springer, Berlin, 2005), pp. 172–186

[33] F. Regazzoni, A. Cevrero, F.-X. Standaert, S. Badel, T. Kluter, P. Brisk, Y. Leblebici, P. Ienne, A design flow and evaluation framework for DPA-resistant instruction set extensions, in *Cryptographic Hardware and Embedded Systems—CHES 2009*. Lecture Notes in Computer Science, vol. 5747 (Springer, Berlin, 2009), pp. 205–219

[34] C. Rolfes, A. Poschmann, G. Leander, C. Paar, Ultra-lightweight implementations for smart devices—security for 1000 gate equivalents, in *Smart Card Research and Advanced Application—CARDIS 2008*, ed. by G. Grimaud, F.-X. Standaert. Lecture Notes in Computer Science, vol. 5189 (Springer, Berlin, 2008), pp. 89–103

[35] A. Shamir, How to share a secret. *Commun. ACM* **22**(11), 612–613 (1979)

[36] Side-channel attack standard evaluation board (SASEBO). Further information are available via http://www.rcis.aist.go.jp/special/SASEBO/index-en.html

[37] D. Suzuki, M. Saeki, T. Ichikawa, DPA leakage models for CMOS logic circuits, in *CHES 2005*. Lecture Notes in Computer Science, vol. 3659 (Springer, Berlin, 2005), pp. 366–382

[38] Synopsys, Design compiler user guide—version A-2007.12. Available via http://tinyurl.com/pon88o, December 2007

[39] Synopsys, Power compiler user guide—version A-2007.12. Available via http://tinyurl.com/lfqhy5, March 2007

[40] National Security Agency. TEMPEST: a signal problem. *Cryptol. Spectr.* **2**(3), 1972 (declassified 2007)

[41] K. Tiri, M. Akmal, I. Verbauwhede, A dynamic and differential CMOS Logic with signal independent power consumption to withstand differential power analysis on smart cards, in *European Solid-State Circuits Conference—ESSCIRC 2002* (2002), pp. 403–406

[42] K. Tiri, I. Verbauwhede, A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation, in *Design, Automation and Test in Europe Conference—DATE 2004* (2004), pp. 246–251

[43] Virtual Silicon Inc. 0.18 µm VIP standard cell library tape out ready, part number: UMCL18G212T3, process: UMC logic 0.18 µm generic II technology: 0.18 µm, July 2004

[44] J. Waddle, D. Wagner, Towards efficient second-order power analysis, in *Cryptographic Hardware and Embedded Systems—CHES 2004*. Lecture Notes in Computer Science, vol. 3156 (Springer, Berlin, 2004), pp. 1–15

[45] M. Weiser, The computer for the 21st century. *ACM SIGMOBILE Mob. Comput. Commun. Rev.* **3**(3), 3–11 (1999)

[46] Xilinx, Virtex-II Pro and Virtex-II ProX Platform FPGAs: Complete data sheet. Available via http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf, November 2007

[47] S. Yang, W. Wolf, N. Vijaykrishnan, D.N. Serpanos, Y. Xie, Power attack resistant cryptosystem design: a dynamic voltage and frequency switching approach, in *Design, Automation and Test in Europe—DATE 2005* (IEEE Computer Society, Los Alamitos, 2005), pp. 64–69