J Hardw Syst Secur (2017) 1:203–218
https://doi.org/10.1007/s41635-017-0021-2

An Evaluation of Lightweight Block Ciphers for Resource-Constrained Applications: Area, Performance, and Security

Rajat Sadhukhan[1] · **Sikhar Patranabis**[1] · **Ashrujit Ghoshal**[1] · **Debdeep Mukhopadhyay**[1] · **Vishal Saraswat**[2] · **Santosh Ghosh**[3]

Received: 4 October 2017 / Accepted: 17 October 2017 / Published online: 3 November 2017

Abstract In March 2017, NIST (National Institute of Standards and Technology) has announced to create a portfolio of lightweight algorithms through an open process. The report emphasizes that with emerging applications like automotive systems, sensor networks, healthcare, distributed control systems, the Internet of Things (IoT), cyber-physical systems, and the smart grid, a detailed evaluation of the so called light-weight ciphers helps to recommend algorithms in the context of profiles, which describe physical, performance, and security characteristics. In recent years, a number of lightweight block ciphers have been proposed for encryption/decryption of data which makes such choices complex. Each such cipher offers a unique combination of resistance to classical cryptanalysis and resource-efficient implementations. At the same time, these implementations must be protected against implementation-based attacks such as side-channel analysis. In this paper, we present a holistic comparison study of four lightweight block ciphers, PRESENT, SIMON, SPECK, and KHUDRA, along with the more traditional Advanced Encryption Standard (AES). We present a uniform comparison of the performance and efficiency of these block ciphers in terms of area and power consumption, on ASIC and FPGA-based platforms. Additionally, we also compare the amenability to side-channel secure implementations for these ciphers on ASIC-based platforms. Our study is expected to help designers make suitable choices when securing a given application, across a wide range of implementation platforms.

Keywords Cyber physical systems · IoT · Security · Lightweight · Block ciphers · Side channels · Threshold implementations · ASIC · FPGA

1 Introduction

The advent of the era of Internet-of-Things (IoT) has given rise to a number of smart devices with the ability to communicate with each other across heterogeneous network interfaces. Among the application domains fueled by the growth of IoT are wireless sensor networks (WSNs) and RFID technology, that are used widely in industrial applications, medical monitoring, home automation, and traffic surveillance. The main constituents of any IoT framework are the numerous end nodes/devices, that are often constrained in terms of their memory capacity, processing speed, and power consumption rates. At the same time, these nodes commonly process sensitive data that needs

✉ Rajat Sadhukhan
rajat.sadhukhan@iitkgp.ac.in

Sikhar Patranabis
sikhar.patranabis@iitkgp.ac.in

Ashrujit Ghoshal
ashrujitg@iitkgp.ac.in

Debdeep Mukhopadhyay
debdeep@cse.iitkgp.ernet.in

Vishal Saraswat
vishal.saraswat@gmail.com

Santosh Ghosh
santosh.ghosh@intel.com

[1] Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur, India

[2] R. C. Bose Centre for Cryptology and Security, Indian Statistical Institute, Kolkata, India

[3] Intel Labs, Intel Corporation, Hillsboro, OR 97124, USA

to cryptographically protected against possible leakages to malicious adversaries. Traditional encryption mechanisms in the public and private-key settings are mostly resource-hungry, which makes them unsuitable for deployment in IoT devices. This has motivated the development of a large number of symmetric-key block ciphers that are *lightweight*, in the sense that they are area-efficient and/or low power consuming. Given such a rich class of ciphers, a designer is often faced with the challenge of judiciously choosing the appropriate cipher for a given application. This motivates a comparative study of popularly deployed lightweight block ciphers, not only in terms of their classical cryptographic security and implementation overheads/performance, but also in terms of their resistance to implementation-level attacks such as side-channel analysis (SCA). SCA is considered as a potent threat to nearly all cryptographic implementations in today's world, with the ability to compromise the security of even mathematically robust algorithms unless appropriately countered. This study is also in coherence with the recent declarations from NIST on the guidelines, and importance of design and analysis of lightweight block ciphers with physical security also an important design criteria [1].

1.1 Related Work

Existing surveys on lightweight block ciphers (LWC) have primarily targeted metrics related to hardware implementations (gate equivalent, area, hardware efficiency, throughput, power consumption, etc.) or software implementations (code size, software efficiency, RAM size, etc.), however, surprisingly not considering the cost for side channel protection. In [2], the authors have extensively studied the trends in hardware and software implementations of LWC, analyzing their suitability for different embedded devices in terms of hardware area (logic gates) and complexity. For software implementations, different LWCs were surveyed using RAM size requirement, execution time and code size as benchmarking criteria [3]. In [4] and [5], the authors have proposed benchmarking of LWCs targeting WSNs. Targeting usage of LWCs in automotive industries [6] surveyed the same in terms of hardware area (gate equivalent), energy consumption and latency while in [7] analyzes various LWCs with respect to energy as metric. Using ATMEL AVR ATtiny45 8-bit microcontroller as common platform performance of different LWCs have been evaluated w.r.t RAM usage, code size and cycle count for smart devices in [8]. In [9], side-channel resistance of lightweight weight block ciphers has been studied concerning software implementations, while [10] discusses mistakes in IEEE standard P1735 itself that allows successful launch of attack vectors to recover plaintext. To the best of our knowledge, there exist no prior studies that compare the additional overheads incurred when incorporating side-channel countermeasures in hardware across different LWC implementations. Our paper addresses this issue by comparing the area requirement of various LWCs, both with and without side-channel countermeasures, on both ASIC and FPGA platforms. This provides a designer with a holistic overview of the suitability of different block ciphers in resource constrained applications with varying security requirements and processing capabilities.

1.2 Our Contributions

This paper is meant to serve as a guideline for designers to choose between popularly deployed lightweight block ciphers for applications targeting either ASIC or FPGA platforms, with area/power consumption constraints. While a multitude of lightweight block ciphers have been recently proposed in the cryptographic literature, we choose four ciphers—PRESENT, SIMON, SPECK, and KHUDRA. The choice of ciphers is motivated by the fact that each of these ciphers differ widely from the other in terms of structure and implementation overhead.

PRESENT [11] is currently standardized by NIST as the international standard for lightweight block ciphers for hardware implementations targeting ASIC platforms. PRESENT is an SPN (substitution-permutation network) cipher with a set of non-linear S-Boxes (substitution boxes) and a bit-permutation based linear diffusion layer in each round. SIMON, proposed by NSA [12], is an ultra-lightweight block cipher family optimized for hardware implementations. It follows a balanced Feistel structure, with each round comprising of bitwise AND, XOR and circular left shift operations. While this allows for extremely area-efficient implementations, the rate of per-round diffusion for SIMON is low, thus increasing the necessary number of rounds for adequate security. SPECK, again proposed by NSA [12], is a family of block ciphers optimized for software implementations, targeting microcontrollers, although it is also suitable for hardware implementations. SPECK follows the add-rotate-xor (ARX) design paradigm, where the addition operation is modulo 2^n and the rotation operations include both left and right circular shifts. According to ECRYPT's stream cipher benchmarks (eBASC),[1] Speck is one of the fastest ciphers available, both for long as well as short plaintext messages. KHUDRA [13] is a lightweight block cipher targeted specifically for FPGA platforms. It uses a recursive Feistel structure and focuses on balancing the use of look-up-tables (LUTs) and registers, so as to minimize the requirement of FPGA slices. It also achieves a superior area-time product as compared to several existing lightweight block ciphers.

[1] https://bench.cr.yp.to/ebasc.html

The aforementioned lightweight block ciphers are compared against the Advanced Encryption Standard (AES), which serves as a baseline and highlights the resource savings that dedicated lightweight block ciphers can achieve. For each block cipher, we present an overview of the encryption/decryption algorithms, followed by a study of their security against classical cryptanalytic attacks such as linear and differential cryptanalysis. We then compare their implementation overheads in both ASIC and FPGA platforms. Note that FPGA devices are becoming increasingly popular for IoT applications owing to their easy reconfigurability, low power consumption, and low memory bandwidth requirements. Modern FPGAs are equipped with even more sophisticated features such as dynamic partial reconfiguration (DPR), that allows dynamic, energy-efficient non-invasive modification of the existing circuit on the FPGA, mostly to enhance functionality in the form of added plug-ins. The plug and play philosophy is particularly suitable for IoT applications since it supports multiple functions at a very large scale without the need for having dedicated hardware available at all times. In view of these advantages of FPGAs, we lay equal impetus on ASIC and FPGA implementations in this paper.

The focal point of this paper is side-channel security of lightweight block cipher implementations. The main factor in quantifying the security of an implementation against side-channel attacks is the number of traces required to recover the secret key. Hence, to protect against these attacks, several countermeasures have been proposed to increase the data complexity. One of the approaches often employed here is to decrease the signal-to-noise ratio (SNR) (i.e., the ratio of the variance of the leakage signal to the variance of the noise present in the device) by adding noise [14, 15]. However, these countermeasures are generally ad-hoc and hence provide only limited resistance. On the other hand, the countermeasures based on randomizing the intermediate variables (called *Masking*) had shown to be highly resistant against DPA attacks [16–18]. In this paper, we focus on a specific form of masking known as *threshold implementations* [19]. Threshold implementations are provably secure against first-order attacks and are based on multi-party computations and secret-sharing. They are, in particular, resistant against glitch-based attacks [19]. We present case-studies on threshold implementations of the aforementioned lightweight block ciphers. A comparison is presented in terms of the relative overhead of the threshold implementation of each cipher with respect to its corresponding unprotected implementation. The comparisons show that the choice of the algorithm is critical in deciding the cost of the side channel countermeasures: an aspect which can be critical for securing lightweight cryptosystems against physical attacks.

We briefly summarize the main contributions of the paper below:

1. The paper serves as a guideline for designers to choose between popularly deployed lightweight block ciphers for applications targeting either ASIC or FPGA platforms, with area/power consumption constraints.
2. The paper compares four lightweight block ciphers, PRESENT, SIMON, SPECK, and KHUDRA, in terms of their area and power consumption requirements on ASIC and FPGA-based platforms. The aforementioned lightweight block ciphers are compared against the Advanced Encryption Standard (AES), which serves as a baseline and highlights the resource savings that dedicated lightweight block ciphers can achieve.
3. We also study the classical cryptanalytic security of each of the aforementioned block ciphers laying special stress on their security against linear and differential cryptanalysis, as well as the state-of-the-art attacks on reduced round versions of these ciphers.
4. We finally compare side-channel secure implementations of the aforementioned lightweight ciphers on ASIC-based platforms. In particular, we focus on a specific form of implementation known as *threshold implementations* (TI) that are provably secure against first-order DPA attacks. For a fair comparison, we illustrate the relative overheads of these threshold implementations with respect to the corresponding unprotected implementations.

2 Brief Introduction of Lightweight Block Ciphers

In this section, we will briefly describe lightweight block ciphers that we have used in our analysis as test cases to implement them at common hardware platform (FPGA and ASIC design technology). A block cipher in general contains iterative round function, where a round function can be described as

$$Lin_1 Lin_2 \dots Lin_n(NonLin(x)),$$

where x is the message block and is subjected through a non-linear layer ($NonLin$) followed by $n (n \geq 1)$ linear layer (Lin). Compared to linear operation, a non-linear operation is far more expensive to implement at hardware level, so is designed as a weaker function to be operated on smaller message size than on the entire block size and the operation is applied in parallel to whole block. In order to compensate for the weak function, the linear operation is applied to the whole block at a time as a mixing operation. The mixing operation should be such that one bit change in the input x will change m components in the output y, given by *maximum distance separable* (MDS) matrix [20]. Structurally any block cipher can be broadly categorized either as Feistel structure [21] or Substitution Permutation Network (SPN) [22].

2.1 Advanced Standard Encryption (AES)-128/128

The AES encryption algorithm was first published in the year 2001 by National Institute of Standards and Technology (NIST) [23] and adopted as standard replacing DES. The arithmetic operations in AES operates over finite field $GF(2^8)$, using the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ for multiplication operations. The block cipher takes 128 bits (16 bytes) as message size or plaintext size as input, along with the key of length 128 bits (16 bytes), 192 bits (24 bytes), or 256 bits (32 bytes), consisting of 14 rounds, 12 rounds, and 10 rounds respectively, depending on the key size. Structurally, the AES algorithm belongs to the SPN family. The encryption and the key scheduling algorithms are shown in Algorithms 1 and 2, respectively. The block diagram of the encryption process is given in Fig. 1.

Algorithm 1 AES-128 encryption algorithm

```
Require: $Plaintext[128], Key[128]$
Ensure: $Ciphertext[128]$
  $k_0 \leftarrow Key$
  $State_1 \leftarrow Plaintext \oplus k_0$
  while $Round \neq 9$ do
      $k_{Round} \leftarrow AESgenerateKey(k_{Round-1})$
      $State_{Round+1} \leftarrow RoundFunction(State_{Round}, k_{Round})$
      $Round \leftarrow Round + 1$
  end while
  $Ciphertext \leftarrow RoundFunction(State_{10}, k_{10})$
```

Algorithm 2 AESgenerateKey for AES-128

```
Require: $k_i[128]$
Ensure: $k_{i+1}[128]$
  $tempVar \leftarrow k_i(column_3)$
  $tempVar \leftarrow SBoxLayer(tempVar)$
  $tempVar \leftarrow LeftCircularShift(tempVar, 1)$
  $tempVar[0] \leftarrow tempVar[0] \oplus roundConstant_i$
  $k_i(column_0) \leftarrow tempVar \oplus k_i(column_0)$
  $k_i(column_1) \leftarrow k_i(column_0) \oplus k_i(column_1)$
  $k_i(column_2) \leftarrow k_i(column_1) \oplus k_i(column_2)$
  $k_i(column_3) \leftarrow k_i(column_2) \oplus k_i(column_3)$
```

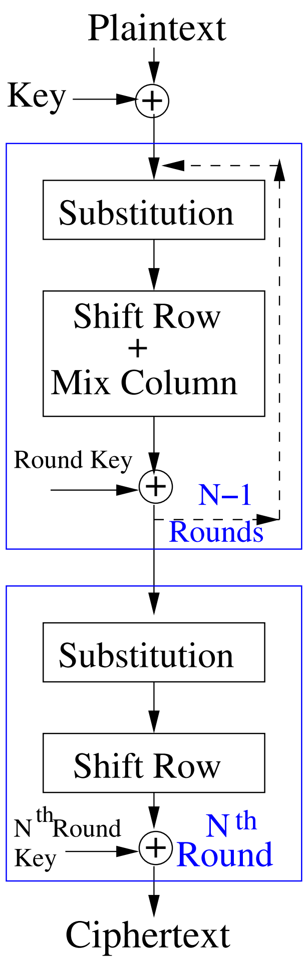

Fig. 1 AES block cipher operation

The RoundFunction step contains S-box layer substitution, the only non-linear operation, followed by linear operations ShiftRows, MixColumns and finally addition of round key. A state in AES contains four rows of 32 bytes data, so in shift row every row is given a left cyclic shift by the amount of that corresponding row number. For instance, row 2 will be given a left cyclic shift by two times. Mixcolumns is the mixing operation used in AES where every column of the state is multiplied by a matrix using $\mathcal{GF}(2)$ as shown below:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} s_0' \\ s_1' \\ s_2' \\ s_3' \end{bmatrix}$$

where s_0, s_1, s_2, s_3 is a column input. The final step is round key addition which is XORed with the MixColumns output.

2.2 PRESENT-64/80

PRESENT [11] is a hardware optimized lightweight cipher which comes with good features from DES and optimized hardware features from Serpent [24]. It falls in the SPN family too and has 31 rounds of operations, with a key size of 80-bits and plaintext text size of 64-bits. The encryption algorithm consist of 31 rounds where in each round key bits are XORed with message block of last round, followed by a substitution layer (operates over finite field $GF(2^4)$) as non-linear layer and a permutation layer (operates over finite field $GF(2^{64})$) as linear layer. The substitution layer uses 4×4 S-Box 16 times in parallel and 32 keys in total. The encryption algorithm has a post-whitening step after the 31st round, where the last generated key (K_{32}) from the key generation algorithm is used to for the purpose. To strengthen security, PRESENT can also take 128 bits key as input, and in that case, the only change will be in the key scheduling algorithm. The PRESENT encryption and the 80-bit key scheduling algorithm are presented in Algorithms 3 and 4, respectively. The block diagram of the encryption process is given in Fig. 2.

Algorithm 3 PRESENT encryption algorithm

```
Require: $Plaintext[64], Key[80]$
Ensure: $Ciphertext[64]$
  $Round \leftarrow 1$
  $State \leftarrow Plaintext$
  while $R \neq 31$ do
      $RoundKey_R \leftarrow PRESENTgenerateKey_{80}(Key, R)$
      $State \leftarrow State \oplus RoundKey_R$
      $State \leftarrow SBoxLayer(State)$
      $State \leftarrow PermutationLayer(State)$
      $R \leftarrow R + 1$
  end while
  $lastRoundKey \leftarrow generateKey(Key, R)$
  $State \leftarrow State \oplus lastRoundKey$
  $Ciphertext \leftarrow State$
```

Algorithm 4 $PRESENTgenerateKey_{80}$

```
Require: $Key[80], roundConstant$
Ensure: $roundKey[64]$
  $Key \leftarrow RightCircularShift(Key, 19)$
  $Key[76-79] \leftarrow SBoxLayer(Key[76-79])$
  $Key[15-19] \leftarrow Key[15-19] \oplus roundConstant$
  $roundKey[0:63] \leftarrow Key[16-79]$
```

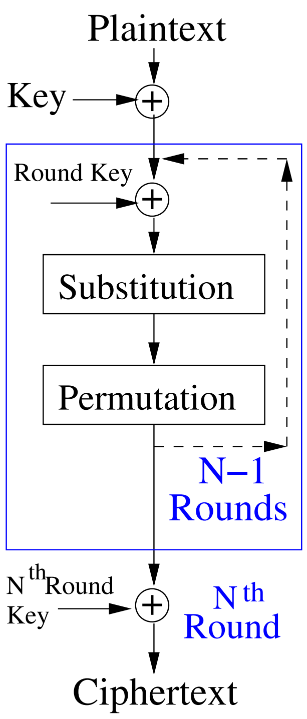

Fig. 2 PRESENT block cipher operation

2.3 KHUDRA-64/80

The block cipher KHUDRA [13] was designed keeping in mind the requirement of designing block ciphers which are also lightweight on FPGAs that are often preferred where reconfigurability and low development cost is a mandate. It has been shown that some algorithm choices like PRESENT are more apt for ASIC libraries with specialized library cells, while on FPGAs their compactness diminishes [13]. KHUDRA is unique in that way as its design process is with FPGAs as primary target. The encryption algorithm operates on 64-bits plaintext block, along with 80-bits key length to produce 64-bits ciphertext. The structure of KHUDRA belongs to the category of Feistel ciphers (generalized type-2 [25]) and consist of 18 rounds. The Feistel structure of KHUDRA has two parts: a permutation based on Feistel and F-function, where the F-function in turn contains substitution-permutation-substitution layer. The block cipher use S-Box layer of PRESENT as it has "High Algebraic Degree" and "Low Differential and Linear Probability" [11]. The number of rounds inside the F-function is 6. The key scheduling algorithm and encryption algorithm is shown in Algorithms 4 and 5, respectively. The block diagram of the encryption process is given in Fig. 3.

Algorithm 5 KHUDRA encryption algorithm

```
Require: $Plaintext[64]$, $Key[80]$
Ensure: $Ciphertext[64]$
  $R \leftarrow 0$
  while $R \neq 17$ do
      $branch_3[0:15] \leftarrow Plaintext[48-63]$
      $branch_1[0:15] \leftarrow Plaintext[16-31]$
      $internalRound \leftarrow 0$
      while $internalRound \neq 5$ do
          $internalbranch_3[0:3] \leftarrow Plaintext[60-63]$
          $internalbranch_1[0:3] \leftarrow Plaintext[52-55]$
          $Plaintext[60-63] \leftarrow SBoxlayer(Plaintext[60-63]) \oplus Plaintext[56-59]$
          $Plaintext[52-55] \leftarrow SBoxlayer(Plaintext[52-55]) \oplus Plaintext[48-51]$
          $Plaintext[56-59] \leftarrow internalbranch_1[0:3]$
          $Plaintext[48-51] \leftarrow internalbranch_3[0:3]$
          $internalbranch_3[0:3] \leftarrow Plaintext[28-31]$
          $internalbranch_1[0:3] \leftarrow Plaintext[20-23]$
          $Plaintext[28-31] \leftarrow SBoxlayer(Plaintext[28-31]) \oplus Plaintext[27-24]$
          $Plaintext[20-23] \leftarrow SBoxlayer(Plaintext[20-23]) \oplus Plaintext[16-19]$
          $Plaintext[24-27] \leftarrow internalbranch_1[0:3]$
          $Plaintext[16-19] \leftarrow internalbranch_3[0:3]$
          $internalRound \leftarrow internalRound + 1$
      end while
      $Plaintext[48-63] \leftarrow Plaintext[48-63] \oplus Plaintext[32-47] \oplus RoundKey[2 \times R + 1][0:15]$
      $Plaintext[16-31] \leftarrow Plaintext[16-31] \oplus Plaintext[0-15] \oplus RoundKey[2 \times R][0:15]$
      $Plaintext[32-47] \leftarrow branch_1[0:15]$
      $Plaintext[0-15] \leftarrow branch_3[0:15]$
      $R \leftarrow R + 1$
  end while
```

Algorithm 6 KHUDRAgenerateKey

```
Require: $Key[80]$
Ensure: $roundKey[16]$
  $whiteningKey_0[0:15] \leftarrow k_0 \leftarrow Key[0:15]$
  $whiteningKey_1[0:15] \leftarrow k_1 \leftarrow Key[16:31]$
  $whiteningKey_3[0:15] \leftarrow k_3 \leftarrow Key[49:63]$
  $whiteningKey_4[0:15] \leftarrow k_4 \leftarrow Key[64:79]$
  $k_2 \leftarrow Key[32:48]$
  $i \leftarrow 0$
  while $i \neq 35$ do
      $Constant \leftarrow 0||i[0:5]||00||i[0:5]||0$
      $roundKey_i \leftarrow k_{imod5} \oplus Constant$
      $i \leftarrow i + 1$
  end while
```

2.4 Simon-128/128 and Speck-128/128

Simon and Speck forms a family of block ciphers [12] where in each family there are around ten different types of block ciphers differing in block and key size, to be used based on the applications. If the block size in Speck or Simon is b-bits then the key size should be $a \times b/2$ where $(a \geq b)$. The linear and non-linear operations in Simon and Speck is only limited to modular arithmetic (addition or subtraction), circular shifts (right or left), and bitwise operations (and, or, xor). The nonlinear operation in Simon is due to bitwise and while in Speck, it is modular addition. Both Simon and Speck are based on Feistel-based structure, where the round function is the Feistel map. The round function in Simon is

$$RoundF(P_{left}, P_{right}) = (P_{right} \oplus A \oplus Key, P_{left})$$

where

$$A = (LCS(P_{left}, 1))\&(LCS(P_{left}, 8)) \oplus (LCS(P_{left}, 2))$$

and LCS does left circular shift.

The round function in Speck is given as

$$RoundF(P_{left}, P_{right}) = (A, LCS(P_{right}, x) \oplus A)$$

where

$$A = RCS(P_{left}, y) + P_{right}) \oplus key$$

and RCS does right circular shift, $x = 2$ and $y = 7$ when block size is 32-bits else $x = 3$ and $y = 8$.

The block diagram of the encryption process for Simon and Speck is given in Figs. 4 and 5, respectively.

3 Lightweight ASIC and FPGA Implementation and Comparison of Block Ciphers

3.1 ASIC Design Flow and Metrics

The starting point of our analysis process is to benchmark all the abovementioned lightweight block ciphers on a common ASIC and FPGA design platform and technology. In literature, we encounter often implementations of these ciphers on different technology nodes, which are then scaled. These scalings are often indicative and not accurate and therefore may provide an inaccurate evaluation. We have used an iterated design style for the block ciphers, with one round being performed in an iteration. The S-Box architecture we mention first is parallel, which means the entire block of input is passed through several bricks of S-Boxes simultaneously. The plaintext size, keysize, and number of rounds that we have used to implemented in our design is summarized in Table 1.

The CMOS technology we use to implement those block ciphers is 180 nm. Each block cipher design is taken through the RTL-to-GDS2 flow to estimate the area overhead and power consumption, along with other physical design implementation data. We have used Synopsys Design Compiler version I-2013.12-SP5-4 for synthesis and Synopsys IC-Compiler version J-2014.09-SP1 for placement and routing the design. For simulation, we have used Synopsys VCS version I-2014.03-SP1-1. Standard cell library TSL18FS120 from Tower Semiconductor Ltd. is used for

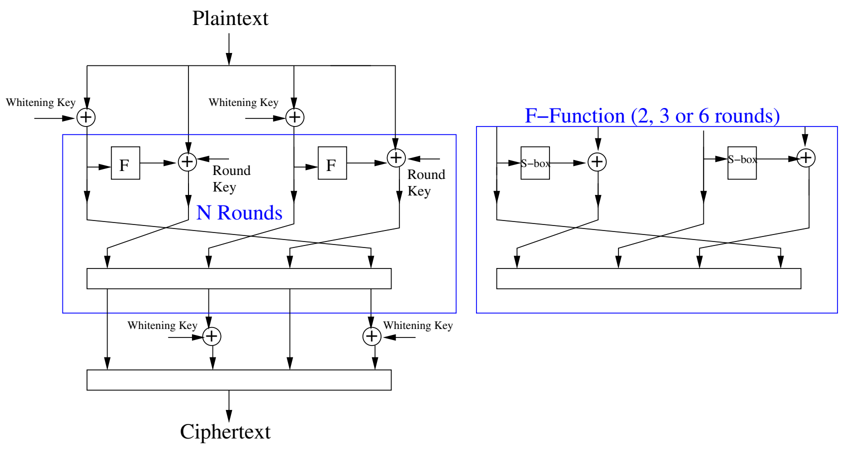

Fig. 3 KHUDRA block cipher operation

physical design. The standard cell library is characterized using SiliconSmart Software (Version: 2008.02-SP1p1) characterized under Fast-Fast process(P), 1.98 V voltage (V) and − 40 °C temperature (T). In typical condition, the library, hence the designs are expected to work at 1.8 V with normal junction temperature of 25 °C. The area is measured in gate equivalents (GE), where a GE is equal to the lowest area occupied by a 2-input NAND gate of 1x drive of given technology (for us 180 nm).

The areas of some basic cells in terms of Gate Equivalent (GE) values of the library that we have used is shown in Table 2, and is in accordance with that shown in [26]. The clock period has been fixed to 40ns (i.e., frequency = 25 MHz). With an estimate of the area of a 1x drive 2-input NAND gate being, 12.544 um^2, the gate equivalent (GE) of a chip is calculated as

$$GE = \frac{\text{chip area (in } um^2)}{\text{2-input NAND gate area (in } um^2)}.$$

We have also implemented the block ciphers on FPGA platforms using Virtex-5 family having XC5VFX100T device (FF1738 package) for implementation. To perform simulation and synthesis, we have used Xilinx ISE Design Suite (v14.2), while for power estimation we have used Xpower Analyzer (v14.2).

The results of ASIC and FPGA implementations are summarized in Tables 3 and 4, respectively. The ASIC implementation data show that AES even though has the best throughput, it has the highest GE. The impact on both static and dynamic power is also visible for AES, as compared to other ciphers. One of the fundamental reasons is the choice of 8 × 8 S-boxes, which makes AES resource hungry. Among the other ciphers, SPECK is more costly in the hardware implementations compared to its counterpart, SIMON, due to the use of integer modular additions. This also has an impact on the power consumption and makes SIMON a better hardware candidate for lightweight applications. KHUDRA, compares well with PRESENT, given that they have both 80-bit keys. The area requirement for KHUDRA is slightly lesser, while the better throughput for PRESENT is due to less number of rounds, followed by SPECK. However, as mentioned in the paper [13], the throughput of KHUDRA can be improved significantly at the cost of slight increase in area. The inner rounds could be unrolled at slight increase in both area and latency (note that each round is a small 16×16 Feistel structure, using 4×4 S-Boxes, and this reduces the overall clock cycles increasing

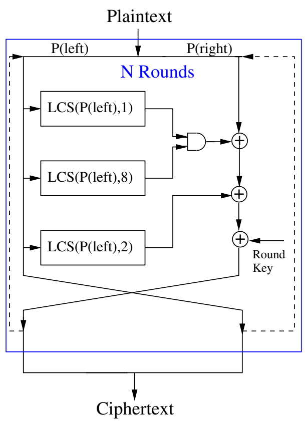

Fig. 4 SIMON block cipher operation

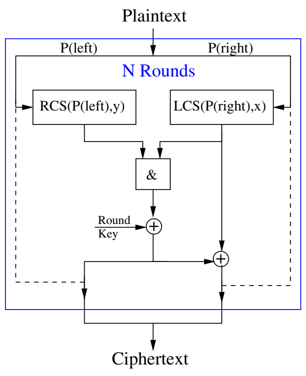

Fig. 5 SPECK block cipher operation

Table 1 Plaintext, key size, and number of rounds of block ciphers used for implementation

Block cipher	Plaintext size (in bits)	Key size (in bits)	Number of rounds	Sub-rounds
KHUDRA	64	80	18	6
AES	128	128	10	–
PRESENT	64	80	31	–
SIMON	128	128	68	–
SPECK	128	128	32	–

the throughput significantly. However, as in this evaluation, we mainly focus on the area, and power consumption, we do not elaborate into those design alternatives. Interested readers can look into [13]. Similar observations can be made for the implementations on FPGA platforms also.

Table 3 Block ciphers ASIC implementation

Specifications	KHUDRA	AES	SIMON	PRESENT	SPECK
Gate equivalent	1939.4	10654.7	2035	2031.2	4078.1
Frequency (MHz)	25	25	25	25	25
Clock period (ns)	40	40	40	40	40
Throughput (Mbps)	14.81	320	47.058	51.61	100
Leakage power (nW)	36.08	176.02	31.568	31.02	67.778
Dynamic power (mW)	0.935	2.481	1.004	0.842	1.5091

4 Attacks on Lightweight Block Ciphers

One of the fundamental challenges of designing lightweight block ciphers is that the underlying transformations involve simple operations as the target platforms are constrained with low processing and memory requirements. However, the primary concern being security, even with these simple operations one has to mitigate the various powerful adversaries.

Cryptanalytic attacks are evaluated based on the following parameters:

– *Space complexity*: The amount of memory required to store various internal or temporary data.
– *Time complexity*: The amount of computation time required to perform the attack. Often, it could be the number of times the cipher needs to be operated for doing the cryptanalysis.
– *Data complexity*: The number of plaintext and/or ciphertext pairs required to perform the attack.

A block cipher should follow *Kerckhoffs' principle* [27], which says, *A cryptosystem should be secure even if everything about the system, except the secret key, is in public knowledge*. For a cryptanalyst, the above principle gives rise to the following four attack scenarios:

– *Ciphertext-only attack*: The attacker has access to only ciphertexts.
– *Known-plaintext attack*: The attacker has also additional knowledge of the corresponding plaintext from which a ciphertext has resulted.
– *Chosen-plaintext/ciphertext attack*: The attacker can choose a number of plaintexts (and/or ciphertexts), and be given the corresponding ciphertexts (and/or plaintexts) by the encrytion (and/or decryption) oracle.
– *Adaptive chosen-plaintext and ciphertext attack*: The attacker can choose plaintexts (and/or ciphertexts) and be given the corresponding ciphertexts (and/or plaintexts) and based on the information obtained, iteratively the attacker can then ask further plaintexts/ciphertexts, and be given the corresponding ciphertexts/plaintexts.

Table 2 Gate equivalent of some cells in TSL18FS120

Gate	Gate equivalent
2-input NAND (1x)	1.0
2-input NOR (1x)	1.0
2-input OR (1x)	1.25
2-input AND (1x)	1.25
2-input XOR (1x)	2.75
2-input XNOR (1x)	2.75
NOT (1x)	0.75
2-1 MUX (1x)	2.0
DFF (1x)	4.75
1-bit full adder (1x)	4.75
Scan FF (1x)	7.5

Table 4 Block ciphers FPGA implementation

Block cipher	Register count	LUT count	Slice count	Power (W)
KHUDRA [13]	184	334	132	1.780
AES	407	2048	1004	1.786
PRESENT-64/80	165	218	109	1.78
SIMON-128/128	149	214	63	1.777
SPECK-128/128	448	599	215	1.794

In Table 5, we summarize the various attacks that have been launched under the above mentioned attack scenarios, comparing the data, space, and time complexities of the attacks.

The analysis shows that indeed while AES offers excellent security against the known and reported cryptanalysis, the other lightweight block ciphers have been quite resistant against most of these attacks, considering the cost of time, space, and data all taken simultaneously. Interestingly, the ciphers Simon and Speck, though they use very simple round functions indeed provide a good overall conventional security, due to its large number of rounds. While this may have an impact on latency, there could be applications where latency is of secondary criteria, compared to area, energy, and security. In the next section, we address the impact on the side channel countermeasures on these bare designs. It would be interesting to see whether the choice of the algorithms have an impact on the cost of these protections.

Table 5 Attack results on block ciphers

Block cipher	Attack scenario	Attack type	Number of rounds attacked	Data complexity	Space complexity	Time complexity	Attack sources
AES 128/128	Chosen-plaintext	Impossible differential	7	$2^{106.2}$	$2^{94.2}$	$2^{110.2}$	[28–32]
AES 128/128	Chosen-plaintext	Square	7	$2^{128}-2^{119}$	2^{64}	2^{120}	[33]
AES 128/128	Chosen-plaintext	Collision	7	2^{32}	2^{80}	2^{128}	[34]
AES 128/128	Chosen-plaintext	Meet-in-the-middle (MiTM)	7	2^{80}	2^{126}	2^{123}	[34–36]
AES 128/128	Adaptive chosen-plaintext	Boomerang attack	6	2^{71}	2^{33}	2^{71}	[37]
AES 128/128	Chosen-plaintext	Biclique MiTM	10	2^{88}	2^{8}	$2^{126.18}$	[38–41]
PRESENT -80	Known-plaintext	Multi-dimensional linear	26	2^{64}	–	2^{72}	[42, 43]
PRESENT -80	Chosen-plaintext	Saturation	24	2^{57}	2^{32}	2^{57}	[44–49]
PRESENT -80	Known-plaintext	Linear	24	$2^{63.5}$	–	–	[50, 51]
PRESENT -80	Chosen-plaintext	Differential	16	2^{64}	2^{32}	2^{64}	[52, 53]
PRESENT -80	Chosen-plaintext	Structure attack	18	2^{64}	–	2^{76}	[54]
PRESENT -80	Chosen-plaintext	Statistical saturation attack	24	2^{57}	2^{32}	2^{57}	[44]
PRESENT -80	Chosen-plaintext	Biclique	31	2^{23}	–	$2^{79.76}$	[55, 56]
KHUDRA	Chosen-plaintext	Meet-in-the-middle (MiTM)	14	2^{51}	$2^{64.86}$	$2^{66.19}$	[57]
KHUDRA	Known-plaintext	Guess-and-determine	14	2	*negligible*	2^{64}	[58]
Speck 128/128	Chosen-plaintext	Differential	17	2^{113}	2^{22}	2^{113}	[59, 60]
Simon 128/128	Chosen-plaintext	Impossible differential	22	$2^{129.226}$	$2^{123.203}$	$2^{187.527}$	[61]
Simon 128/128	Chosen-plaintext	Differential	40	$2^{124.796}$	2^{64}	$2^{120.474}$	[61]

5 Overhead Analysis of Side Channel Countermeasures Using Threshold Implementations

In spite of resistance against classical cryptography, hardware implementations of block ciphers have been challenged by side channel analysis [62, 63]. This powerful class of adversaries tries to exploit the correlation between the measured information, which are referred to as side channels, with the internal data, and eventually the secret key. Many applications like smart meters, automotive components are at the disposal of the adversary, and the physical possession make this threat potent. The exact form of side channels, like power, electromagnetic (EM) radiations, sound, faults, could depend on the applications. As IoT devices are more vulnerable to power and EM radiation based side channel attacks due to their insecure and vulnerable installations, strong and efficient countermeasures are required to protect them. However, though sound countermeasures exist, they come with a formidable cost, making it really challenging for resource constrained environments. Hence, it is an open area of research as to develop cipher operations which are not only lightweight, but also have a lesser cost due to the additions of the countermeasures. In our analysis, we analyze the lightweight ciphers mentioned, wrt. such a sound countermeasure, called *Threshold-Implementation* (TI) [19].

The main purpose of any side channel attack countermeasure pertaining to hardware or software level is to make all cryptographic computations independent of the secret key. To achieve the same use of randomization in data, data hiding and masking are the most prominent methods. But all these methods require additional random values or masks for non-linear layer computations and for intermediate values produced during a round, which otherwise would cause information leakage making it vulnerable to SCA. The presence of circuit level glitches can also lead to leakage of side channel information has been reported in literature. So, we focus on *TI* as a sound countermeasure protecting the designs provably against first-order and higher order side channel attacks even in presence of glitches. This approach is based on secret sharing method [64], where a function having algebraic degree d is implemented with $d+1$ shares to resist first-order side channel attack. Let P and K be the input plaintext and key, respectively. Then, for a d-share *TI* P and K can be represented as

$$\begin{aligned} P &= p_1 \oplus p_2 \oplus p_3 \oplus \cdots \oplus p_d \\ K &= k_1 \oplus k_2 \oplus k_3 \oplus \cdots \oplus k_d \end{aligned}$$

where P and K are n-bit vectors. Let, $Y = f_{nl}(P, K)$, where $f_{nl}(f_{nl} : GF(2^n) \mapsto GF(2^n))$ be a non-linear function. Then for the d-share implementation, there would be d non-linear functions to process d-output shares of Y such that each share processed fulfills three requirements as follows:

- Correctness: This property ensures that when outputs of different shares are combined, the original output can be retrieved in a correct way.

$$\begin{aligned} P &= p_1 \oplus p_2 \oplus p_3 \oplus \cdots \oplus p_d \\ K &= k_1 \oplus k_2 \oplus k_3 \oplus \cdots \oplus k_d \\ Y &= y_1 \oplus y_2 \oplus y_3 \oplus \cdots \oplus y_d \,. \end{aligned}$$

- Uniformity: This property ensures uniform distribution of input share should also result in uniform distribution in output share
- Non-completeness: This property ensures that the equation to evaluate any output share should be independent of at least one input share for each variable. It is because of this property, in spite of the presence of glitches in the circuit secret information is not revealed as at any instant of time not all shares are present to compute the output

$$\begin{aligned} y_i = {} & f_{nl}^{(i)}(x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d, k_1, k_2, \ldots, \\ & k_{i-1}, k_{i+1}, \ldots, k_d), \end{aligned}$$

where, $i \in \{1 \ldots d\}$ and $f_{nl}^{(i)}$ is i-th share function.

As evident from this description, because of the additional shares, the cost of the design increases manifold, making them unsuitable for resource constrained devices. So, we evaluate based on the lightest known 3-share *TI* of the block ciphers in terms of area*(Gate Equivalent)* in ASIC, and the side channel leakage. For side channel evaluation, we resort to the FIPs conformance style of evaluation following the Test-Vector-Leakage Assessment (TVLA) which is based on statistical hypothesis testing using T test [65]. The TVLA evaluations have been performed on the standard SASEBO G-II evaluation platforms [66]. We have also implemented a 3-share *TI* for the KHUDRA block cipher for which there was no reported design. It may be emphasized here that 2-share implementations would have been lighter than 3-share TI implementations but it violates the *non-completeness* property. In the case of 2-share *TI* processing, an output share is always a function of both the shares as illustrated in the below example:

Let $f(x, y) = x \cdot y$, and $x_1,\ x_0,\ y_1,\ y_0$ be shares of x and y such that $x = x_1 + x_0$ and $y = y_1 + y_0$. Then the output shares are processed as

$$\begin{aligned} f_1 &= x_0 \cdot y_0, & f_2 &= x_1 \cdot y_1 \\ f_3 &= x_1 \cdot y_0, & f_4 &= x_0 \cdot y_1 \,. \end{aligned}$$

Then, all possible combinations of f_1, f_2, f_3, f_4 to obtain two output shares would violate the non-completeness property. For example, combining f_1 and f_2 would need all the input shares, while combining f_1 and

f_3 would need all the input shares of x. Likewise, f_1 with f_4 would require all the input shares of y. This makes 2-TI insecure against side channels in the presence of glitches.

The general strategy for 3-share *TI* implementations of block ciphers is shown in Fig. 6. The shared implementation architecture consists of three parallel instances of linear layer, one for each share, and one shared non-linear unit having the 3-share functions as described above processing all the shares together. In Fig. 6, each instance of linear function $Linear$ processes an input share $X[i]$ and key share $K[i]$ to produce intermediate share $M[i]$ where i is the i^{th} share ($1 \leq i \leq 3$). All the $M[i]$ vectors are fed into the shared $NonLinear$ function to produce new output share $Y[i]$ vectors.

5.1 Test Vector Leakage Assessment (TVLA): *T* Test Methodology

The TVLA test is a conformance test which attempts to detect the presence of any leakage in a cryptographic core. The block cipher hardware is made to operate on a constant selected plaintext, and the power consumption is compared with when the cipher operates on randomly chosen inputs. The existence of any differentiability denotes presence of leakage of information which can be potentially exploited by an adversary for key recovery. The basis of the test is statistical hypothesis testing using Welch's T test. We state that briefly. Interested readers are requested to refer [67].

Suppose, the number of power traces collected for a fixed plaintext denoted by $|\mathscr{A}|$ for set $\mathscr{A}$, and number of power traces processing random inputs denoted by $|\mathscr{B}|$ for set $\mathscr{B}$. The sample mean, variance for $\mathscr{A}$ is denoted by $\mu_{\mathscr{A}}$ and $\sigma^2_{\mathscr{A}}$ respectively and similar for $\mathscr{B}$ as well. Then, a null hypothesis is made with $\mu_{\mathscr{A}}$=$\mu_{\mathscr{B}}$, after which Welch's T test is used to accept or reject the null hypothesis with a confidence of 99.9%. The formula for T test is shown below:

$$t = \frac{\mu_{\mathscr{A}} - \mu_{\mathscr{B}}}{\sqrt{\frac{\sigma^2_{\mathscr{A}}}{|\mathscr{A}|} + \frac{\sigma^2_{\mathscr{B}}}{|\mathscr{B}|}}}.$$

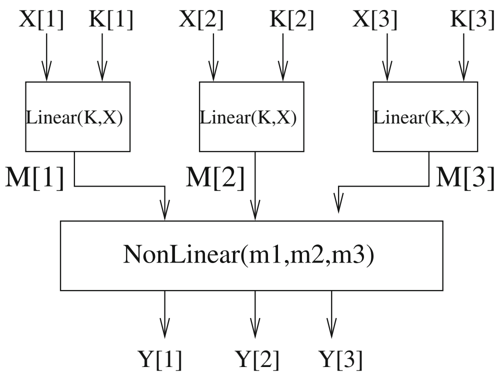

Fig. 6 3-share TI block diagram

If the value of $t > |4.5|$, then the null hypothesis is rejected and the cryptographic algorithm is said to fail first-order leakage.

In the following sections, we discuss the 3-TI implementations of the lightweight ciphers which have been designed in literature. In particular, KHUDRA had not been designed using TI, so we present a TI implementation of the block cipher to compare the overheads wrt. an unprotected design.

5.2 3-TI Simon-128/128

In the bit-serialized *Threshold Implementation* of Simon as shown in Fig. 7 and reported in [68], in every clock cycle, *1-bit* of internal state is processed. So if the circuit takes *n-bit* size plaintext as input, *n/2-clock cycles* will be required to complete the whole round.

In Fig. 7, $S1$ and $S2$ are shift-registers while $F1$ and $F2$ are FIFO (First-In-First-Out) channels. During the first phase (marked as 1 in Fig. 7), the look-up-table (LUT) processes 3-bits from $S1$, during first eight clock cycles of every round, a *key* bit and output of $F2$, storing the result in $S2$. So, during this phase, the new values are stored in $S2$, while old values are consumed from $S1$ for processing. Once $S2$ gets full, $S2$ gets connected to $F1$ which will store new values. This phase is the second phase (marked as 2 in Fig. 7) and continues for 56 clock cycles. The functionalities of $S1$ and $S2$ get swapped in next round, again having two phases. We choose to cover *3-share TI* hardware architecture [69] of the Simon-128/128 as part of our study as the *3-share* implementation adheres to *correctness*, *non-completeness*, and *uniformity* requirements of *TI* as compared to a *2-share* implementation which does not fulfill *non-completeness* property. Let, $P = L^0 \parallel R^0$ be the plaintext having L (left) and R (right) components as input to Simon (superscript 0 represents round). For the 3-share structure, two random masks are generated each for L^0 and R^0 and expressed as

$$L^0 = l^0_1 \oplus l^0_2 \oplus l^0_3 \text{ } (l^0_1 \text{ and } l^0_2 \text{ are random})$$
$$R^0 = r^0_1 \oplus r^0_2 \oplus r^0_3 \text{ } (r^0_1 \text{ and } r^0_2 \text{ are random})$$

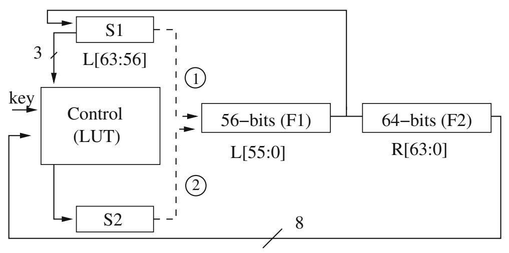

Fig. 7 Bit-serial simon architecture

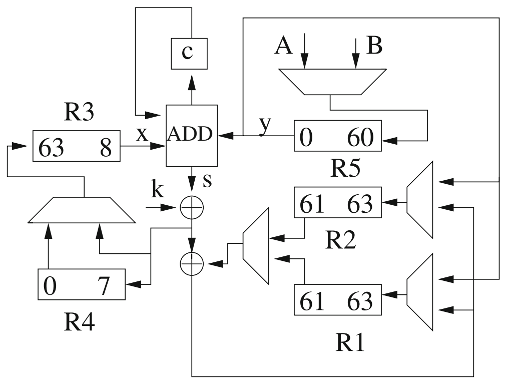

Fig. 8 Bit-serial speck architecture

The above masking method is applicable for generating 3-shares of key as well. The left and right component of each share is processed as given by the following equations:

$$
\begin{aligned}
l_1^{i+1} &= r_2^i \oplus S^2(l_2^i) \oplus S^1(l_2^i)\cdot S^8(l_2^i) \oplus S^1(l_3^i)\cdot S^8(l_2^i)\\
&\quad \oplus S^1(l_2^i)\cdot S^8(l_3^i) \oplus k_2^i\\
l_2^{i+1} &= r_3^i \oplus S^2(l_3^i) \oplus S^1(l_3^i)\cdot S^8(l_3^i) \oplus S^1(l_3^i)\cdot S^8(l_1^i)\\
&\quad \oplus S^1(l_1^i)\cdot S^8(l_3^i) \oplus k_3^i\\
l_3^{i+1} &= r_1^i \oplus S^2(l_1^i) \oplus S^1(l_1^i)\cdot S^8(l_1^i) \oplus S^1(l_2^i)\cdot S^8(l_1^i)\\
&\quad \oplus S^1(l_1^i)\cdot S^8(l_2^i) \oplus k_1^i\\
r_1^{i+1} &= l_1^i\\
r_2^{i+1} &= l_2^i\\
r_3^{i+1} &= l_3^i
\end{aligned}
$$

The linear part of both data-path and key schedule units are instantiated three times one for each share as shown in Fig. 6. The hardware overhead with and without TI implementation are shown in Table 6.

5.3 3-TI Speck-128/128

The architecture of bit-serialized implementation of Speck [70] is shown in Fig. 8. Each round of the cipher in bit-serialized mode requires 64 clock cycles, so a total of 2048 clock cycles are required for 32 rounds. The 128-bit plaintext is divided into left and right component each of 64-bit length. Register R3 (54-bits) and R4 (8-bits) hold 64-bit left part of plaintext while register R5 (61-bits) along with R1/R2 (3-bits) hold the right part and $A = R2[61]$ and $B = R1[61]$. Starting at 8th-bit the feedback function of the left register accepts 1-bit(x) from register R3, 1-bit(y) from register R5 and performs ADD (full adder) operation followed by XOR. To implement three share implementation, three copies of bit-serial implementation are instantiated, each unit processing every share as shown in Fig. 6. In every round, Speck has a cyclic rotation and XOR operations which are linear and a non-linear addition operation. The plaintext and key is split into 3-shares, where 2-shares are generated randomly and third being computed. The linear operations can process each share separately while non-linear layer process the shares according to the following equations:

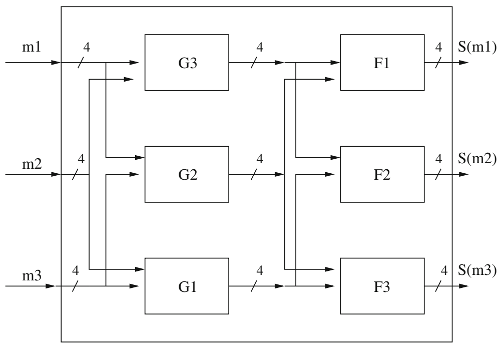

Fig. 10 Decomposed S-box with three shares

$$
\begin{aligned}
s_1^i &= x_1^i \oplus y_1^i \oplus c_1^i\\
s_2^i &= x_2^i \oplus y_2^i \oplus c_2^i\\
s_3^i &= x_3^i \oplus y_3^i \oplus c_3^i\\
c_1^{i+1} &= x_2^i y_2^i \oplus x_2^i y_3^i \oplus x_3^i y_2^i \oplus x_2^i c_2^i \oplus x_2^i c_3^i \oplus x_3^i c_2^i \oplus y_2^i c_2^i\\
&\quad \oplus y_2^i c_3^i \oplus y_3^i c_2^i\\
c_2^{i+1} &= x_3^i y_3^i \oplus x_3^i y_1^i \oplus x_1^i y_3^i \oplus x_3^i c_3^i \oplus x_3^i c_1^i \oplus x_1^i c_3^i \oplus y_3^i c_3^i\\
&\quad \oplus y_3^i c_1^i \oplus y_1^i c_3^i\\
c_3^{i+1} &= x_1^i y_1^i \oplus x_1^i y_2^i \oplus x_2^i y_1^i \oplus x_1^i c_1^i \oplus x_1^i c_2^i \oplus x_2^i c_1^i \oplus y_1^i c_1^i\\
&\quad \oplus y_1^i c_2^i \oplus y_2^i c_1^i
\end{aligned}
$$

Fig. 9 PRESENT (w = 4-bits) serial implementation

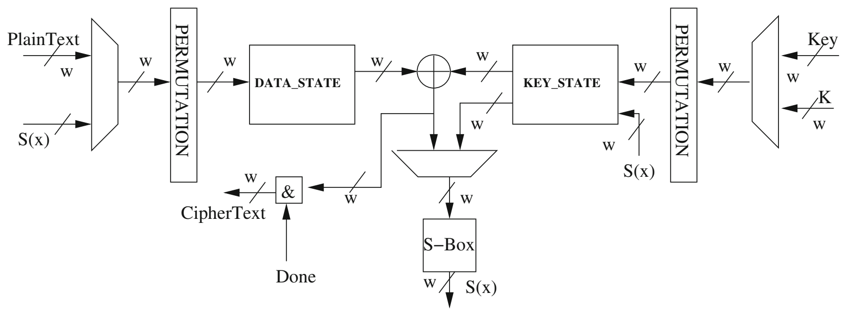

Table 6 Hardware overhead comparison with and without threshold implementation in terms of gate equivalents

Block cipher	Tech. node (nm)	Tech. lib.	Unprotected (GE)	Protected (GE)	Area overhead
KHUDRA -64/80 [13]	180	TSL18FS120 tower Semiconductor ltd.	1090	3738	2.5x
PRESENT -64/80 [76]	45	NanGate 45 nm open Cell library	1619.23	5236.49	2.2x
SIMON -128/128 [69]	90	TSMC 90 nm cell library	1234	5686	3.6x
SPECK -128/128 [26]	180	TSL18FS120 Tower Semiconductor ltd.	2018	5940.6	1.9x
AES-128/128 [74]	45	NanGate 45 nm open Cell library	2421	6340	1.6x

where x^i_j and y^i_j are i-th bits of two words, c^i_j denotes carry bit and j indicates j-th share. So,

$$\begin{aligned} x^i &= x^i_1 \oplus x^i_2 \oplus x^i_3 \\ y^i &= y^i_1 \oplus y^i_2 \oplus y^i_3 \\ c^i &= c^i_1 \oplus c^i_2 \oplus c^i_3 \,. \end{aligned}$$

The hardware overhead with and without TI implementation are shown in Table 6.

5.4 3-TI PRESENT-64/80

The algorithm description of Present has already been introduced before in this paper. The algebraic degree of Present S-box is 3, so the S-box function can be divided into two quadratic functions, where each quadratic function can be decomposed into 3-shares following uniform, non-completeness and correctness properties of *Threshold Implementation* countermeasure against side channel attacks. The 4-bit serialized implementation [71] of Present is shown in Fig. 9. The $DATA\_State$ module consist of 16 4-bit registers which can operate both in serial and parallel mode. In serial mode, it forwards 4-bit data to next stage while in parallel mode it performs permutation using one clock cycle. Similar function is performed by KEY_State register which consist of 20 4-bit wide register. So, a round of PRESENT requires 16 clock cycles to perform substitution layer and 1 clock cycle to perform permutation layer. Each round of serialized implementation takes 17 clock cycles, so for 31 rounds a total of 547 ($17 \times 31 + 20$) clock cycles are required, where 20 clock cycles are required at initialization phase comprising of key and data loading. After completion of 31 rounds *Done* signal becomes 1 to load the final ciphertext.

The S-box (non-linear) function $S(x)$ ($S : GF(2^4) \mapsto GF(2^4)$) can be decomposed into $G(x)$ ($G : GF(2^4) \mapsto GF(2^4)$) and $F(x)(F : GF(2^4) \mapsto GF(2^4))$, where $S(x) = F(G(x))$. Each of $F(x)$ and $G(x)$ is split into three shares as shown in Fig. 10 where,

$$F(m_1 \oplus m_2 \oplus m_3) = F1(m_2, m_3) \oplus F2(m_1, m_3) \oplus F3(m_1, m_2)$$

and

$$x = m_1 \oplus m_2 \oplus m_3 \ (m_1, m_2, m_3 \text{ shares of } x)\,.$$

For 3-share *TI* implementation the linear layer units are instantiated three times one for each share, with one copy of shared S-box unit as shown in Fig. 6. The hardware overhead with and without TI implementation are shown in Table 6. The details of equations of each share inside the S-box is given in [71].

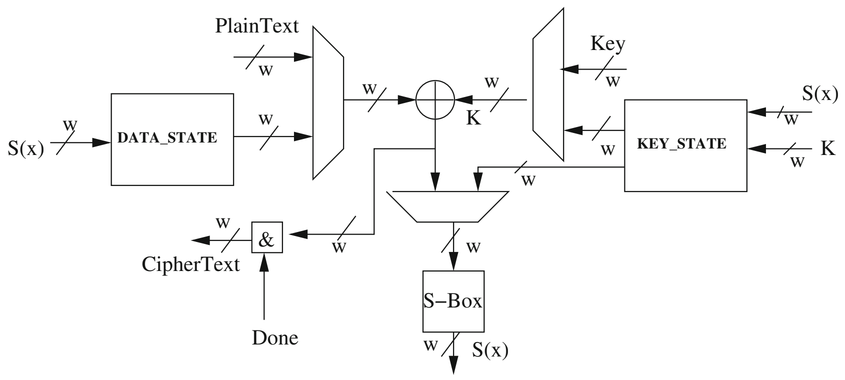

Fig. 11 AES(w = 8-bits) serial implementation

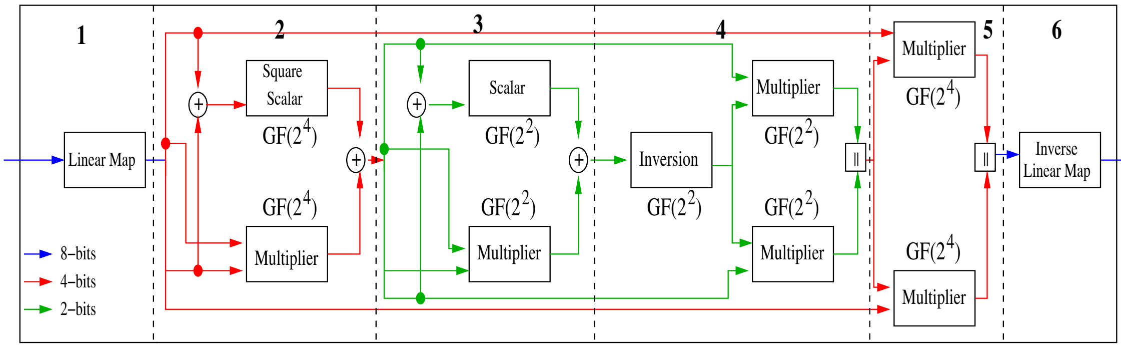

Fig. 12 AES S-box

5.5 3-TI Advanced Standard Encryption (AES)-128/128

The architecture of byte serial implementation of AES is based on compact structure of the AES S-box proposed in [72] (Fig. 11). The design is based on sub-field isomorphisms which executes linear mapping transformations on the input of S-box by changing the basis from $GF(2^8)$ to $GF(2^4)$ and then $GF(2^2)$. The results from the smaller field are then combined back to the original basis $GF(2^8)$, with a final inverse linear mapping transformation to result in the output of the S-box as shown in Fig. 12. More details of these mappings can be found in [73]. The stages of the S-box and how it can be extended to 3 share *Threshold Implementation* has been throughly discussed in [74]. The byte-serial architecture [75] is shown in Fig. 11. The $DATA\_State$ and KEY_State modules consist of 16 8-bit registers. Each round of AES requires 21 clock cycles per round with additional 16 clock cycles to load the ciphertext, so in total 226 clock cycles are required for encryption. Three share masking can be implemented using a shared S-box as described in [74]. The architecture will consist of three instances of linear layer each processing a share as shown in Fig. 6 along with one instance of the shared S-box. Due to the presence of pipelining stages in the shared S-box, 4 additional clock cycles are required along with 21 clock cycles per round, so in total 266 clock cycles are required for the encryption. The hardware overhead with and without TI implementation is shown in Table 6.

5.6 3-TI KHUDRA-64/80

In order to complete this study, we perform a 3-TI implementation of KHUDRA, on which there are no reported works. The architecture of a 4-bit serialized implementation of KHUDRA is shown in Fig. 13. KHUDRA uses the same S-box as that of Present, so the equations for 3 share *TI* implementation of shared S-box of Present is applicable here as well. The F-function block for KHUDRA, as shown in Fig. 13, has 6 rounds and iterates over 24 clock cycles, where in each 4 clock cycles one round of F-function gets executed. So, the left and right branch of F-function that pass through S-box takes 48 clock cycles to complete the whole F-function round and in total 864 (48 × 64) clock cycles for 18 rounds. The *TI* shared S-box will have one instance while other components will have three instances, one for each share. The hardware overhead with and without TI implementation are shown in Table 6.

The increase in hardware overhead of unprotected design with their protected counter-part are noted in Table 6. We observe that the overheads range from 1.6 to 3.6. Interestingly with the TI protection the GE of AES does not blow up as high as that of some of the more lightweight ciphers, like SIMON. Hence, we conclude with the comment that although a cipher without protection may be lightweight, the cost along with the side channel protections is also a very important design choice.

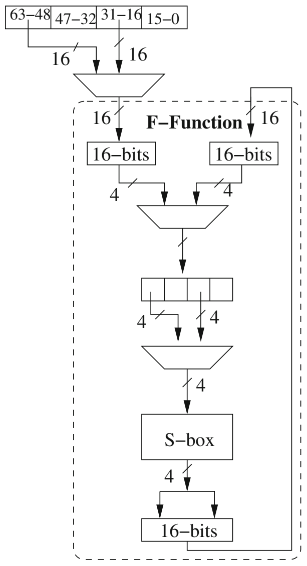

Fig. 13 KHUDRA serial implementation

6 Conclusion

The importance of securing IoT devices along with meeting the resource constraints requirements of IoT is both challenging and of utmost importance. Amidst the plethora of block ciphers, it is a difficult design choice to select appropriate ciphers for a given application. In this paper, we present a comprehensive analysis and study of some chosen lightweight block ciphers with various design principles and evaluate them wrt. area overhead, throughput, power consumption, security against classical cryptanalysis and side channel security. We evaluate hardware implementations of SIMON, SPECK, PRESENT, KHUDRA, and AES on both ASIC and FPGA platforms. Notably, we evaluate the unprotected designs on the same ASIC technology, and bring out the extremes in these ciphers which could be suitable to various applications. Furthermore, we evaluate 3-TI implementations of these ciphers as a sound countermeasure against side channels and show that different crypto-algorithms have different overheads wrt. these countermeasures. The study can be further developed in coherence to the recent call from NIST, to design and analyze lightweight ciphers with physical security as also an important design criteria.

Acknowledgements The authors would like to thank Intel Corporation, USA, for the partial funding of the work under the project "LightCrypto: Ultra-Light-weight Robust Crypto-Architectures for Performance and Energy."

References

1. McKay KA, Bassham L, Turan MS, Mouha N (2016) Report on lightweight cryptography. NIST DRAFT NISTIR 8114
2. Hatzivasilis G, Fysarakis K, Papaefstathiou I, Manifavas C (2017) A review of lightweight block ciphers. J Cryptogr Eng. https://doi.org/10.1007/s13389-017-0160-y
3. Dinu D, Le Corre Y, Khovratovich D, Perrin L, Großschädl J, Biryukov A (2015) Triathlon of lightweight block ciphers for the internet of things. IACR Cryptology ePrint Archive 2015:209
4. Cazorla M, Marquet K, Minier M (2013) Survey and benchmark of lightweight block ciphers for wireless sensor networks. In: SECRYPT. SciTePress, pp 543–548
5. Roman R, Alcaraz C, Lopez J (2007) A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes. MONET 12(4):231–244
6. Ghosh S, Misoczki R, Zhao L, Sastry MR (2017) Lightweight block cipher circuits for automotive and iot sensor devices. In: Proceedings of the hardware and architectural support for security and privacy, HASP '17. ACM, New York, NY, USA, pp 5:1–5:7
7. Kerckhof S, Durvaux F, Hocquet C, Bol D, Standaert F-X (2012) Towards green cryptography: a comparison of lightweight ciphers from the energy viewpoint. In: CHES, vol 7428 of LNCS. Springer, pp 390–407
8. Balasch J, Ege B, Eisenbarth T, Gérard B, Gong Z, Güneysu T, Heyse S, Kerckhof S, Koeune F, Plos T, Pöppelmann T, Regazzoni F, Standaert F-X, Van Assche G, Van Keer R, van Oldeneel tot Oldenzeel L, von Maurich I (2012) Compact implementation and performance evaluation of hash functions in attiny devices. In: CARDIS, vol 7771 of LNCS. Springer, pp 158–172
9. Heuser A, Picek S, Guilley S, Mentens N (2017) Side-channel analysis of lightweight ciphers: does lightweight equal easy? IACR Cryptology ePrint Archive 2017:261
10. Chhotaray A, Nahiyan A, Shrimpton T, Forte DJ, Tehranipoor M (2017) Standardizing bad cryptographic practice—a teardown of the ieee standard for protecting electronic-design intellectual property. Cryptology ePrint Archive Report 2017:828
11. Bogdanov A, Knudsen LR, Leander G, Paar C, Poschmann A, Robshaw MJB, Seurin Y, Vikkelsoe C (2007) PRESENT: An ultra-lightweight block cipher. In: CHES, vol 4727 of LNCS. Springer, pp 450–466
12. Beaulieu R, Shors D, Smith J, Treatman-Clark S, Weeks B, Wingers L (2015) SIMON And SPECK: block ciphers for the internet of things. IACR Cryptology ePrint Archive 2015:585
13. Kolay S, Mukhopadhyay D (2014) Khudra: a new lightweight block cipher for fpgas. In: SPACE, vol 8804 of LNCS. Springer, pp 126–145
14. Benini L, Macii A, Macii E, Omerbegovic E, Pro F, Poncino M (2003) Energy-aware design techniques for differential power analysis protection. In: Proceedings of the 40th design automation conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003, pp 36–41
15. Yang S, Wolf W, Vijaykrishnan N, Serpanos DN, Xie Y (2005) Power attack resistant cryptosystem design: a dynamic voltage and frequency switching approach. In: 2005 design, automation and test in europe conference and exposition (DATE 2005), 7–11 March 2005, Munich, Germany, pp 64–69
16. Akkar M-L, Giraud C (2001) An implementation of des and aes, secure against some attacks. In: Cryptographic hardware and embedded systemsCHES 2001. Springer, pp 309–318
17. Standaert F-X, Peeters E, Quisquater J-J (2005) On the masking countermeasure and higher-order power analysis attacks. In: International conference on information technology: coding and computing, 2005. ITCC 2005, vol 1. IEEE, pp 562–567
18. Maghrebi H, Danger J-L, Flament F, Guilley S, Sauvage L (2009) Evaluation of countermeasure implementations based on boolean masking to thwart side-channel attacks. In: 3rd international conference on signals, circuits and systems (SCS), 2009. IEEE, pp 1–6
19. Nikova S, Rechberger C, Rijmen V (2006) Threshold implementations against side-channel attacks and glitches. In: ICICS, vol 4307 of LNCS. Springer, pp 529–545
20. Gupta KC, Ray IG (2013) On constructions of MDS matrices from companion matrices for lightweight cryptography. In: CD-ARES workshops, vol 8128 of LNCS. Springer, pp 29–43
21. Feistel H (1973) Cryptography and computer privacy. Sci Am 228(5):15–23
22. Katz J, Lindell Y (2007) Introduction to modern cryptography. Chapman and Hall/CRC Press
23. National Institute of Standards and Technology (2001) Advanced encryption standard (AES). Federal Information Processing Standards Publication 197(441):1–47
24. Biham E, Anderson RJ, Knudsen LR (1998) Serpent: a new block cipher proposal. In: FSE, vol 1372 of LNCS. Springer, pp 222–238
25. Hoang VT, Rogaway P (2010) On generalized feistel networks. In: CRYPTO, vol 6223 of LNCS. Springer, pp 613–630
26. Yang G, Zhu B, Suder V, Aagaard MD, Gong G (2015) The simeck family of lightweight block ciphers. In: CHES, vol 9293 of LNCS. Springer, pp 307–329
27. Kerckhoffs A (1883) La cryptographie militaire. Journal Des Sciences Militaires IX:5–83
28. Mala H, Dakhilalian M, Rijmen V, Modarres-Hashemi M (2010) Improved impossible differential cryptanalysis of 7-round AES-128. In: INDOCRYPT, vol 6498 of LNCS. Springer, pp 282–291
29. Cheon JH, Kim M, Kim K, Lee J-Y, Kang S (2001) Improved impossible differential cryptanalysis of Rijndael and Crypton. In: ICISC, vol 2288 of LNCS. Springer, pp 39–49

30. Bahrak B, Aref MR (2008) Impossible differential attack on seven-round AES-128. IET Inf Secur 2(2):28–32
31. Liu Y, Gu D, Liu Z, Li W (2012) Improved results on impossible differential cryptanalysis of reduced-round camellia-192/256. J Syst Softw 85(11):2451–2458
32. Yuan Z (2010) New impossible differential attacks on AES. IACR Cryptology ePrint Archive 2010:93
33. Ferguson N, Kelsey J, Lucks S, Schneier B, Stay M, Wagner DA, Whiting D (2000) Improved cryptanalysis of Rijndael. In: FSE, vol 1978 of LNCS. Springer, pp 213–230
34. Gilbert H, Minier M (2000) A collision attack on 7 rounds of Rijndael. In: AES candidate conference, pp 230–241
35. Demirci H, Selçuk AA (2008) A meet-in-the-middle attack on 8-round AES. In: FSE, vol 5086 of LNCS. Springer, pp 116–126
36. Demirci H, Taskin I, Çoban M, Baysal A (2009) Improved meet-in-the-middle attacks on AES. In: INDOCRYPT, vol 5922 of LNCS. Springer, pp 144–156
37. Biryukov A (2004) The boomerang attack on 5 and 6-round reduced AES. In: AES conference, vol 3373 of LNCS. Springer, pp 11–15
38. Bogdanov A, Khovratovich D, Rechberger C (2011) Biclique cryptanalysis of the full AES. In: ASIACRYPT, vol 7073 of LNCS. Springer, pp 344–371
39. Khovratovich D, Rechberger C, Savelieva A (2012) Bicliques for preimages: attacks on skein-512 and the SHA-2 family. In: FSE, vol 7549 of LNCS. Springer, pp 244–263
40. Guo J, Ling S, Rechberger C, Wang H (2010) Advanced meet-in-the-middle preimage attacks: first results on full tiger, and improved results on MD4 and SHA-2. In: ASIACRYPT, vol 6477 of LNCS. Springer, pp 56–75
41. Aoki K, Sasaki Y (2009) Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In: CRYPTO, vol 5677 of LNCS. Springer, pp 70–89
42. Hermelin M, Cho JY, Nyberg K (2008) Multidimensional linear cryptanalysis of reduced round serpent. In: ACISP, vol 5107 of LNCS. Springer, pp 203–215
43. Cho JY (2010) Linear cryptanalysis of reduced-round PRESENT. In: CT-RSA, vol 5985 of LNCS. Springer, pp 302–317
44. Collard B, Standaert F-X (2009) A statistical saturation attack against the block cipher PRESENT. In: CT-RSA, vol 5473 of LNCS. Springer, pp 195–210
45. Baignères T, Junod P, Vaudenay S (2004) How far can we go beyond linear cryptanalysis? In: ASIACRYPT, vol 3329 of LNCS. Springer, pp 432–450
46. Harpes C, Kramer GG, Massey JL (1995) A generalization of linear cryptanalysis and the applicability of matsui's piling-up lemma. In: EUROCRYPT, vol 921 of LNCS. Springer, pp 24–38
47. Harpes C, Massey JL (1997) Partitioning cryptanalysis. In: FSE, vol 1267 of LNCS. Springer, pp 13–27
48. Knudsen LR, Wagner DA (2002) Integral cryptanalysis. In: FSE, vol 2365 of LNCS. Springer, pp 112–127
49. Hwang K, Lee W, Lee S, Lee S, Lim J (2002) Saturation attacks on reduced round skipjack. In: FSE, vol 2365 of LNCS. Springer, pp 100–111
50. Miyaji A, Nonaka M (2002) Cryptanalysis of the reduced-round RC6. In: ICICS, vol 2513 of LNCS. Springer, pp 480–494
51. Ohkuma K (2009) Weak keys of reduced-round PRESENT for linear cryptanalysis. In: Selected areas in cryptography, vol 5867 of LNCS. Springer, pp 249–265
52. Biham E, Shamir A (1991) Differential cryptanalysis of des-like cryptosystems. J Cryptol 4(1):3–72
53. Wang M (2008) Differential cryptanalysis of reduced-round PRESENT. In: AFRICACRYPT, vol 5023 of LNCS. Springer, pp 40–49
54. Wang M, Sun Y, Tischhauser E, Preneel B (2012) A model for structure attacks, with applications to PRESENT and Serpent. In: FSE, vol 7549 of LNCS. Springer, pp 49–68
55. Jeong K, Kang H, Lee C, Sung J, Hong S (2012) Biclique cryptanalysis of lightweight block ciphers PRESENT, Piccolo and LED. IACR Cryptology ePrint Archive 2012:621
56. Abed F, Forler C, List E, Lucks S, Wenzel J (2012) Biclique cryptanalysis of the PRESENT and LED lightweight ciphers. IACR Cryptology ePrint Archive 2012:591
57. Tolba M, Abdelkhalek A, Youssef AM (2015) Meet-in-the-middle attacks on round-reduced khudra. In: SPACE, vol 9354 of LNCS. Springer, pp 127–138
58. Özen M, Çoban M, Karakoç F (2015) A guess-and-determine attack on reduced-round khudra and weak keys of full cipher. IACR Cryptology ePrint Archive 2015:1163
59. Dinur I (2014) Improved differential cryptanalysis of round-reduced speck. In: Selected areas in cryptography, vol 8781 of LNCS. Springer, pp 147–164
60. Abed F, List E, Lucks S, Wenzel J (2013) Cryptanalysis of the speck family of block ciphers. IACR Cryptology ePrint Archive 2013:568
61. AlKhzaimi H, Lauridsen MM (2013) Cryptanalysis of the SIMON family of block ciphers. IACR Cryptology ePrint Archive 2013:543
62. Kocher PC (1996) Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: CRYPTO, vol 1109 of LNCS. Springer, pp 104–113
63. Kocher PC, Jaffe J, Jun B, Rohatgi P (2011) Introduction to differential power analysis. J Cryptogr Eng 1(1):5–27
64. Shamir A (1979) How to share a secret. Commun ACM 22(11):612–613
65. Goodwill G, Jun B, Jaffe J, Rohatgi P (2011) A testing methodology for side-channel resistance validation. In: NIST non-invasive attack testing workshop
66. Guntur H, Ishii J, Satoh A (2014) Side-channel attack user reference architecture board SAKURA-g. In: 3rd IEEE global conference on consumer electronics (GCCE). IEEE, pp 271–274
67. Roy DB, Bhasin S, Patranabis S, Mukhopadhyay D (2017) Testing of side-channel leakage of cryptographic intellectual properties: metrics and evaluations. In: Hardware IP security and trust. Springer, pp 99–131
68. Shahverdi A, Taha M, Eisenbarth T (2015) Silent simon: a threshold implementation under 100 slices. In: HOST
69. Shahverdi A, Taha M, Eisenbarth T (2017) Lightweight side channel resistance: threshold implementations of simon. IEEE Trans Comput 66(4):661–671
70. Chen C, Inci MS, Taha M, Eisenbarth T (2016) Spectre: a tiny side-channel resistant speck core for FPGAs. In: CARDIS
71. Poschmann A, Moradi A, Khoo K, Lim C-W, Wang H, Ling S (2011) Side-channel resistant crypto for less than 2, 300 GE. J Cryptol 24(2):322–345
72. Canright D (2005) A very compact s-box for AES. In: CHES, vol 3659 of LNCS. Springer, pp 441–455
73. Mukhopadhyay D, Chakraborty RS (2014) Hardware security: design, threats, and safeguards. CRC Press
74. De Cnudde T, Reparaz O, Bilgin B, Nikova S, Nikov V, Rijmen V (2016) Masking AES with $d + 1$ shares in hardware. IACR Cryptology ePrint Archive 2016:631
75. Moradi A, Poschmann A, Ling S, Paar C, Wang H (2011) Pushing the limits: a very compact and a threshold implementation of AES. In: EUROCRYPT, vol 6632 of LNCS. Springer, pp 69–88
76. De Cnudde T, Nikova S (2017) Securing the present block cipher against combined side-channel analysis and fault attacks. IEEE Trans Very Large Scale Integr VLSI Syst PP(99):1–11