SpecTre: A Tiny Side-Channel Resistant Speck Core for FPGAs

Cong Chen, Mehmet Sinan Inci, Mostafa Taha*, and Thomas Eisenbarth

Worcester Polytechnic Institute, Worcester, MA 01609, USA
Email: {cchen3, msinci, teisenbarth}@wpi.edu
*Assiut University, Egypt
Email: mtaha@aun.edu.eg

Abstract. Emerging applications such as the Internet of Things require security solutions that are small and low cost, yet feature solid protection against a wide range of sophisticated attacks. Lightweight cryptographic schemes such as the Speck cipher that was recently proposed by the NSA aim to solve some of these challenges. However, before using Speck in any practical application, sound protection against side-channel attacks must be in place. In this work, we propose a bit-serialized implementation of Speck, to achieve minimal area footprint. We further propose a Speck core that is provably secure against first-order side-channel attacks using a threshold implementation technique which depends on secure multiparty computation. The resulting design is a tiny crypto core that provides AES-like security in under 45 slices on a low-cost Xilinx Spartan 3 FPGA. The first-order side-channel resistant version of the same core needs less than 100 slices. The security of the protected core is validated by state-of-the-art side-channel leakage detection tests.

1 Introduction

Lightweight cryptography aims to answer the need for smaller, less energy consuming security tokens as commonly used for authentication and micropayments. Lightweight cryptographic implementations are commonly used in hardware modules in RFID tokens, remotes, and all types of devices for the Internet of Things (IoT). In these application scenarios the available area footprint as well as the computation power and the battery life are heavily constrained, where the commonly used and trusted standard ciphers like the AES are too costly. Here comes the arena of Speck as a lightweight block cipher. For more information on lightweight cryptography, please refer to [1].

Another major concern for embedded security solutions—besides cost—is side-channel attacks. Given potential physical access to the device, an attacker can collect and exploit various emanations from the electrical circuits, like electromagnetic waves, power consumption, sound or execution timings to recover secret information [2]. Over the last decade, a vast body of work has been performed to find effective methods to prevent these powerful attacks, especially the *differential power analysis* (DPA) attack. Usually, the implementation is hardened by adding *masking* or *hiding* countermeasures [3]. One of the promising and fairly generic masking techniques is Threshold Implementation [4]. While fairly expensive to apply on standard ciphers like AES [5, 6], its application to lightweight ciphers (e.g. Simon and KATAN) comes at reasonable overheads [7, 8]. More importantly, prior work verified its postulated resistance to first-order side-channel attacks [5, 6, 8, 7], making the introduced area overhead worthwhile.

Speck and its sister Simon are two lightweight block ciphers proposed by NSA as versatile alternatives to the AES [9]. Speck was optimized for software applications, while Simon was optimized for hardware applications. Speck supports various key sizes (64, 72, 96, 128, 144, 192 and 256 bits) and block sizes (32, 48, 64, 96 and 128 bits), making it suitable candidate for a broad range of applications. The design of Speck depends entirely on modular addition, rotation and XOR in a number of rounds ranging from 22, to 34 (depending on key and block sizes). Although Simon showed a small footprint on FPGAs (only 36 slices) [10], its throughout was not promising (3.6 Mbps). Moreover, in most IoT applications, the infrastructure is mixed where some nodes are equipped with low-power microcontrollers, others with dedicated hardware or even FPGAs. If one block cipher is to be adopted in both platforms, Simon will cause a huge hit in performance. On the other hand, Speck was optimized for software and shows good throughput that is comparable to AES (while being more than double the throughput of Simon) on both low-end and high-end processors [11, 9]. However, being optimized for software, Speck was never tested on FPGA platforms, where comes our contribution.

In this paper, we propose a bit-serialized implementation of Speck with the aim to minimize area footprint, making it an ideal candidate for low-cost embedded applications. Next, we show how the design can be converted into a Threshold implementation to address the need for a side-channel protected cryptographic core. Finally, we apply the state-of-the-art leakage detection method, namely the TVLA test introduced in [12], to practically verify the claimed first-order resistance of our design. To that end, our contribution is two-fold:

- We present a bit-serialized implementation of of the lightweight block cipher Speck. This implementation style yields a highly area-efficient hardware implementation on FPGAs.
- We further show that the bit-serialized implementation of Speck can be protected against first-order side-channel analysis using the Threshold Implementation technique. The design has been thoroughly tested using state-of-the-art leakage detection methods, yet has a smaller area footprint than many *unprotected* symmetric ciphers. As such, the bit serialized Threshold Implementation of Speck (named SpecTre) provides a more secure crypto core for embedded scenarios while still maintaining the goal of a low-cost implementation.

It is surprising that a FPGA implementation of Speck, which was optimized for software, requires only slightly more area than Simon which was optimized for hardware. Meanwhile, SpecTre achieves more than double the throughput

Table 1. Speck parameters for various block and key sizes

Block Size	Key Size	Word Size	Rounds
32	64	16	22
48	72	24	22
48	96	24	23
64	96	32	26
64	128	32	27
96	96	48	28
96	144	48	29
128	128	64	32
128	192	64	33
128	256	64	34

of Simon. Our Threshold Implementation design occupies only 99 slices on an entry level FPGA, i.e. the Xilinx Spartan 3, while generating output at 9.68 Mbps, making it suitable for lightweight applications like wireless sensor networks, RFIDs, etc. We also prove the security of our design against first order side-channel attacks by performing leakage detection tests on the SpecTre core.

The paper is organized as follows. Section 2 provides background information on Speck block cipher, masking countermeasure and Threshold Implementations. Section 3 details the bit-serialized implementation of Speck on FPGAs. Section 4 shows how the Threshold Implementation of Speck is designed and implemented. Section 5 compares the implementation results against other ciphers. Section 6 discusses the experimental setup and the leakage detection test along with the result of evaluating the protected core. The paper is concluded in Section 7.

2 Background

We introduce the Speck cipher and give some background on side-channel analysis and how they can be mitigated.

2.1 Speck Cipher:

Speck is a family of lightweight block ciphers publicly released by the NSA in June 2013 [9]. Speck has been optimized for performance in software implementations. Like other common block ciphers, Speck supports a range of key and block size options ranging from 64 bits to 256 bits and 32 bits to 128 bits, respectively. Speck is commonly notated by the block size ($2n$) and key size (mn), where n is the word size. For example, Speck32/64 has a word size of 16 bits and works with input block size of 32 bits and key size of 64 bits. Depending on both the key and the block size, the number of rounds range from 22 rounds to

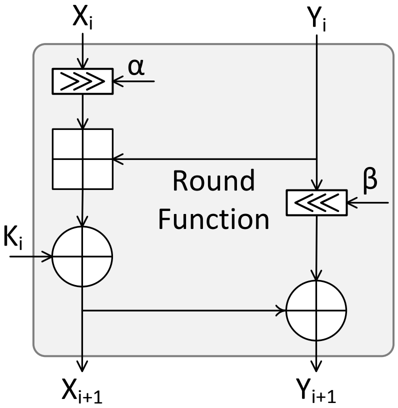


Fig. 1. Block diagram of the Speck cipher showing the round function

34 rounds. A detailed list of block and key size options and the resulting number of rounds are given in Table 1.

As shown in Figure 1, the input is split into two words, each of size n. Each round of Speck consists of bitwise XOR, an addition modulo 2^n, and left and right circular shift operations. These operations enable high throughput and efficient implementation on most microprocessors. As we will later show, the design also lends itself to efficient implementation in hardware. The round function can be represented as:

$$\begin{aligned} x_{i+1} =&(S^{-\alpha}(x_i) + y_i) \oplus k_i \\ y_{i+1} =&S^{\beta}(y_i) \oplus x_{i+1} \end{aligned} \tag{1}$$

where S^j is left circular shift by j bits. Note that $\alpha = 8$ and $\beta = 3$, except for the Speck32/64 toy cipher, where $(\alpha, \beta) = (7, 2)$. Notably, the exact same round function is also used for the key scheduling. The key schedule of Speck takes the key as input and outputs a round key for each round of the encryption by applying the round function. In the key schedule, differently than the encryption, the left word is XOR'ed with a constant representing the round number starting from 0.

According to [9], Speck with block and key sizes of 128-bits requires 1396 GE (Gate Equivalent) and produces 12.1 Kbps throughput compared to 2400 GE and 56.6 Kbps throughput of the AES 128. In most applications, this smaller circuit size translates into lower power consumption and more portability at lower cost. Note that even though the throughput/area ratio of the Speck is lower than the AES for 128-bits, it still has the advantage of being lightweight in addition to supporting smaller block sizes.

In [11], the authors of the Simon and Speck implemented both ciphers on a commercially available, 8-bit AVR microcontroller and compared the performance results to other block ciphers. The results show that the both ciphers perform well on the 8-bit test platform. Also, the Speck cipher has the best overall performance among all tested block ciphers *in software*.

2.2 Differential Power Analysis and Masking Countermeasure

Differential Power Analysis (DPA) is an implementation attack that targets the underlying implementation of a crypto algorithm rather than its mathematical structure. DPA exploits the fact that learning even minimal information about intermediate states of a cryptographic algorithm can result in key recovery an hence a full break of the cryptosystem. In this attack, the adversary measures the power consumption (or electromagnetic emanation) of the targeted platform while it performs cryptographic operations. Based on a guess on a small part of the secret key (subkey), the adversary can compute the hypothetical values of an intermediate cipher state, and thus predict changes in the power consumption. Finally, the predicted changes in power consumption are compared against the measured power traces, where the correct subkey that results in the best match. A detailed introduction to DPA is given in [2]. These attacks have been widely studied, apply to virtually any implementation of cryptography, and are very difficult to prevent [3].

One of the popular methods to thwart this attack is masking. Masking depends on using a fresh random variable to blind all the intermediate variables, hence prevents the ability to estimate correct hypothetical traces. Masking is typically achieved by splitting the input data (plaintext and/or key) into d shares using a random variable. This countermeasure is notated as $(d-1)$-order masking. Each share is processed independently to produce an output. The outputs are then combined to retrieve the original output (ciphertext). The effect of linear functions within a crypto algorithm can easily be re-expressed in terms of the input shares. However, re-expressing non-linear functions is typically a challenging task and every crypto algorithm needs a special solution.

The adversary can still break a $(d-1)$-order masked implementation if he can combine the information from all the d shares. If the d shares leak at the same point in time, the attack is called higher-order DPA. However, if the shares leak at different points in time, the attack is called multivariate DPA.

2.3 Threshold Implementation

Threshold Implementation (TI) is a popular method of applying masking countermeasure that is provably secure against first order side-channel attacks. TI applies the concepts of XOR-secret sharing based multiparty computation with some basic requirements [4]. In addition to a straightforward XOR secret sharing, TI requires that any sub functions operating on the shares to be *correct*, *non-complete* and *uniform*. That is: the combination of the output shares must always return the correct result, inputs to each sub-function must always exclude at least one share, and the output shares must be uniformly distributed if the input shares were uniform . Non-completeness enforces that the secret state is not processed by any function within the embedded module (at least one share will be always missing). Uniformity enforces that all the intermediate variables within the new masked module have the same entropy of the original secret state. This last requirement is typically the most difficult to achieve. However, it can be achieved by introducing fresh randomness into the output state.

There are quite a few designs in the literature with Threshold Implementation countermeasure, including TI-AES [6], TI-Keccak [13] and TI-Simon [7]. Moreover, higher-order Threshold Implementations, where each sub-function excludes at least $n \geq 2$ shares, can also be realized [8].

3 A Bit-Serialized Speck Hardware Core

One goal of lightweight cryptography in hardware is to minimize the area footprint of a block cipher implementation. However, in most scenarios, the block size and key size enforce a lower bound for a given configuration, since each key and state bit needs to be stored. One exception is Ktantan [14], where the authors out-sourced the key storage to reduce the area footprint even further in case where the key is fixed. The authors of [15] showed that the combinational parts (i.e. the cryptographically critical part of the cipher) of their most serialized (and hence smallest) implementation of Present consumed only about 5% of the area. In general, ciphers that are fully bit-serializable can achieve the best area footprint for a given state and key size. In [10], a-bit-serialized design of the Simon block cipher was proposed to achieve a compact implementation. Besides being fully bit-serialized, the implementation stores key and cipher state in shift registers which can be efficiently implemented in FPGA slices.

Following the same concept for the Speck cipher, the arithmetic addition between two n-bit words can be implemented as n serialized one-bit full additions between the two corresponding-bits and the carry-bit from prior-additions. This approach sacrifices the performance while achieving small design size since only one-bit full adder is used instead of an n-bit adder.

Figure 2 shows the structure of the bit-serialized round function for the unprotected Speck128/128 (128-bit block and 128-bit key size). The structure runs for 32 rounds, where each round requires 64 clock cycles (a total of 2048 cycles). The 128-bit input plaintext is stored into two separate 64-bit shift registers. The left shift register represents x in Eq. 1, while the right register represents y.

The left register (x) is split into two parts to expose bit number α (=8 in Speck128/128), which eliminates the need for a dedicated cyclic shift operation. Instead, we directly feed the α-*th*-bit into the feedback function. Here, the feedback function of the left register accepts one bit from register x (starting from bit number α), one bit from register y, one key bit to perform a 1-bit full adder and an XOR. During the first α clock cycles of each round, the feedback function feeds the least significant part of the register. Meanwhile, the old values of this part are sequentially feed into the most-significant part waiting for their turn to

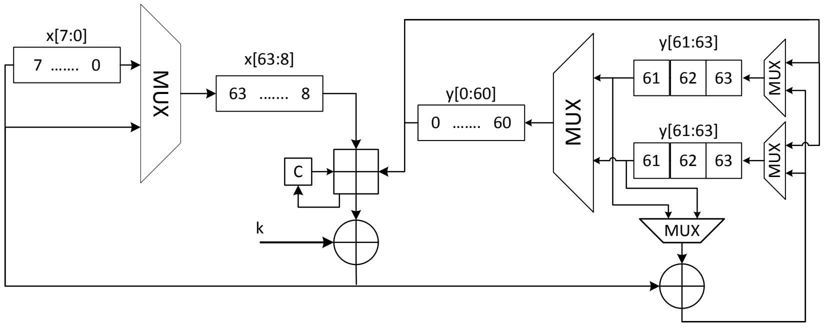


Fig. 2. Structure of one round of-bit-serialized Speck

activate the feedback function. After α clock cycles, the feedback function feeds the most significant part directly preparing the state register for the next round (putting bit number α in the lead).

While processing register x, we did not use any extra storage. This was possible because each bit of the x register is used only once. However, each bit of the y register is used two times, one in the left feedback function (that of register x) and one in the right feedback function (that of register y). Hence, we had to duplicate the β (=3 in Speck128/128) most significant bits of the register to hold the new and old values. Also, by exposing bit number $(64 - \beta = 61)$ and using it directly in the feedback function, we eliminated the need for a dedicated cyclic shift operation. Here, the feedback function of right register accepts the output of the left feedback function (that of register x) and one bit from register y (starting from bit number $(64 - \beta)$) to perform a single XOR. The output of this feedback function is feed to the most significant bit of register y. Note that one copy of β duplicated bits is used to hold the old values of register y, while the other copy holds the new values. This role is revered in the beginning of each round.

The key schedule applies the same round function and is performed in parallel to the encryption round function. Note that, since the round function is highly area optimized, this does not introduce any significant area overhead.

4 Threshold Implementation of Speck

In order to minimize the cost of design while fulfilling the three properties of Threshold Implementation, we chose to split the secrets, both the key and the plaintext blocks, into three shares. In the following, we show how *Correctness*, *Non-completeness* and *Uniformity* are achieved using three shares.

In 3-share Speck implementation, the key k and the plaintext p are split into three shares. Two of the shares are chosen uniformly at random while the third share is the XOR-sum of k (or p) with the two random shares

$$\begin{aligned} p_1 \overset{\$}{\leftarrow} \{0,1\}^n ; p_2 \overset{\$}{\leftarrow} \{0,1\}^n ; p_3 = p_1 \oplus p_2 \oplus p \\ k_1 \overset{\$}{\leftarrow} \{0,1\}^n ; k_2 \overset{\$}{\leftarrow} \{0,1\}^n ; k_3 = k_1 \oplus k_2 \oplus k \end{aligned} \tag{2}$$

where ($\overset{\$}{\leftarrow}$) denotes selecting at random from the given set. This yields a uniform and correct XOR-shared representation of the state p and key state k:

$$\begin{aligned} p = p_1 \oplus p_2 \oplus p_3 \\ k = k_1 \oplus k_2 \oplus k_3 \end{aligned} \tag{3}$$

In each round operation, both *cyclic rotation* and $\oplus$ are linear operations which can be performed on each share separately. The only nonlinear operation is the full addition between two n-bit words. We used the following equations to implement a valid TI 3-share addition. Moreover, we used-bit-serialized addition as shown in [16] where 1-bit addition is performed in one clock cycle. Hence a n-bit full addition will only cost a single addition circuit and n clock cycles. Suppose, at the *i-th* clock cycle, the *i-th*-bit addition, $i \in \{0, 1, ...n-1\}$, is performed between two plaintext-bits a and b and one carry bit c as follows.

$$\begin{aligned} a_i =& a_{i,1} \oplus a_{i,2} \oplus a_{i,3} \\ b_i =& b_{i,1} \oplus b_{i,2} \oplus b_{i,3} \\ c_i =& c_{i,1} \oplus c_{i,2} \oplus c_{i,3} \end{aligned} \tag{4}$$

Where a_i and b_i are *i-th*-bit of the two words, c_i is the *i-th* carry-bit and subscript $j \in \{1, 2, 3\}$ indicates the *j-th* share of each-bit. Then the three shares of the sum-bit s_i and the carry-bit c_{i+1} can be written as follows:

$$\begin{aligned} s_{i,1} =& a_{i,1} \oplus b_{i,1} \oplus c_{i,1} \\ s_{i,2} =& a_{i,2} \oplus b_{i,2} \oplus c_{i,2} \\ s_{i,3} =& a_{i,3} \oplus b_{i,3} \oplus c_{i,3} \end{aligned} \tag{5}$$

$$\begin{aligned} c_{i+1,1} =& (a_{i,2} \cdot b_{i,2}) \oplus (a_{i,2} \cdot b_{i,3}) \oplus (a_{i,3} \cdot b_{i,2}) \oplus \\ & (a_{i,2} \cdot c_{i,2}) \oplus (a_{i,2} \cdot c_{i,3}) \oplus (a_{i,3} \cdot c_{i,2}) \oplus \\ & (b_{i,2} \cdot c_{i,2}) \oplus (b_{i,2} \cdot c_{i,3}) \oplus (b_{i,3} \cdot c_{i,2}) \\ c_{i+1,2} =& (a_{i,3} \cdot b_{i,3}) \oplus (a_{i,3} \cdot b_{i,1}) \oplus (a_{i,1} \cdot b_{i,3}) \oplus \\ & (a_{i,3} \cdot c_{i,3}) \oplus (a_{i,3} \cdot c_{i,1}) \oplus (a_{i,1} \cdot c_{i,3}) \oplus \\ & (b_{i,3} \cdot c_{i,3}) \oplus (b_{i,3} \cdot c_{i,1}) \oplus (b_{i,1} \cdot c_{i,3}) \\ c_{i+1,3} =& (a_{i,1} \cdot b_{i,1}) \oplus (a_{i,1} \cdot b_{i,2}) \oplus (a_{i,2} \cdot b_{i,1}) \oplus \\ & (a_{i,1} \cdot c_{i,1}) \oplus (a_{i,1} \cdot c_{i,2}) \oplus (a_{i,2} \cdot c_{i,1}) \oplus \\ & (b_{i,1} \cdot c_{i,1}) \oplus (b_{i,1} \cdot c_{i,2}) \oplus (b_{i,2} \cdot c_{i,1}) \end{aligned} \tag{6}$$

Table 2. Resource usage and Performance of Unprotected and Protected TI-Speck. The slices in paretheses are used as shift registers.

Cipher	**Regs**	**LUTs (ShiftRegs)**	**Slices**	**Speed [MHz]**
Speck32/64	29	58(9)	36	166
TI-Speck32/64	70	137(27)	79	123
Speck128/128	31	76(22)	43	161
TI-Speck128/128	70	181(66)	99	155

Correctness requirement of the threshold cryptography holds true and can be verified using following equations.

$$\begin{aligned} s_i &= s_{i,1} \oplus s_{i,2} \oplus s_{i,3} \\ c_{i+1} &= c_{i+1,1} \oplus c_{i+1,2} \oplus c_{i+1,3} \end{aligned} \tag{7}$$

Also, it can be easily seen that each output share is independent of at least one share of each input, hence satisfying the *non-completeness* requirement. As pointed in [16], (s_i, c_{i+1}) pair is *uniformly distributed* and hence the three properties of Threshold Implementation are achieved.

The actual TI implementation of the above equations can be constructed using three copies of the bit-serialized Speck shown in Figure 2 with a slight modification. Note that the linear operations involve only one share of any input such as rotation, computing the sum bit and exclusive OR. However, computation of each share of the carry bit requires two shares of inputs a_i, b_i, c_i. Hence, the regular arithmetic adder which takes three input bits a_i, b_i, c_i in Figure2 is replaced with a TI adder according to equations (5) and (6).

5 FPGA Implementation Results

The unprotected and the TI versions of Speck as introduced in Sections 3 and 4 are realized using Verilog and implemented on a Xilinx Spartan 3 FPGA. Detailed results are shown in Table 2. We chose the outdated Spartan 3 platform to enable better comparability to related lightweight cipher designs. This comparison is given in Table 3. For the experiments, we implemented two versions of Speck cipher: Speck32/64 and Speck128/128. These two versions have 22 and 32 encryption rounds respectively. The designs are synthesized using Xilinx ISE 14.7. The unprotected implementation of Speck32/64 requires 29 Flip-Flops(FFs), 58 LUTs of which 9 are used as Shifter Registers (SRs), and totally occupies 36 slices. In terms of speed, the maximum frequency of the design is 166 MHz. Its protected version requires 70 FFs, 137 LUTs and 79 slices, and it can run at 123 MHz.

The unprotected Speck128/128 version requires 31 FFs, occupying 43 slices and runs at 161 MHz. TI-Speck128/128 requires 70 FFs, occupying 99 slices and can run at 155 MHz resulting in 9.68 Mbps throughput. Note that unlike some

Table 3. Comparison of area requirements and throughput on FPGAs of various block and stream ciphers.

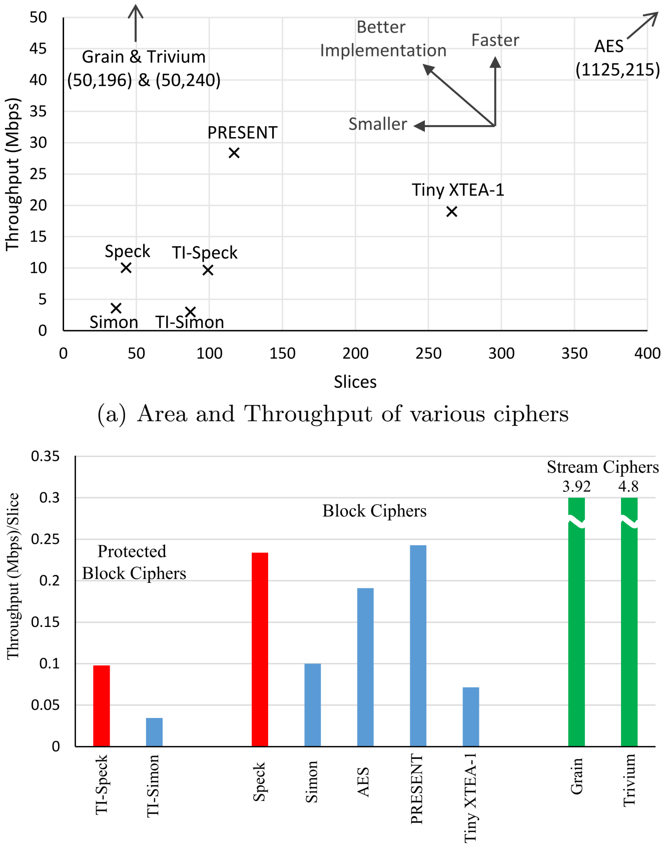
Cipher	**Slices**	**Throughput [Mbps]**	**Platform**
Threshold Implementation;			
TI-Speck128/128	**99**	**9.68**	**xc3s50**
TI-Simon128/128 [7]	87	3.0	xc3s50
Unprotected Block Ciphers;			
Speck128/128	**43**	**10.05**	**xc3s50**
Simon128/128 [10]	36	3.6	xc3s50
AES [17]	1125	215	xcv1000e
PRESENT [18]	117	28.4	xc3s50-5
Tiny XTEA-1 [19]	266	19	xc3s50-5
Stream Ciphers;			
GRAIN 128 [20]	50	196	xc3s50-5
TRIVIUM [20]	50	240	xc3s50-5

other block cipher implementations on FPGAs, we did not use any block RAMs or any other type of storage.

After obtaining the implementation results, we compared our Speck implementations to other lightweight block cipher implementations. The most comparable implementations to Speck and TI-Speck are the Simon variants. As expected, Simon was optimized for hardware and can be smaller than Speck. However, Speck achieves more than double the throughput of Simon for a minor increase in the area (7 slices for the unprotected core, and 12 slices for the protected one). Due to the lower number of rounds, Speck also has a much lower delay. The bit-serialized implementations of both ciphers can be protected against side-channel attacks using the TI countermeasure at a reasonable overhead. In terms of comparison to other TI implementations, while there are some previous publications [7, 21, 22] and all of them have been applied to FPGAs for side-channel evaluation, only the Simon paper [7] reported synthesis results for FPGAs, making this the only design we can compare to.

When compared to other block ciphers, especially AES, but also lightweight versions such as Present, it is remarkable that Speck is significantly smaller. In fact, the side-channel protected implementation of Speck128/128 is about the same size as a Present64/80 core without side-channel protection. Hence, Speck (and Simon as well) make great choices for a wide range of embedded security solutions.

When considering size and throughput, stream ciphers such as Grain and Trivium also achieve remarkable results on FPGAs. As can be seen in Table 3, both ciphers need slightly larger area, 50 slices instead of 43 for Speck128/128 while giving a throughput increase of a factor of 22 and 27 respectively. Although this is correct asymptotically, stream ciphers like Trivium suffer from a warm-up



(a) Area and Throughput of various ciphers

(b) Throughput per Slice

Fig. 3. Area and performance results of various ciphers.

phase. The design must be clocked for up to 1124 cycles before the first bit can be encrypted, resulting in high delays. This makes stream ciphers unattractive in most IoT scenarios where payloads are small and infrequent.

6 Leakage Analysis

In this section, we show the results of leakage detection tests as applied to the protected TI-Speck32/64 implementation following the test suite proposed in [12]. Unlike traditional side-channel attacks, leakage detection tests are statistical tests that are designed to measure the influence of secret intermediate variables on side-channel traces. In other words, we are not interested in full recovering of the secret key but only validating first order side-channel resistance. To this goal, we show that all the intermediate variables have no significant influence on the power consumption.

6.1 Experimental Setup

We use the Side-channel Attack Standard Evaluation Board G-II (SASEBO) that is designed specifically to measure side-channel leakage of FPGA hardware designs. The board contains a Virtex-5 XC5VLX30 FPGA used for cryptographic implementation and a Spartan-3A XC3S400A FPGA for control. The analyzed design is the TI-Speck32/64 since it has the lowest number of rounds, yet all relevant components to ensure a meaningful analysis. After re-synthesizing the design for Virtex-5 and loading it into the board, the board is controlled via PC connection.

We use Tektronix DPO 5104 to measure the power consumption of the cryptographic engine with high precision and sampling rate of 1 Giga-samples per second. Then, we perform peak extractions on the original power traces in which only the peak value at each clock cycle is picked out so that we have only a small amount of power samples which can be processed efficiently. Speck32/64 requires 22x16 clock cycles to run the encryption and another 18 clock cycles for data input and output. The total number of required cycles per encryption is 370. Hence, in order to ensure that all the leakage points are extracted, we recorded the leakage in 370 clock cycles.

6.2 Leakage Detection Tests

In order to evaluate the side-channel leakage of both the original and the TI-Speck designs, we used a test suite [12] that is commonly used to detect leakage of systems [23, 8, 7]. The test suite checks the leakage data under two scenarios;

- Fixed versus Random (FvR)
- Random versus Random (RvR)

The first scenario, FvR, takes two sets of leakage traces as input: one set of traces collected with a fixed plaintext while the other set is collected with random plaintexts. Note that the same key is used for all encryptions. The two sets are collected in an interleaved fashion using a uniform binary random variable. Then, for each set, the sample mean (μ) and the sample standard deviation (σ) are calculated. Then, the Welchs t-test is performed to see if t exceeds pre-defined threshold in order to determine whether to fail the device or not. We use the same commonly used threshold of 4.5 [12]. The formula for the Welchs t-test is as follows with a and b denoting the two data sets and N_i the number of traces in set $i \in \{a, b\}$.

$$t = \frac{\mu_a - \mu_b}{\sqrt{(\sigma_a^2/N_a) + (\sigma_b^2/N_b))}} \tag{8}$$

In simple terms, the FvR test validates the hypothesis that the processing of any secret data is statistically indistinguishable from the processing of random data. This test is generic and assumes no particular knowledge about the running algorithm or the underlying module. Hence, it examines the leakage of all the intermediate variables along the algorithm.

Table 4. Side-Channel Leakages in different scenarios

Leakage Scenario	**Maximum t value**	**Minimum t value**	**Standard deviation**
FvR Unprotected	63.16	-68.81	20.74
RvR Unprotected	9.78	-9.72	2.13
FvR Protected	2.28	-4.44	1
RvR Protected	2.31	-1.97	0.82

The second scenario (RvR), uses the same analysis method but all the traces are collected with random plaintexts and the same key. In this scenario, traces are divided according to a chosen binary intermediate variable. This test resembles a profiled attack using the original 1-bit DPA of Kocher [2]. The test validates the hypothesis that knowledge of any internal variable does not help in identifying leakage traces. The RvR test assumes knowledge of the key and the target algorithm (in order to compute the intermediate variable). Although the test does not assume any power consumption model, knowledge of the underlying implementation determines if the leakage should depend on the current state only, or on the transition between two states. In our experiment, we selected the intermediate variable as the least significant bit of the x register after the first round.

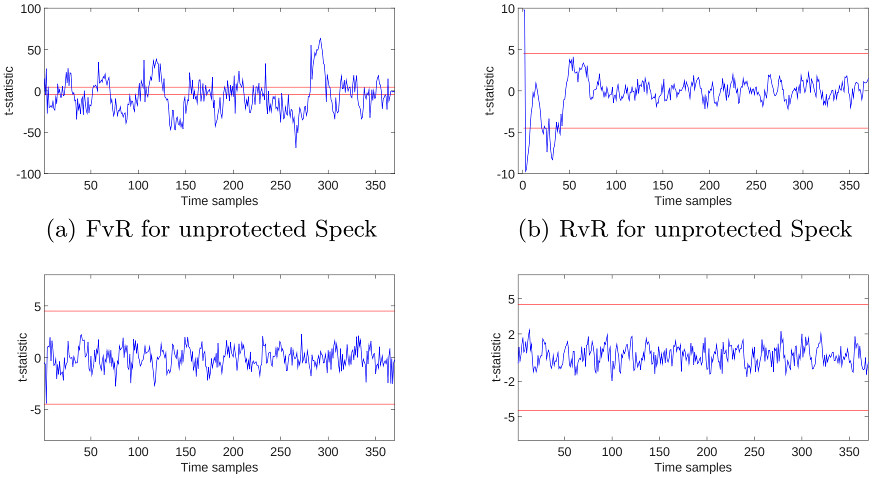
We would like to note that the FvR and RvR tests are stronger and far more sensitive to side-channel leakage than SPA and DPA attacks. Their ability to distinguish points in a leakage trace may not even result in partial or full recovery. In these types of attacks, the adversary must be able to distinguish a key dependent internal state from noise to actually carry out an attack and recover partial or full information about the secret. Having said that, SPA and DPA can be used to test the side-channel resistance of the proposed TI-Speck and can be conducted as future research.

6.3 Results

In order to fairly compare between the two proposed designs, we used the TI-Speck to realize both the protected and unprotected implementations. For the unprotected implementation, we set the value of one share to the input secret (plaintext or key) and the value of the other shares to zero. For the protected implementation, all the three shares are set randomly. This way, we can use the exact same script for collecting and analyzing traces.

The unprotected core is expected to leak and serves as a reference to show that the leakage detection actually works properly in the given setup. As for the protected core, the masks are properly randomized and no leakage should be detected even for a high number of observations.

As noted in Table 4, the t values clearly prove that the Threshold Implementation of Speck is resistant to side-channel analysis according to the pass/-fail criteria. More specifically, Figure 4(a) and Figure 4(c) shows the FvR tests



(a) FvR for unprotected Speck

(b) RvR for unprotected Speck

(c) FvR for protected Speck

(d) RvR for protected Speck

Fig. 4. Side-Channel Leakage Results. The Leakage detection clearly indicates leakage for the implementations where masking is turned off, and indicates absence of leakage in the other cases.

for unprotected and protected implementation respectively. For unprotected implementation, t values at most time moments exceed the predefined threshold indicating that the leakage caused by fixed input can be easily distinguished from the one of random inputs and the leakage is dependent on the sensitive intermediate values in the unprotected core. In contrast, the lesser t values in the protected implementation show the first order leakage caused by fixed inputs and random inputs cannot be distinguished and exploited for a key recovery, and the side-channel resistance of the protected TI core is validated.

The RvR tests lead to the same conclusion. Note that we use the LSB of the left part (bit 0 in register x) after the first round operation to partition the power traces. If the LSB is 0, all the corresponding traces are put in set 0 and otherwise in set 1. Figure 4(b) shows that t values during the first two rounds are beyond the threshold. This is because some intermediates are related to chosen LSB but after two rounds of operation the dependency disappear. In other words, the leakages near the operation of the chosen intermediate values depend on them and demonstrate the vulnerability of the unprotected implementation. In contrast, the less t in the protected core implies little dependency between the intermediate values and the leakage, and the effectiveness of the TI countermeasure is again validated.

7 Conclusion

In conclusion, we designed a bit-serialized version of Speck and implemented it on FPGA. The bit-serialized Speck core is only slightly larger that a bit-serialized

Simon core—the current record-holder in smallest crypto core on FPGAs—but improves throughput and reduces latency by a factor of 2.4 times.

We also proposed a novel Threshold Implementation of the Speck cipher with three shares. We analyzed the power consumption of SpecTre and verified that the three-share TI-Speck implementation is resistant to first order side-channel attacks. Area-wise, our SpecTre design did not create a significant overhead and stayed true to the cipher's essence. Also, speed-wise we observed only marginal slow down compared to the unprotected version.

References

1. T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Design & Test of Computers*, no. 6, pp. 522–533, 2007.
2. P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, 2011. [Online]. Available: http://dx.doi.org/10.1007/s13389-011-0006-y
3. S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smartcards.* US: Springer, 2007.
4. S. Nikova, C. Rechberger, and V. Rijmen, "Threshold Implementations Against Side-Channel Attacks and Glitches," in *Information and Communications Security*, ser. Springer LNCS, P. Ning, S. Qing, and N. Li, Eds., 2006, vol. 4307, pp. 529–545.
5. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, "Pushing the Limits: A Very Compact and a Threshold Implementation of AES," in *Advances in Cryptology — EUROCRYPT 2011*, ser. Springer LNCS, K. G. Paterson, Ed., 2011, vol. 6632, pp. 69–88.
6. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "A More Efficient AES Threshold Implementation," in *Progress in Cryptology –AFRICACRYPT 2014*, ser. Springer LNCS, D. Pointcheval and D. Vergnaud, Eds., 2014, vol. 8469, pp. 267–284.
7. A. Shahverdi, M. Taha, and T. Eisenbarth, "Silent simon: A threshold implementation under 100 slices," *IACR Cryptology ePrint Archive*, vol. 2015, p. 172, 2015.
8. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "Higher-order threshold implementations," in *Advances in Cryptology–ASIACRYPT 2014.* Springer, 2014, pp. 326–343.
9. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The simon and speck families of lightweight block ciphers." *IACR Cryptology ePrint Archive*, vol. 2013, p. 404, 2013.
10. A. Aysu, E. Gulcan, and P. Schaumont, "Simon says: Break area records of block ciphers on fpgas," *Embedded Systems Letters, IEEE*, vol. 6, no. 2, pp. 37–40, 2014.
11. R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The simon and speck block ciphers on avr 8-bit microcontrollers," Cryptology ePrint Archive, Report 2014/947, 2014, http://eprint.iacr.org/.
12. B. J. Gilbert Goodwill, J. Jaffe, P. Rohatgi *et al.*, "A testing methodology for side-channel resistance validation," in *NIST Non-invasive attack testing workshop*, 2011.
13. B. Bilgin, J. Daemen, V. Nikov, S. Nikova, V. Rijmen, and G. Van Assche, "Efficient and First-Order DPA Resistant Implementations of Keccak," in *Smart Card*

Research and Advanced Applications, ser. Springer LNCS, A. Francillon and P. Rohatgi, Eds., 2014, pp. 187–199.

14. C. De Cannière, O. Dunkelman, and M. Knečević, "KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers," in *Cryptographic Hardware and Embedded Systems — CHES 2009*, ser. Lecture Notes in Computer Science, C. Clavier and K. Gaj, Eds. Springer Berlin Heidelberg, 2009, vol. 5747, pp. 272–288. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04138-9_20
15. C. Rolfes, A. Poschmann, G. Leander, and C. Paar, "Ultra-lightweight implementations for smart devices security for 1000 gate equivalents," in *Smart Card Research and Advanced Applications*, ser. Lecture Notes in Computer Science, G. Grimaud and F.-X. Standaert, Eds. Springer Berlin Heidelberg, 2008, vol. 5189, pp. 89–103. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85893-5_7
16. T. Schneider, A. Moradi, and T. Güneysu, "Arithmetic addition over boolean masking - towards first- and second-order resistance in hardware," Cryptology ePrint Archive, Report 2015/066, 2015, http://eprint.iacr.org/.
17. N. Pramstaller and J. Wolkerstorfer, "A universal and efficient aes co-processor for field programmable logic arrays," in *Field Programmable Logic and Application.* Springer, 2004, pp. 565–574.
18. P. Yalla and J.-P. Kaps, "Lightweight cryptography for fpgas," in *Reconfigurable Computing and FPGAs, 2009. ReConFig'09. International Conference on.* IEEE, 2009, pp. 225–230.
19. J.-P. Kaps, "Chai-tea, cryptographic hardware implementations of xtea," in *Progress in Cryptology-INDOCRYPT 2008.* Springer, 2008, pp. 363–375.
20. D. Hwang, M. Chaney, S. Karanam, N. Ton, and K. Gaj, "Comparison of fpga-targeted hardware implementations of estream stream cipher candidates," *The State of the Art of Stream Ciphers*, pp. 151–162, 2008.
21. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen, "A more efficient aes threshold implementation," in *Progress in Cryptology–AFRICACRYPT 2014.* Springer, 2014, pp. 267–284.
22. B. Bilgin, J. Daemen, V. Nikov, S. Nikova, V. Rijmen, and G. Van Assche, "Efficient and first-order dpa resistant implementations of keccak," in *Smart Card Research and Advanced Applications.* Springer, 2014, pp. 187–199.
23. A. J. Leiserson, M. E. Marson, and M. A. Wachs, "Gate-level masking under a path-based leakage metric," in *Cryptographic Hardware and Embedded Systems–CHES 2014.* Springer, 2014, pp. 580–597.