DPA on hardware implementations of Ascon and Keyak

Niels Samwel[1] and Joan Daemen[1,2]

[1] Digital Security Group, Radboud University Nijmegen
{n.samwel, joan}@cs.ru.nl
[2] ST Microelectronics

Abstract. This work applies side channel analysis on hardware implementations of two CAESAR candidates, Keyak and Ascon. Both algorithms are cryptographic sponges with an iterated permutation. The algorithms share an s-box so attacks on the non-linear step of the permutation are similar. This work presents the first results of a DPA attack on Keyak using traces generated by an FPGA. A new attack is crafted for a larger sensitive variable to reduce the number of traces. It also presents and applies the first CPA attack on Ascon. Using a toy-sized threshold implementation of Ascon we try to give insight in the order of the steps of a permutation.

Keywords: Differential power analysis, Keccak, Keyak, Ascon, Hardware implementations

1 Introduction

Side-channel analysis on cryptographic implementations have been known for quite a while [1]. Implementations can leak through many different channels, one of them is the amount of power that is consumed during cryptographic operations performed on a device. A technique to exploit this leakage is differential power analysis (DPA). DPA uses the difference of means to distinguish between correct and incorrect key candidates. Other distinguishers are also used, one uses the Pearson correlation. This is called correlation power analysis (CPA) [2]. In this paper we apply DPA and CPA on two authenticated encryption schemes, namely Keyak and Ascon. The ciphers are candidates in the CAESAR competition for an authenticated encryption scheme. Both of these schemes are based on the duplex construction [3] and apply an iterated permutation on a state. The round functions of these permutations contain a non-linear step which is similar in both ciphers. We present attacks on straightforward hardware implementations and apply them. A countermeasure against DPA and CPA is a threshold implementation. We also present a higher-order DPA attack on a threshold implementation of Ascon.

We summarize our contributions as follows:

- We apply the attack from [4] on Keyak implemented on an FPGA.

- We craft an attack which uses a larger sensitive variable to increase the efficiency compared to the basic attack and apply it.
- We craft an attack for Ascon and apply it on an FPGA implementation.
- We simulate traces for a toy version of Ascon protected by a threshold implementation and attack it using a third-order distinguisher.
- We provide insight in the effect of the linear step of a permutation in DPA.

Related work to the attack on Keyak is work on MAC-Keccak [5][6]. Where they attack the linear part, we attack the non-linear part in the permutation.

2 Background Information

2.1 Differential Power Analysis

DPA is the statistical analysis of the power consumption of a device. In all attacks in this paper we use this statistical analysis to exploit leakage of the power consumption caused by the switching activity of a register. We attack a value of a specific bit called the sensitive variable. Whether or not the value of the sensitive variable switches causes a difference in the power consumption. We exploit this leakage by correlating the actual power consumption with intermediate values. These values can be computed for each trace using known input and part of the key. We pick a sensitive variable that depends on a small number of key bits such that we can compute the intermediate values for all possible candidates for that part of the key. Instead of the correlation we could also compute the difference of means. These methods are explained in detail in [7]. Many power traces are often necessary to be able to distinguish between the correct and incorrect key candidates due to noise. Since many traces are required we use formulas from [8][9] to compute the correlation in a memory efficient way. With these formulas we can split up the trace set in different smaller partitions and compute the correlation for each partition of traces separately. In the end, the resulting coefficients can be recombined. These methods also allow for parallel computation to speed up the process.

2.2 Countermeasures

In applications such as IoT, designers of devices that perform cryptographic operations must protect their designs against such attacks. This can be accomplished on different levels in the architecture of a cipher, for instance in the protocol where the usage of a single key can be limited or in the in actual design of the implementation of the algorithm. There are several techniques for protecting the design on implementation level. One is masking [10][11] where the data is masked and split up into two shares. The algorithm is computed separately on both shares. At the end of the computation the shares are combined to obtain the correct output. The computations on the data and the mask can be done in parallel. A specific type of masking is a threshold implementation where more than two shares are used. These techniques make it harder to correlate the intermediate value with the power consumption as the power consumption is no longer correlated to the data that is processed. A more detailed explanation about threshold implementations is given in section 5.3.

2.3 Higher-order attacks

To attack masked implementations, a simple DPA such as described in Section 2.1 is not going to work. These attacks fall into the category of first-order attacks. To attack implementations with multiple shares, higher-order attacks should be used [12][13]. If an implementation has two or more shares and is implemented correctly, a higher order attack must be used. A second-order attack is similar to a difference of means attack except for the distinguisher. Instead of the mean, the variance is used which is the average of the squared distance to the mean of all the samples in a distribution. The variance of a distribution X is defined as follows.

$$Var(X) = E(X - \mu)^2$$

Where $E()$ is the expected value and μ is the mean of distribution X. Other names for the mean and the variance are respectively the first and second central moment.

When an implementation uses three shares and a second order attack does not work. A third-order attack must be used. The attack is again similar to difference of means except where we use the difference of the mean, the difference of the skewness is used [11]. The skewness determines if a distribution is symmetrical on the mean or leans more to the left or the the right. The skewness of distribution X is defined as follows.

$$Skew(X) = \frac{E(X - \mu)^3}{\sigma^3}$$

From the previous formula the $E(X - \mu)^3$ is also called the third central moment and the skewness is the third standardized moment.

Higher-order attacks typically require a high amount of traces. To compute the difference of skewness in a memory efficient way we use formulas from [14]. With these formulas we can compute the skewness for small partitions and combine them in the end. Doing this reduces the usage of RAM and allows for parallel computation of the results.

2.4 Theoretical success probability

We use a power consumption model where we assume we can exploit the difference in the power consumption of a register flipping or keeping its value. This leakage can be modeled using formulas derived in [4].

$$G_h(\sigma^2) = \int_0^\infty \left(erf\left(\frac{t}{\sqrt{2}\sigma}\right)\right)^{h-1} \left(\mathcal{N}_{(-1;\sigma^2)}(t) + \mathcal{N}_{(1;\sigma^2)}(t)\right) dt$$

$$P_{\text{success}} = G_h \left(\frac{b}{|M|} \right) \tag{1}$$

Where b is the width of the state, h is the number of key candidates, M is the number of traces and $\mathcal{N}_{(\mu,\sigma^2)}(x)$ is a value of the normal distribution with a mean μ and a variance σ^2. The equation uses that the number of traces required to distinguish between two distributions with a certain probability is inversely proportional to the Kullbach-Leibler Divergence. The left part of the equation with the error function presumes no correlation occurs for a hypothesis on an incorrect key candidate. The right part presumes only the switching of the b registers in a state contribute to the noise. Other algorithmic noise can be characterized as additional b'' bits. This means there is a total of $b' = b + b''$ bits that contribute to the algorithmic noise. Noise other than algorithmic noise can also be present but is not considered in this theoretical success probability of a first-order attack on the state.

3 Experimental Setup

The experiments are executed on an FPGA, a Spartan-6 XC6SLX75. The implementations for Keccak-f[1600] and Ascon-128 are written in VHDL. In both permutations one round is computed in a single clock cycle. The computations are done in the combinatorial logic of the FPGA. The registers are updated at the end of a round. Both implementations are adapted to run on the SAKURA-G evaluation board. This board contains an additional FPGA, a Spartan 6 XC6SLX9 which controls the communications between the main FPGA and a PC. The PC is connected through USB using the FTDI interface. To measure the power consumption of the main FPGA the SAKURA-G already has circuitry implemented. There is a connector to measure the raw signal and one where the signal is amplified. We use the amplified signal to measure the power consumption for the experiments.

The oscilloscope we use is a Lecroy Waverunner z610i. It is triggered at the start of each encryption run. The trigger is provided by an I/O pin on the SAKURA-G. The trigger is very stable so the resulting traces are perfectly aligned. The internal clock in both FPGA's of the SAKURA-G run at 48MHz. The internal amplifier has a bandwidth of 360MHz. We used a sampling rate of 500 million samples per second.

4 Keyak

Keyak [15] is a cipher which is a candidate in the Competition for Authenticated Encryption: Security, Applicability, and Robustness or in short CAESAR [16].

Keyak is based on the Motorist authenticated encryption mode defined in [15]. A Motorist consists of several layers. The top layer is the Motorist itself. Below that is the Engine and in the bottom layer are the Pistons. A Piston contains a b-bit state and applies the permutation f to it. The Engine layer

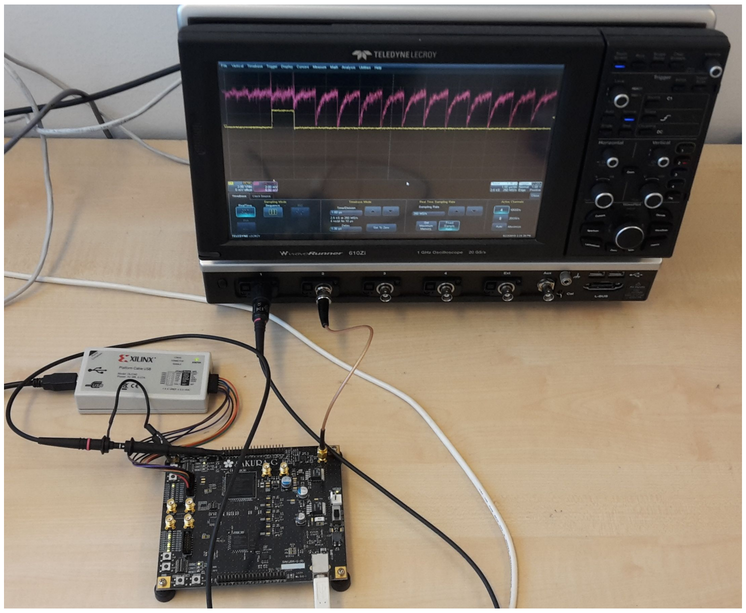


Fig. 1. A photo of the setup for capturing traces with the SAKURA-G

controls the pistons. Pistons work in parallel. An Engine contains at least one Piston. The Engine keeps the Pistons busy and ensures correct use of the input and output for the Pistons. The Motorist layer implements the user interface. It starts a session to encrypt messages or decrypt cryptograms by using the Engine.

In this paper we only look at Lake Keyak obtained from the recommended list of parameters. From this point when we refer to Keyak we mean Lake Keyak. This instance of Keyak has only a single piston which means there is only a single core of Keccak-p. The Keccak-p permutations are derived from the Keccak-f permutations [17]. Keccak-$p[b, n_r]$ is defined by its width b and its number of rounds n_r. Keccak-$p[b, n_r]$ applies the last n_r rounds of Keccak-$f[b]$. In the case of Lake Keyak $b = 1600$ and $n_r = 12$. The permutation Keccak-$p[b, n_r]$ is described as a sequence of operations on a state a that is a three-dimensional array of elements of GF(2), namely $a[5, 5, 64]$ in case of Lake Keyak. Coordinates x and y should be taken modulo 5 and coordinate z should be taken modulo 64. Sometimes an index is omitted implying the statement is valid for all values of the omitted indices. Keccak-$p[b, n_r]$ is an iterated permutation over n_r rounds of R.

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

$$\begin{aligned}
\theta : C_{(x)} &= \sum_{y=0}^{4} A_{(x,y)} \\
D_{(x)} &= C_{(x-1)} + rot(C_{(x+1)}, 1) \\
A_{(x,y)} &= A_{(x,y)} + D_{(x)} \\
\pi \text{ and } \rho : B_{(y,2x+3y)} &= \mathrm{rot}(A_{(x,y)}, r(x,y)) \\
\chi : A_{(x,y)} &= B_{(x,y)} + (B_{(x+1,y)} \times B_{(x+2,y)} + 1) \\
\iota : A_{(0,0)} &= A_{(0,0)} + RC
\end{aligned}$$

All operations are carried out in GF(2). Function $\mathrm{rot}(W, i)$ is a bitwise cyclic shift operation. The constants $r(x, y)$ are rotation offsets. RC is the round constant. For more info see [17]. We refer to the linear steps as $\lambda = \pi \circ \rho \circ \theta$.

To start a session a fixed length key and arbitrary length nonce (in CAESAR also known as the public message number) is absorbed into the state by the Piston. The whole state is used except for one 64-bit lane. In our case we use a 40-byte keypack which contains a padded secret key and a 344-byte nonce. Together that is two complete blocks except for two lanes. The first block containing the 40-byte keypack together with the first 152 bytes of the nonce are absorbed. Following up 12 rounds of Keccak-p are applied. Next the remaining block which contain 192 bytes of the nonce are absorbed and again 12 rounds of Keccak-p are applied. Finally the plaintext can be absorbed but since this is out of scope of the attack, for more details we refer to the Keyak reference document [15]. In both attacks on the internal state of the Piston, where from this point we refer to as the Keyak state, the first block is kept at a constant value. The second block is distinct each time. In our experiments we use random values for the second block. We attack the first round after the second block is absorbed into the state.

4.1 Attack on a single bit of Keyak state

In this attack the sensitive variable is a single bit of the 1600-bit Keyak state. The attack on this sensitive variable is equivalent to the attack on the simulation in [4] where the round function of Keccak-p is attacked. At the previously described point it is possible to compute the sensitive variable which is a function of the bits of the secret state and bits of the second block of the nonce. Once we know the complete secret state, by applying inverse of λ on the bits of the state we found. We can reconstruct the key by applying the inverse of the permutation. Or we can use the state and apply the algorithm from this point. We call these bits, bits from the key state. The variable nonce is called M. Next we compute the sensitive variable using the selection function.

The non-linear step χ of Keccak-p is defined as follows.

$$\chi(a_{(x,y,z)}) \leftarrow a_{(x,y,z)} + (a_{(x+1,y,z)} + 1)a_{(x+2,y,z)}$$

The linear part can be computed separately for different data like the input and the key state after the absorption of the key and combined later before χ.

$$\chi(\lambda(k + m)) = \chi(\lambda(k) + \lambda(m))$$

This way the input of χ can be split up into bits from the key state and bits from the message. We compute the sensitive variable as follows.

$$\chi(a_0) \leftarrow k_0 + m_0 + (k_1 + m_1 + 1)(k_2 + m_2)$$

Where m_* are bits of $\lambda(M)$ and k_* are the bits of $\lambda(keystate)$. We are interested in the activity d of the register where the bit is stored. The values depend on the previous value a_0 of the register which is unknown as it depends on the key.

$$\begin{aligned} d &= a_0 + k_0 + m_0 + (k_1 + m_1 + 1)(k_2 + m_2) \\ d &= a_0 + k_0 + m_0 + k_1k_2 + m_1k_2 + k_2 + m_2k_1 + m_2m_1 + m_2 \\ d &= a_0 + k_0 + (k_1 + 1)k_2 + m_0 + (m_1 + 1)m_2 + m_2k_1 + m_1k_2 \end{aligned}$$

The result of $a_0 + k_0 + (k_1 + 1)k_2$ is equal for each trace and contributes a constant amount to the activity so it can be removed. This results in the following selection function.

$$S(M, K^*) = m_0 + (m_1 + 1)m_2 + m_2k_1^* + m_1k_2^*$$

The selection function contains two key bits resulting in four key candidates.

4.2 Result Keyak single bit

We can compute the theoretical success probability using the formula's from Section 2.4. Figure 2 shows this probability on the y-axis with number of traces on the x-axis where $b = 1600$ and $h = 4$. The figure also shows the result of the attack on a single bit of the Keyak state. When sixty thousand or more traces are used the success probability approaches one. As we can see the result from the attack on the FPGA is not equal to the theory. We need twice as many traces compared to the theoretical success probability. Which means other noise then just noise caused by the switching of the registers is present in the traces. Averaging traces over equal inputs did not reduce the noise much. The greater part of the remaining noise is caused by the algorithmic noise of the combinatorial logic of the FPGA. We go into more detail in Section 5.2.

4.3 Attack on a row of Keyak state

Using the previous attack we can obtain two correct key bits with a certain probability. To find the key bits of one row using the previous attack we have to guess 5 times 2 key bits, in total this results in 5 key bits. We try to find an attack where we only need to guess 5 key bits which hopefully increases the success probability to make the attack more efficient. With the new attack, we find the value of 5 bits in a row of the key state. To accomplish this we try to correlate the traces with values of the sensitive variable which consists of more then a single bit. A row of the state seems to be an efficient choice as for those five bits we only need to guess five key bits. To create a selection function for

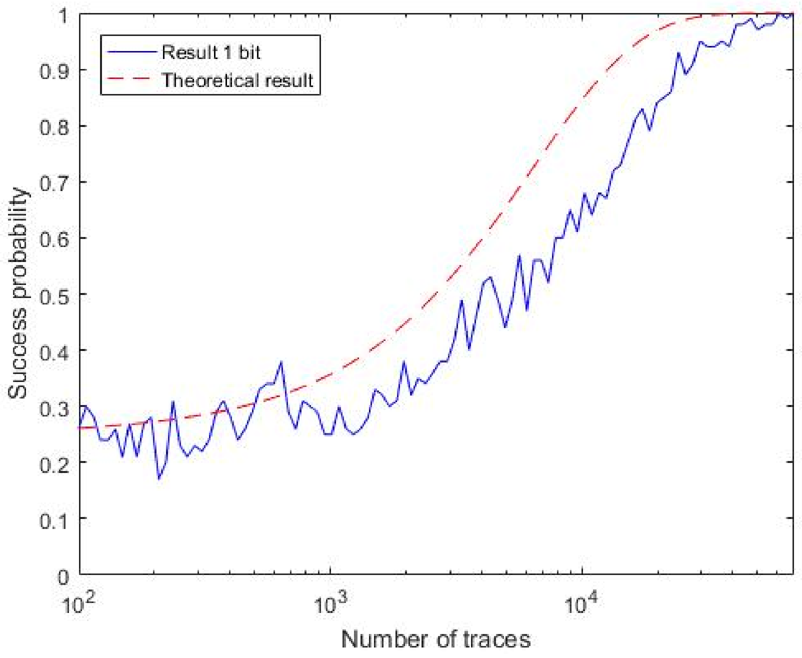


Fig. 2. Success rate of attack on single bit of the Keyak state

a row of five bits of the state of Keyak it seems like a good idea to extend the previous attack on one bit in the following way. One might think this can easily be accomplished by summing up the resulting bits from the previous selection function for each bit in the targeted row.

$$\begin{aligned} S(M,K^*) = {} & m_0 \oplus (m_1 \oplus 1)m_2 \oplus m_2k_1^* \oplus m_1k_2^* \\ & + m_1 \oplus (m_2 \oplus 1)m_3 \oplus m_3k_2^* \oplus m_2k_3^* \\ & + m_2 \oplus (m_3 \oplus 1)m_4 \oplus m_4k_3^* \oplus m_3k_4^* \\ & + m_3 \oplus (m_4 \oplus 1)m_0 \oplus m_0k_4^* \oplus m_4k_0^* \\ & + m_4 \oplus (m_0 \oplus 1)m_1 \oplus m_1k_0^* \oplus m_0k_1^* \end{aligned}$$

This straightforward approach does not work. If we recall how the selection function was derived we see that the activity d of a sensitive variable depends on its previous value a_0. When one bit is attacked this value is constant and is irrelevant when splitting the set up based on the intermediate value for each key candidate. It is not important how the set is split up, as long as it is split up correct for the correct key candidate which is why it is removed from the selection function. When doing this for five bits at the same time the previous values are still constant but now these values are important. They are unknown so we also need to guess them. To do this we use a different approach.

The first step of the attack is to create 32 bins. Each bin contains the mean trace of all traces with same value of the targeted row after $\lambda(M)$ is computed on the variable part of the nonce.
Next we compute a matrix containing all the values for the sensitive variable. Which is a 32×32 matrix as there are 32 different inputs and key candidates. To do this we exploit the same leakage as in the previous attack on a single bit. For each element in the matrix we compute the output of χ based on the input and the key candidates.
With these values we compute the result matrix R with dimensions $K \times T$ where K equals the number of key candidates and T equals the number of samples in each trace. Each element contains the correlation between the values of the sensitive variable for that key candidate and actual power consumption values for that time sample. The row in R with the highest correlation value corresponds to the key candidate with the highest probability to be the correct one.

4.4 Result Keyak row

Figure 3 shows the result of the attack on a row of the Keyak state. If we compare this result with the result of the attack on a single bit we see this attack is less efficient as the success probability is lower for a similar amount of traces. To compare these results we use the single bit approach where do the attack five times on each bit of the row independently and view it as a success if all five the attacks result in the correct key hypothesis without taking inconsistent key bit values suggested by neighboring sensitive bits into account. The success probability of the attack on the row is nearly the same as the of the attack on

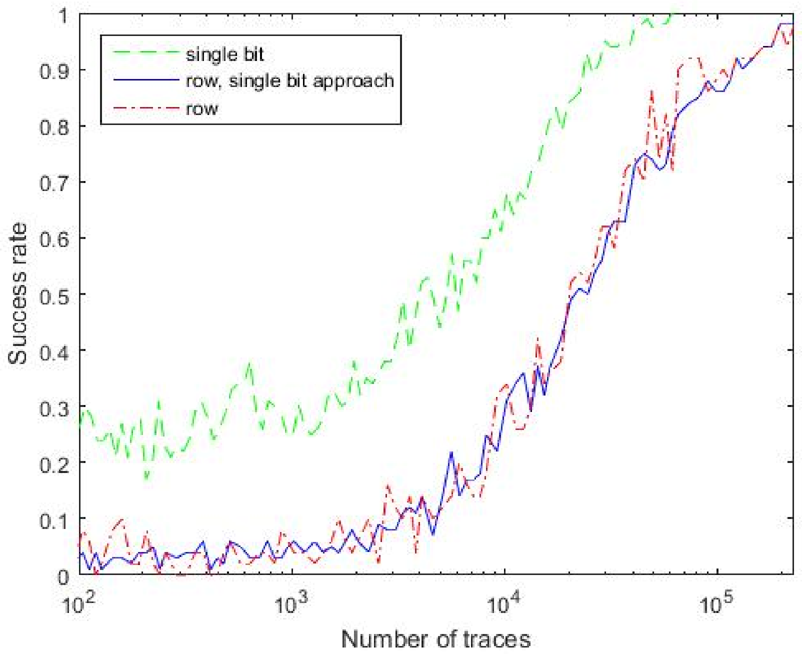


Fig. 3. Result of attack on row compared with attack on single bit

a single bit. This means that both attacks are equivalent. The basic attack on a single bit is already efficient as increasing the size of the sensitive variable does not reduce the number of traces required for the attack.

5 Ascon

Like Keyak, Ascon [18] is a CAESAR candidate. From the list of recommended parameters we use Ascon-128, so from this point when we refer to Ascon, we mean Ascon-128. This instance of Ascon uses 128-bit keys and nonces, the block-size of the input data is 64-bit.
The authenticated encryption algorithm Ascon uses a duplex construction. Figure 4 shows the encryption process of Ascon. It takes four inputs, plaintext P_i, associated data A_i, nonce M and a key K. The block cipher produces two outputs, the ciphertext C_i and a tag T. The permutation is denoted by p^a and p^b. The values $a = 12$ and $b = 8$ equal the number of rounds. The nonce is public. The tag is used during decryption to authenticate the ciphertext. During decryption the algorithm uses the tag as an input and produces the plaintext as output along the result of the validation of the tag. If a tag is invalid no output is returned.

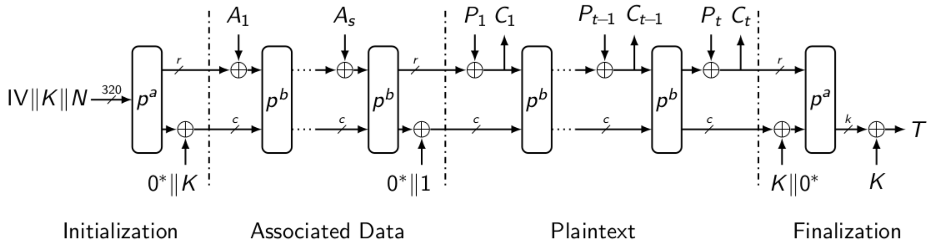


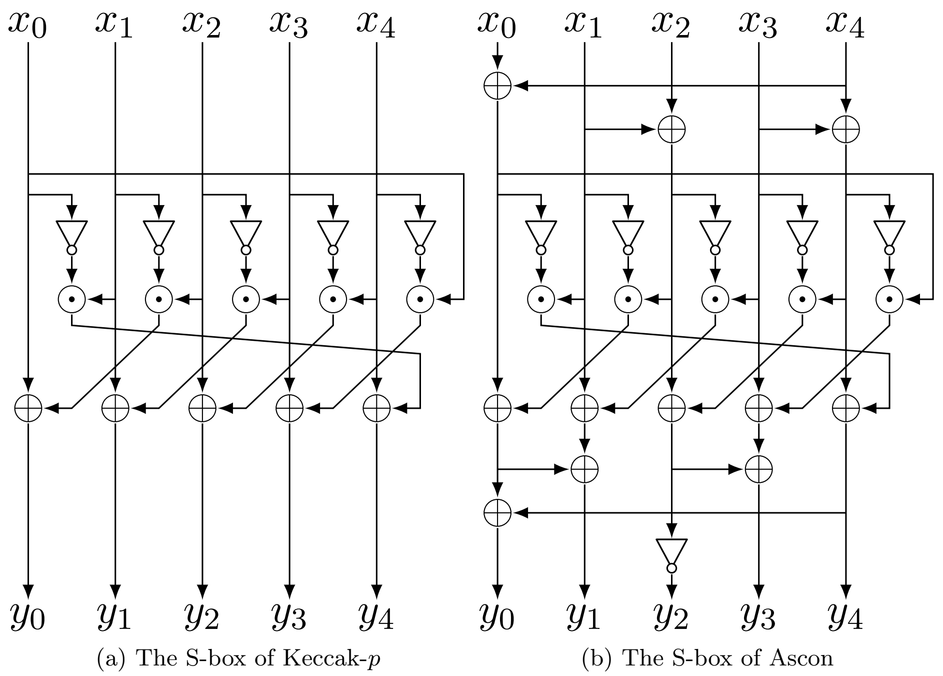
Fig. 4. Ascon encryption

The internal state of the algorithm consists of 320 bits which is split up into five registers of 64 bits, named x_0 to x_4. During the initialization of the algorithm, a 64-bit constant IV is written in register x_0. Registers x_1 and x_2 contain the key and the variable nonce is stored in registers x_3 and x_4. The associated data and plaintext are padded to have a length of a multiple of 64 bits. When the internal state is initialized p^a is applied. Next, the optional associated data is absorbed into the state. After this the plaintext is absorbed into the state. After each block of associated data and plaintext p^b is applied except for the final block of plaintext. During the finalization of the part of the algorithm p^a is again applied and a tag is produced.

The round function or permutation used in Ascon consists of three steps. The first part is the addition of a round constant to register x_2 on low indices which can be computed as follows for each round i.

$$RC_i = \texttt{0xF} - i || \texttt{0x0} + i$$

Where || means the concatenation of the two parts. The second part is a non-linear five-bit S-box with algebraic degree two, which takes one bit from each register shown in Figure 6a and replaces it with the output of the S-box. It is a variant of the S-box used in Keyak. In Figure 5 we can see similarities between the S-box of Ascon and Keyak. The designers of Ascon added a few XOR's and a not.



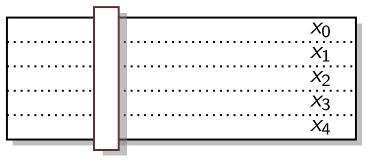
(a) The S-box of Keccak-p (b) The S-box of Ascon

Fig. 5. Comparison between S-boxes

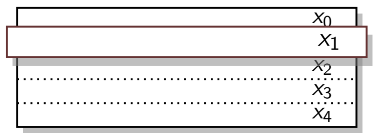
The final step is the linear diffusion layer where each register is rotated twice and XOR'ed with itself which is called $\Sigma_i(x_i)$. Below are the expressions of the linear diffusion layer.

$$\Sigma_0(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28)$$

In permutation Keccak-p the non-linear step is at the end of a round where in the permutation of Ascon it is at the start.



(a) Grouping of bits S-box



(b) Input diffusion layer

Fig. 6. Inputs for the functions of the round function of Ascon [18]

5.1 Attack on Ascon

Since we know the contents of the state at initialization, except for the key part. And we can vary the nonce each run, we pick the end of the first round as our point of attack. As sensitive variable we pick a bit of x_0.
We denote the initialized registers as x_i. In this attack we are only interested in the first round so x_i' denotes the register after the first round. The output the the S-box is specified by y_i.
For the attack intermediate values have to be computed for each different key guess on the nonce. The linear step for x_0 is the following:

$$\Sigma_0(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \tag{2}$$

The output of the non-linear S-box for x_0 can be expressed in the algebraic normal form as follows. Where y_0 is the output and x_i are the input bits.

$$y_0 = x_4x_1 + x_3 + x_2x_1 + x_2 + x_1x_0 + x_1 + x_0$$

This can be rewritten to an expression with a single quadratic term.

$$y_0 = x_1(x_4 + x_2 + x_0 + 1) + x_3 + x_2 + x_0$$

We derive the selection function similar as in [4, Section 3]. We exploit the leakage of the power consumption similar as in the previous attacks. If we rewrite the previous equation.

$$y_0 = x_1(x_4 + 1) + x_1x_2 + x_1x_0 + x_3 + x_2 + x_0$$

All the bits that contribute a constant amount to the activity of the register, namely $x_1x_2 + x_1x_0 + x_2 + x_0$ can be removed.

$$y_0 = x_1(x_4 + 1) + x_3$$

As a result the intermediate value now only depends on one bit from one register of the key and two bits from two registers of the nonce.

$$y0 = k(m' + 1) + m \tag{3}$$

Where $m = x_3, m' = x_4$. If we combine equations (2) and (3) we get the following selection function for bit 0.

$$\begin{aligned} S_i(M, K^*) = {} & k_0^*(m'_0 + 1) + m_0 + k_1^*(m'_{45} + 1) \\ & + m_{45} + k_2^*(m'_{36} + 1) + m_{36} \end{aligned} \tag{4}$$

We can generalize this for all bits in the register.

$$\begin{aligned} S_i(M, K^*) = {} & k_0^*(m'_i + 1) + m_i + k_1^*(m'_{i+45} + 1) \\ & + m_{i+45} + k_2^*(m'_{i+36} + 1) + m_{i+36} \end{aligned} \tag{5}$$

Where the M is the 128-bit nonce split up into two registers m, m' and k_i^* is a bit from a key guess. Additions to index i are done modulo 64 because the bits are located in a 64-bit register. Since there are three key bits in the selection function, there are eight key candidates.

By attacking register x'_0 we obtain only half of the key. To obtain the remaining key bits we need to attack register x'_1. This is the only S-box which has a quadratic term containing a bit from register x_2. To create a selection function for this register we derive the selection function similar as for x'_0. In the attack it will not be possible to distinguish the XOR between x_1 and x_2 so their result will be regarded as one term x_{12}.

$$\begin{aligned} S_i(M, K^*) = {} & m_i(k_0^* + 1) + m'_i + m_{i+3}(k_1^* + 1) \\ & + m'_{i+3} + m_{i+25}(k_2^* + 1) + m'_{i+25} \end{aligned} \tag{6}$$

With the attack on register x'_1 we obtain an XOR between the key bits in register x_1 and x_2. Looking at the remaining output registers of the S-box we have:

$$\begin{aligned} y_2 &= x_4x_3 + x_4 + x_2 + 1 \\ y_3 &= x_4x_0 + x_4 + x_3x_0 + x_3 + x_2 + x_1 + x_0 \\ y_4 &= x_4x_1 + x_4 + x_3 + x_1x_0 + x_1 \end{aligned}$$

We can see that y_2 and y_3 have no non-linear terms with a key and a nonce register so they can not be attacked. Register y_4 does have a non-linear term with a key and a nonce register and can be attack. The expression can be rewritten in a similar way.

$$\begin{aligned} y_4 &= x_4x_1 + x_4 + x_3 + x_1x_0 + x_1 \\ y_4 &= x_4(x_1 + 1) + x_3 + x_1x_0 + x_1 \\ y_4 &= x_4(x_1 + 1) + x_3 \end{aligned}$$

Since we are already be able to obtain the whole key by attacking register x'_0 and x'_1 register x'_4 was not attacked.

5.2 Results attack on Ascon

Figure 7 shows the success probability of the attack on register x'_0 approaching 1 at fifty thousand traces. With this attack we obtain the first half of the key. The success probability of register x_1' is slightly lower. Using the result from register x_0' we can obtain the complete key. Looking at the results of the theoretical success probability computed using Section 2.4 we observe that it is lower.

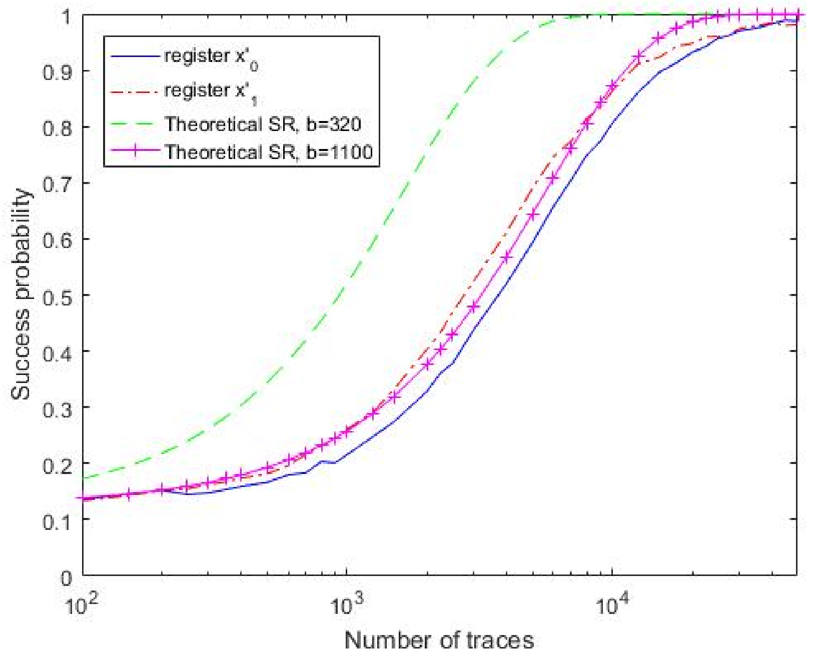
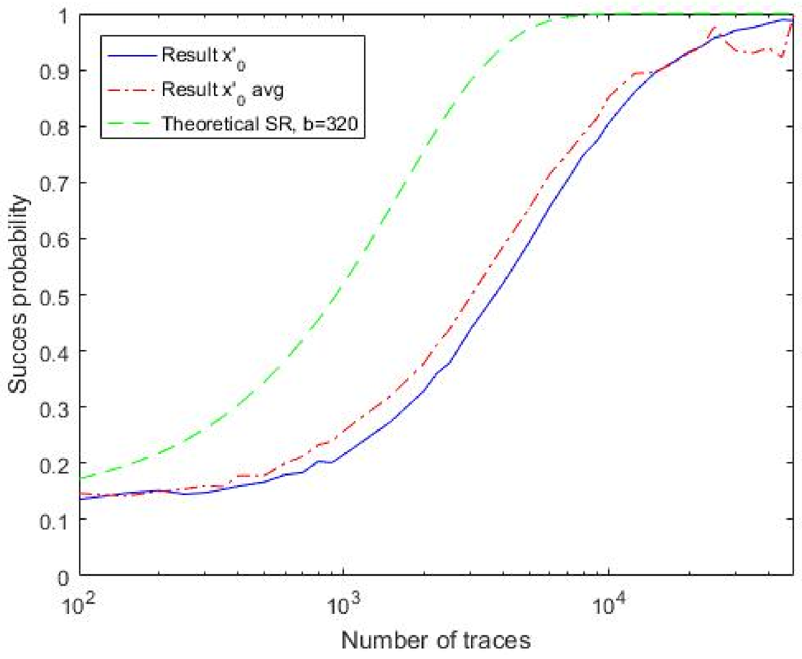


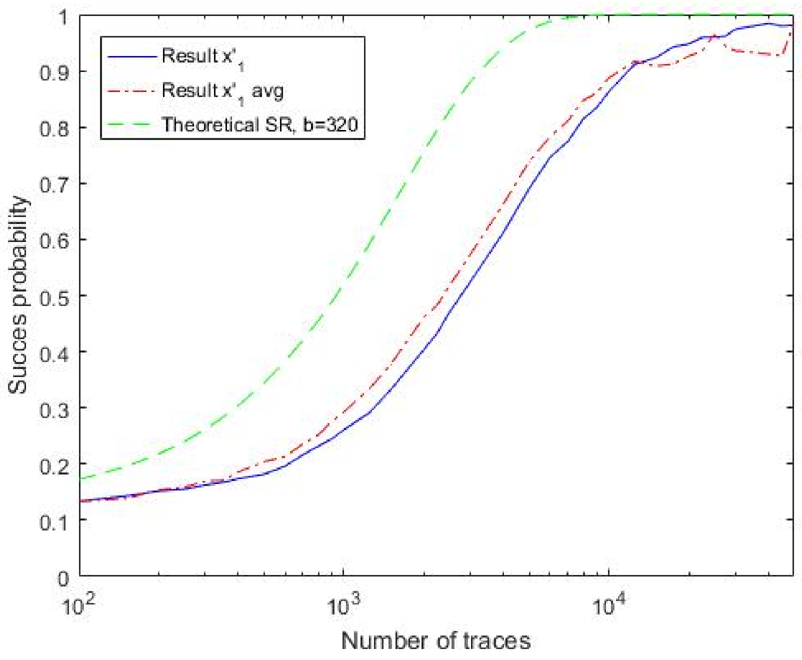
Fig. 7. Success rate of attack on Ascon bit by bit

If we compare the results from the attack on Keyak with the results from the attack on Ascon we see that it requires less traces to attack Ascon. This is due to the fact that the internal state of Ascon is smaller compared to the internal state of Keyak. When the state in a fully parallel implementation is larger there will be more algorithmic noise which makes it harder to distinguish between correct and incorrect key candidates.

Algorithmic and other noise In Figure 7 we saw that the success probability is lower compared to the theoretical success probability computed using Equation 1. This indicates the noise levels are a lot higher in the FPGA that was used compared to the noise of the model. If we compute it for a state with width $b = 1100$ and $h = 8$ we see this is close to the actual result. This means the algorithmic noise $320/1100 \approx 0.29$ which is approximately 29%. This shows the flipping of the registers only makes up 29% of the noise.



(a) Result noise reduction register x'_0



(b) Result noise reduction register x'_1

Fig. 8. Result attack Ascon with noise reduction by averaging

We observed a lot of the generated noise is other than algorithmic caused by switching of the registers. We assume the other noise is unrelated to the input. This makes it possible to reduce it by averaging over a set amount of traces with an equal input. We averaged over fifty equal inputs and the results can be observed in Figure 8a and 8b. Both figures show that the success probability is a bit higher compared to the result where no traces are averaged. The result is still not close to the theoretical curve where the state contains 320 bits. This means that noise other than algorithmic caused by flipping of the registers or unrelated to the input there is still a lot of noise present in the traces. Since it does not reduce by averaging over equal inputs, it is related to the input. The remaining noise is caused by the combinatorial logic. This leaves more possibilities to attack. The current attack only attacks flipping of the registers. Since the combinatorial logic also leaks this could also be used to attack the implementation. Perhaps exploiting leakage caused by bits flipping in the registers combined with the combinatorial logic could lead to higher success probabilities.

5.3 Threshold Implementations

A technique to protect hardware implementations against first-order DPA is a threshold scheme [19]. To protect against first-order DPA, the number of required shares N is the algebraic degree of the non-linear function plus one. A threshold implementation or TI has three requirements: non-completeness, correctness and uniformity. An implementation is non-complete if at most $N - 1$ shares are combined at once. Linear steps can be computed separately on each share and recombined in the end. In the non-linear step shares must be combined. This looks as follows for a TI with three shares.

$$\begin{aligned} A' &= f_a(B, C) \\ B' &= f_b(C, A) \\ C' &= f_c(A, B) \end{aligned}$$

A threshold implementation is correct if the following holds.

$$f(A \oplus B \oplus C) = f_a(B, C) \oplus f_b(C, A) \oplus f_c(A, B)$$

Not all non-linear functions keep the uniformity of their input shares, this is a problem because it has been proven that if the shares are uniformly distributed, correct and non-complete the implementation is secure against first-order DPA [20]. It is possible to preserve the uniformity by adding new random bits since both Ascon and Keyak do not a have uniform round function this is required. It is possible to reduce the amount of new random bits that are required each round [21] for Keyak. Recently another technique was proposed to increase the state by a few bits to keep it uniform [22]. This way, no new random bits are required each round. Since Ascon uses a variation of Keyak's S-box this works for Ascon as well.
The threshold implementation we use has three shares. One round of the round function is computed in parallel on all three shares each clock cycle.

Power consumption model We use the same model as in [4] where we compute the Hamming distance between the input and output of a round. This models the switching of a register to leak information. If the Hamming distance between two bits is 0 it contributes +1 to the power consumption and if it is 1 it contributes -1. A bit in a state can be represented by three bits in the threshold implementation, for 0 there are 000, 011, 101 or 110, for 1 there are 111, 001, 010 or 100. Since the threshold implementation is initialized randomly the occurrence of each triplet is also random. For each triplet in the state we show the contribution of the power consumption in Table 1. This results in a mean of 0 and a variance of 3.

Table 1. Modeled power consumption of threshold implementation

Bit 0	Contribution
000	+3
011	-1
101	-1
110	-1

Bit 1	Contribution
111	-3
001	+1
010	+1
100	+1

5.4 Attack on TI Ascon

In this attack we attack a threshold implementation. A fully parallel threshold implementation with an internal state of 320 bits will have a lot of algorithmic noise. Figure 9 shows the theoretical success probability of a fully parallel threshold implementation where the width of the state $b = 320$ bits using functions from [4, Section 4C]. We see that the required amount of traces for a success probability higher than guessing the correct key candidate requires over 10^8 traces. To collect this many traces is not feasible in our experimental setting. To be able to obtain the correct key with a feasible amount of traces, we decreased the size of the state to a toy version. To do this we modify the implementation to have a smaller internal state. In Ascon the state consists of five 64-bit registers, to reduce the amount of algorithmic noise we study a toy-sized version of Ascon with 4-bit words. In total the state consists of 20 bits and with three shares there are 60 bits.

The round constant is changed to the following where i is the round number starting from 0.

$$x_2 = x_2 \oplus (\texttt{0xF} - i * \texttt{0x1})$$

The substitution layer is affected as the S-box is shared between the three shares. If we have three shares A, B and C the sharing works as follows. The shares are split up into 5 registers ranging from 0 to 4. We split the S-box up into three

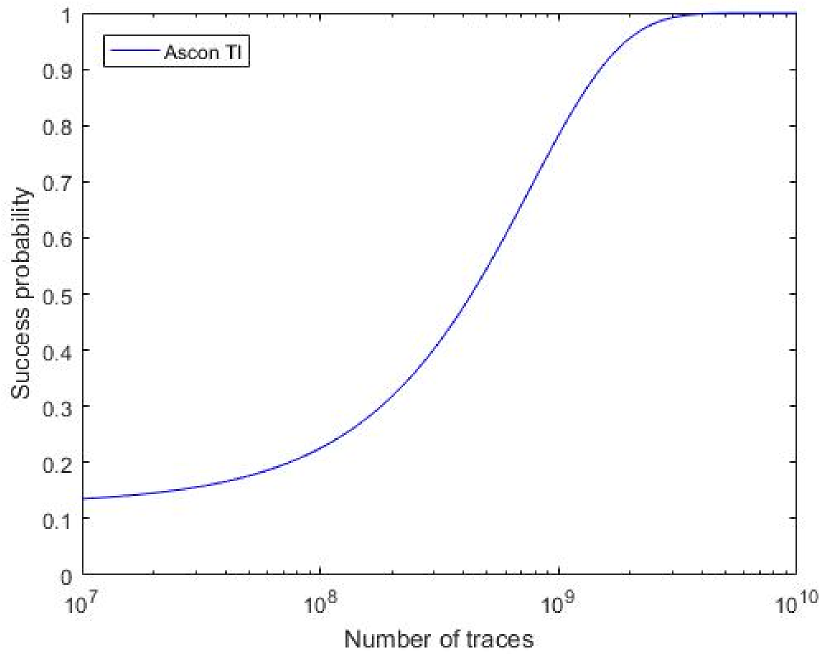


Fig. 9. Theoretical success probability Ascon TI

parts, two linear parts and a non-linear part in between.

$$
\begin{aligned}
R_{00} &= A_0 \oplus A_4 \\
R_{01} &= A_1 \\
R_{02} &= A_2 \oplus A_1 \oplus RC \\
R_{03} &= A_3 \\
R_{04} &= A_4 \oplus A_3
\end{aligned}
$$

$$
\begin{aligned}
S_{00} &= R_{10} \oplus (\neg R_{11} \wedge R_{12}) \oplus (R_{11} \wedge R_{22}) \oplus (R_{12} \wedge R_{21}) \\
S_{01} &= R_{11} \oplus (\neg R_{12} \wedge R_{13}) \oplus (R_{12} \wedge R_{23}) \oplus (R_{13} \wedge R_{22}) \\
S_{02} &= R_{12} \oplus (\neg R_{13} \wedge R_{14}) \oplus (R_{13} \wedge R_{24}) \oplus (R_{14} \wedge R_{23}) \\
S_{03} &= R_{13} \oplus (\neg R_{14} \wedge R_{10}) \oplus (R_{14} \wedge R_{20}) \oplus (R_{10} \wedge R_{24}) \\
S_{04} &= R_{14} \oplus (\neg R_{10} \wedge R_{11}) \oplus (R_{10} \wedge R_{21}) \oplus (R_{11} \wedge R_{20})
\end{aligned}
$$

Rotate index i, to compute S_{ij} using R_{ij}.

$$
\begin{aligned}
A'_0 &= S_{00} \oplus S_{04} \\
A'_1 &= S_{01} \oplus S_{00} \\
A'_2 &= \neg S_{02} \\
A'_3 &= S_{03} \oplus S_{02} \\
A'_4 &= S_{04}
\end{aligned}
$$

Except for the RC which is only added at one share, the linear steps are equal for register B and C. The linear diffusion layer is also affected, the rotations are computed modulo the size of the registers and result in the following expressions.

$$
\begin{aligned}
\Sigma_0(x_0) &= x_0 \oplus (x_0 \ggg 3) \oplus (x_0 \ggg 0) \\
\Sigma_1(x_1) &= x_1 \oplus (x_1 \ggg 1) \oplus (x_1 \ggg 3) \\
\Sigma_2(x_2) &= x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 2) \\
\Sigma_3(x_3) &= x_3 \oplus (x_3 \ggg 2) \oplus (x_3 \ggg 1) \\
\Sigma_4(x_4) &= x_4 \oplus (x_4 \ggg 3) \oplus (x_4 \ggg 1)
\end{aligned}
$$

Since $28 \mod 4 = 0$ the two XOR's cancel each other out which results in a shorter linear expression.

$$\Sigma_0(x_0) = (x_0 \ggg 3) \tag{7}$$

To attack the y_0 register we can combine Equation (3) from the attack on the unprotected implementation and $\Sigma_0(x_0)$ to obtain a selection function.

$$S_i(M, K^*) = k_0^*(m'_{i+1} + 1) + m_{i+1}$$

The equation contains only one key bit, as a result there are two key candidates in the attack.

To attack register y_1 we use a similar selection function as in previous attack on the unprotected implementation.

$$\begin{aligned} S_i(M, K^*) = {} & m_i(k_0^* + 1) + m_i' + m_{i+3}(k_1^* + 1) \\ & + m_{i+3}' + m_{i+1}(k_2^* + 1) + m_{i+1}' \end{aligned}$$

This equations contains three key bits and as a result there are eight key candidates to be considered in the attack.

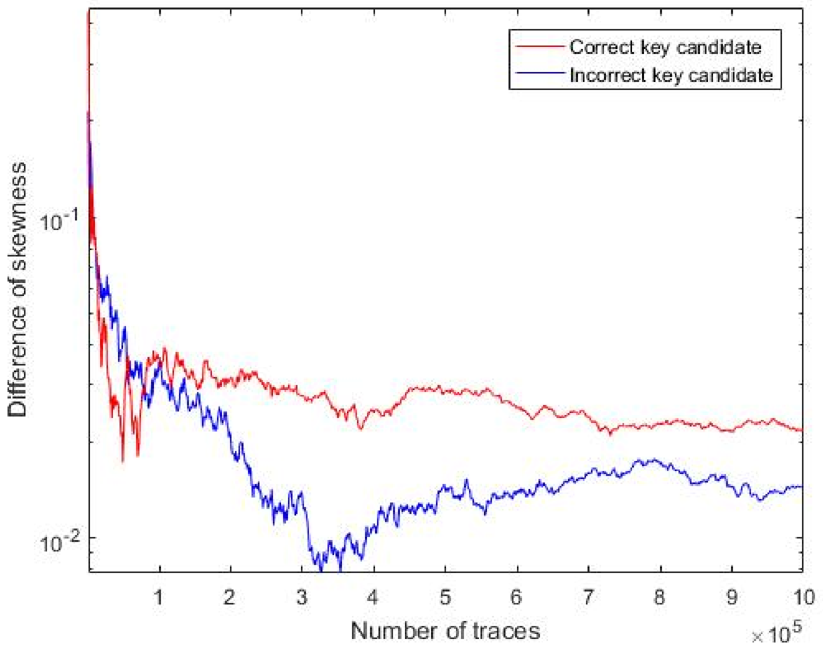
Since the implementation is protected with a threshold, the mean of the power consumption of the two sets for each key candidate will be equal and a first-order attack with distinguisher like difference of means or the Pearson correlation will not work. Unless there is some unforeseen leakage but we assume this is not the case. As a consequence, a higher order distinguisher is required. In this attack we will use a third order attack, the difference of skewness. The skewness is defined in Section 2.3. We simulate traces using the power model discussed in 5.3. For the attack we simulate the first twelve rounds of Ascon which corresponds to the initialization part.

5.5 Results attack on Ascon TI

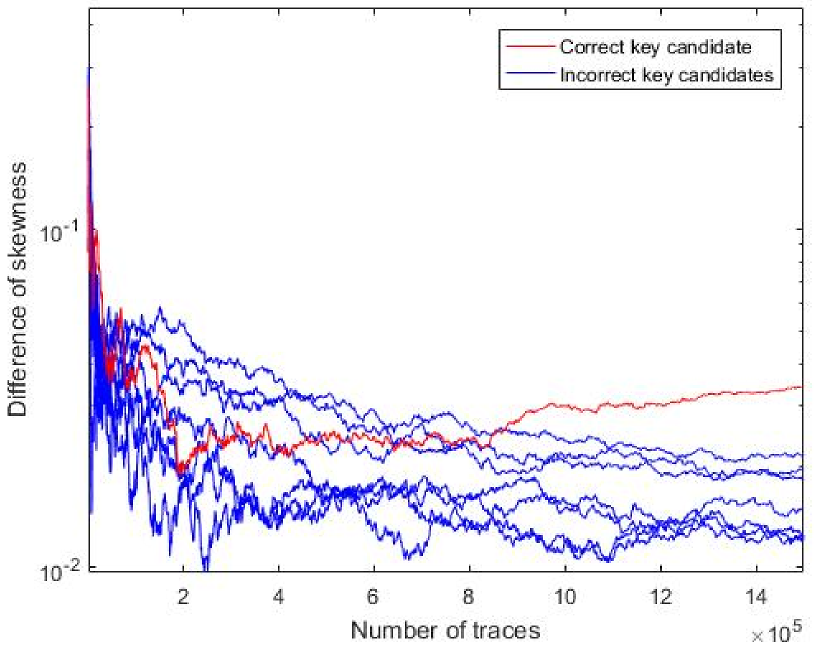
Figure 10 shows the result of the attack on the simulated traces for register x_0' and x_1' where the difference of skewness was used as a distinguisher. From both figures can be observed that it is possible to obtain the correct key candidate with the attack. Register x_0' required 150.000 traces, where register x_1' required 900.000. This large difference in traces is explained in Equation 7 where two of the three XOR operations use an equal rotation offset so they cancel out. This means instead of having to guess three key bits for this register, we only need to guess a single key bit. This shows the effect of the linear diffusion layer after the S-box in Ascon. Having this linear layer increases the difficulty of the attack as we need to guess two additional key bits. If we compare these results with the attack on the simulated traces in [4] on Keyak we see that required amount of traces scales by the third power with the size of the state. If we would simulate traces for the full Ascon state this number reaches billions as Figure 9 suggests. Comparing our results with the results on Keyak we see a similar trend. For this reason we did not simulate traces for a larger state. Also acquiring real traces from the SAKURA-G would be unfeasible.

6 Conclusion

In this paper we performed an attack on a fully parallel unprotected FPGA implementation of Keyak. We improved the attack to be more efficient on the state by attacking a row instead of a single bit. For Ascon we created an attack and applied it on a parallel hardware implementation. To reduce the electrical



(a) Simulation Ascon register x'_0



(b) Simulation Ascon register x'_1

Fig. 10. Simulation results for Ascon TI with 20-bit state

noise we averaged traces with an equal input. We also created an attack for a threshold implementation of Ascon. We simulated traces for a toy-sized version and attacked those to reduce the required number of traces. We omit results for an attack on simulated traces for Keyak as this can be found in [4, Section 5E].

The attack on Keyak resulted in a lower success probability compared to the attack in the simulated traces. This means more noise is present in the traces compared to what was taken into account in the model. In the attack on Ascon we tried to reduce the noise by averaging traces. This only reduced the noise slightly which means the greater part of the noise is related to the input. As the model only takes algorithmic noise caused by switching of values in registers into account, this led to the conclusion that a lot of noise present in the traces is algorithmic noise generated by the combinatorial logic.

Our approach to create a more efficient attack on the state, resulted in an attack with a similar success probability, this is due to the attack being equivalent. It did take less time to do the computations for the attack on a row. Creating an attack that is more efficient compared to the attack on a single bit based on the required amount of traces is regarded as future work. We were able to successfully attack the simulated traces of the protected implementation of Ascon. Even though it was a simulation of a toy-sized implementation it gives insight in the effect of a linear layer after the S-box. The addition of this layer makes it much harder to attack the implementation. To attack real traces of a full implementation that is protected with a threshold would be infeasible.

7 Acknowledgments

This work was supported in part by a project funded by Dark Matter LLC. Additionally we would like to thank Lejla Batina for her feedback on the project.

References

1. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology — CRYPTO' 99. Springer Nature (1999) 388–397
2. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Lecture Notes in Computer Science. Springer Nature (2004) 16–29
3. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Permutation-based encryption, authentication and authenticated encryption. Directions in Authenticated Ciphers (2012)
4. Bertoni, G., Daemen, J., Debande, N., Le, T.H., Peeters, M., Van Assche, G.: Power analysis of hardware implementations protected with secret sharing. In: Proceedings - 2012 IEEE/ACM 45th International Symposium on Microarchitecture Workshops, MICROW 2012, Institute of Electrical and Electronics Engineers (IEEE) (dec 2012) 9–16
5. Taha, M., Schaumont, P.: Side-channel analysis of MAC-keccak. In: 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), Institute of Electrical and Electronics Engineers (IEEE) (jun 2013)

6. Luo, P., Fei, Y., Fang, X., Ding, A.A., Kaeli, D.R., Leeser, M.: Side-channel analysis of MAC-keccak hardware implementations. In: Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy - HASP '15, Association for Computing Machinery (ACM) (2015)
7. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: Revealing the secrets of smart cards. Volume 31. Springer Science & Business Media (2008)
8. Dunlap, J.W.: Combinative properties of correlation coefficients. The Journal of Experimental Education **5**(3) (jan 1937) 286–288
9. Bottinelli, P., Bos, J.W.: Computational aspects of correlation power analysis. Journal of Cryptographic Engineering (feb 2015) 1–15
10. Prouff, E., Rivain, M.: Masking against side-channel attacks: A formal security proof. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, Springer Science & Business Media (2013) 142–159
11. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Advances in Cryptology — CRYPTO' 99. Springer Nature (1999) 398–412
12. Peeters, E., Standaert, F.X., Donckers, N., Quisquater, J.J.: Improved higher-order side-channel attacks with fpga experiments. In: International Workshop on Cryptographic Hardware and Embedded Systems, Springer (2005) 309–323
13. Standaert, F.X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The world is not enough: Another look on second-order DPA. In: Advances in Cryptology - ASIACRYPT 2010. Springer Nature (2010) 112–129
14. Pebay, P.P.: Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. Technical report, Sandia National Laboratories (sep 2008)
15. Bertoni, G., Daemen, J., Peeters, M., van Assche, G., van Keer, R.: Caesar submission: Keyak v2 (2016)
16. Abed, F., Forler, C., Lucks, S.: General overview of the authenticated schemes for the first round of the caesar competition. Technical report, Cryptology ePrint Archive: Report 2014/792.[2] CAESAR submissions, second-round candidates. Available: http://competitions. cr. yp. to/caesar-submissions. html (2014)
17. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The keccak reference (January 2011) http://keccak.noekeon.org/.
18. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.1 submission to caesar (2015)
19. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Information and Communications Security. Springer Science & Business Media (2006) 529–545
20. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. Journal of Cryptology **24**(2) (oct 2010) 292–321
21. Bilgin, B., Daemen, J., Nikov, V., Nikova, S., Rijmen, V., Assche, G.V.: Efficient and first-order DPA resistant implementations of keccak. In: Smart Card Research and Advanced Applications. Springer Science & Business Media (2014) 187–199
22. Daemen, J.: Changing of the Guards : a simple and efficient method for achieving uniformity in threshold sharing. (2016) 1–8