FIRST ORDER MASKING IMPLEMENTATION OF PICCOLO-128

Hamed Ramzanipour

June 2022

Abstract

One of the main aims of developing an efficient cryptographic algorithm for use in different systems with a limited implementation area is to achieve lightweight implementation while ensuring security. On the other hand, one of the most used attacks against cryptographic systems is side-channel attacks(SCA), specifically differential power analysis(DPA) attacks. Hence, one of the criteria of a lightweight cryptographic scheme is to protect against such attacks. threshold implementation(TI) is one of the approaches for countering the DPA, in which random shares are allocated to the inputs of various levels of encryption to eliminate power trace leakage information. In this study, the TI is present by employing 4 shares on Piccolo, an ultra-lightweight block cipher, in the context of VHDL language by using ISE software. In the theory phase, a substitution box is constructed for first-order masking with its algebraic normal form(ANF), and these connections are then used to implement Piccolo masked in the design phase. Ultimately, the TI of the Piccolo cipher is presented, including a comparison to the unprotected version.

Keywords: threshold implementation(TI), first-order masking, differential power analysis(DPA), Piccolo cipher

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A cryptographic system is considered secure if its design assumptions and components are not vulnerable to statistical analysis [21]. Hardware attacks, in addition to statistical analysis, refer to a variety of other kinds of attacks. If the attacker has physical access to the encryption algorithm, the attacker can obtain the secret values of the algorithm using leakage information. Electromagnetic radiation, the sound of the encryption process, execution time, power traces, and so on are instances of leakage [11]. Hence, after the proposed concept, the definition of security was modified.

Since its passive and non-aggressive character, power analysis [10] has received the most attention from attackers of all approaches to physical attacks. Following the presentation of various schemes for obtaining the sensitive values of an algorithm using power consumption trace analysis in recent years, some countermeasures such as the implementation of masking are employed to increase security against power analysis [5]. The masking approach in cryptography hardware implementation can provide proven protection against power analysis attacks.

In non-ideal conditions of hardware implementation, optimal processing time (the delay of the signal propagation at the gate level) and area are the most challenging parts of the masking schemes. Therefore, providing security against power analysis attacks, in the hardware implementation phase of the masking and its software simulation gives two distinct concepts. The threshold implementation(TI) [3], which was presented in

2014, is one of the solutions to increase the security level of the masking schemes despite the hardware challenges. This method provided proven security in the same order against power analysis attacks. The mentioned masking approaches have been applied on various cryptographic algorithms such as AES [2, 14], PRESENT [6, 17], PRINCE [4, 15], and Keccak [21] thus far.

Piccolo is one of the ultra-lightweight block ciphers presented in CHES 2011 [18]. This algorithm provided sufficient security against related-key differential attacks. Two versions of 80-bit and 128-bit keys for this cipher were given in the implementation section, requiring 683 and 758 equivalent gates, respectively. These aspects, in addition to other notable features of this cipher, indicate the high performance of Piccolo cryptography over lightweight cryptographic algorithms.

In this study, for TI approach exploit first-order masking on the Piccolo algorithm in ISE Xilinx software using the VHDL programming language. Four shares are allocated for masking the input and output values, which evaluates the security of this scheme against the first-order DPA, according to the existing primitive concepts of TI.

The ideas of DPA and TI were discussed in the first chapter 1. The second chapter focuses on Piccolo's structure 2. The first-order masking theory for Piccolo, as well as the features required to achieve sufficient security against the power analysis probing model, are described in the third chapter 3. In the last chapter 4, the results of the TI hardware implementation are presented and compared to the Piccolo version which is not protected.

1.1 Differential Power Analysis

A power analysis attack is based on exploiting an encryption system's instantaneous power consumption traces, which are dependent on the intermediate values being processed or the operations performed on the algorithm. This dependency is due to a number of factors, such as the hardware implementation substrate, which is ultimately related to transistor manufacturing technology. Simple and differential power analysis (SPA-DPA) attacks are classified by how much the attacker understands the implemen-

tation details and how many power consumption traces he or she can acquire for a given encryption device [12].

The DPA attack process is performed in five steps [21]:

1. Choose an intermediate value
2. Power consumption measurements for various inputs
3. Key guessing
4. Modeling Power Consumption
5. Comparison of practical power measurement and simulation results

1.1.1 Resistance to Power Analysis Attacks

Approaches to counteracting power analysis have evolved in accordance with the development of various techniques for these attacks. Since the connection between the algorithm's intermediate values and the cryptographic device's encryption power traces are the causes of information leakages in a differential power analysis. hence, we must reduce this correspondence and connection as much as possible to enhance the implementation security of algorithms against the DPA. There are two approaches to this:

- From the attacker's viewpoint, hide the connections between intermediate values and the device's power trace [12].
- Randomizing intermediate values, In order to create a random power trace pattern, in this case, achieving real intermediate values is difficult for the attacker.

This research uses the second approach to implement a countermeasure against a power analysis attack.

1.1.2 Masking scheme

To the attacker, the masking scheme is providing incorrect information. Masking is based on the concept of secret sharing. The intermediate values of the encryption are

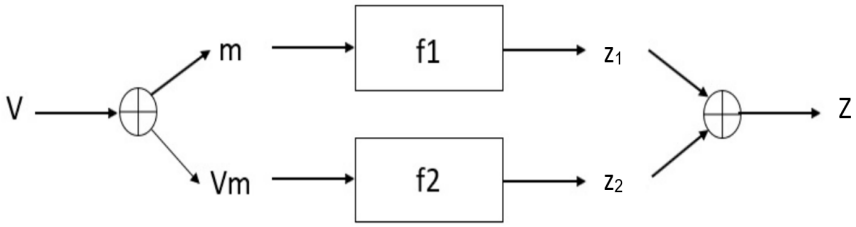

Figure 1.1: Masking scheme definition [19, 20]

directly related to the power consumption in this approach, but the intermediate values are not real and are chosen randomly. So each algorithm's intermediate value v is masked by a random value m. It is not sufficient to process the value of v using m, and a connection must be between the two to make the mask meaningful. As a result, the algorithm must process another value in addition to m, and the result of processing these values is exactly the same as the result of processing v. v_m is the description of this value. Equation 1.1 illustrates the connection between v and m.

$$v_m = v * m \tag{1.1}$$

* is a group operator that can be XOR, AND, OR, and other combinations. The XOR operator is used in Boolean masking. If the value of z was obtained by processing the value of v, we now have two random values that are variable shares of z after executing the algorithm, and the initial relation between z and its shares is the same (Figure 1.1).

Since knowing the value of one of the shares does not aid to retrieve the original variable according to the secret-sharing specification, if an attacker obtains one of the variable shares using DPA or CPA attacks, the original variable will remain inaccessible [20]. As a result, it is possible to define provable security against DPA.

First-order masking is a method of masking in which each variable is covered by a mask. This mentioned definition is resistant to a first-order DPA in which the attacker uses only one probe to measure the power trace of intermediate values.

d-order masking: If each variable is masked by a d mask (random value) and becomes $d + 1$ shares, it is known as d-order masking. The first d share of this $d + 1$

shares are chosen randomly, and the d + 1th share is created as follows:

$$X_1 \oplus X_1 \oplus ... \oplus X_{d+1} = X \tag{1.2}$$

d-order power analysis attack: A d-order DPA is performed if, instead of an intermediate value, the leakage information from "d" shares of intermediate value, and this leakage information is combined non-linearly.

Since the attacker does not face real intermediate values at any step of the algorithm and just has shares of that, a d-order masking scheme has proven to be secure against d-order power analysis attacks. If the attacker stores trace using a d-order or lower probe (d-probing model), d shares of the variable will be obtained from the maximum leakage information, which will not aid the attacker in retrieving the real value. A d-order masking strategy, is not always resistant to a d + 1-order or higher of DPA since it does not eliminate all of the leakage information but makes it harder to access exploitable leakage information [8].

1.1.3 Masking scheme features

In order to effectively perform, a masking scheme should always contain two features [19, 20]:

- Correctness. The correctness of a masking scheme is proven by the fact that combining the output shares of an algorithm provides the same variable if masking had not been employed.
- Uniformity. A statistical uniformity means that the probability of each member in distribution is equal to a fixed value. Uniformity in masking implementation refers to the uniformity of the random distribution used to choose the input variable masks, as well as the uniformity of the output shares at each level of the algorithm (outputs of linear and nonlinear functions).

1.2 Threshold Implementation

One of the major weaknesses in the implementation of masking is the occurrence of glitches in the hardware implemetation [13]. With this in mind, the d-Probing model had been studied for a masking scheme without considering the glitch effects. By the presence of a hardware glitch, instead of d shares, the attacker can obtain all $d + 1$ shares of an intermediate value using transient states or detecting a change in their power traces. As a result, in the scenario of a DPA attack on a d-order masking scheme, the attacker may get leakage information equal to a d-order attack or even lower due to the presence of glitches, rather than a $d + 1$-order attack. Since the glitch causes a significant modification in the edge of the signals, it impacts the algorithm's power consumption. In 2006, the TI scheme was introduced as a way to improve glitch effects by adding a third feature to the masking scheme. **Non-completeness** is a new feature that implies each of the shares allocated to a function does not have at least one set of input variable shares. In the worst-case scenario and in the presence of a glitch, the information leakage of each component is limited to one share of each input variable, and the attacker is unable to retrieve the real value in the absence of other shares. This approach was introduced in 2014 for the higher-order DPA [3]. As a result, the TI scheme has three main features: **correctness**, **uniformity**, and **non-completeness**, with the third property making it resistant to the d-order DPA in the presence of phenomena such as glitches.

The following equation 1.3 is the allocation of input-output shares according to the direct sharing approach [1], based on the order of masking implementation and the degree of function(in each algorithm). The number of input-output shares is denoted by s, and the degree of the function is denoted by t:

$$S_{in} \geq t \times d + 1 \qquad , \qquad S_{out} \geq \binom{S_{in}}{t} \tag{1.3}$$

The issue of uniformity of input and output components in various rounds of an algorithm is always a challenge for masking implementation, and with the addition of

the non-completeness feature, this issue becomes much more significant in TI. It is not always possible to find a function that can automatically establish all three of these criteria(correctness, uniformity, non-completeness). In other words, by defining a term for the minimum number of shares, it is typically possible to satisfy the correctness and non-completeness criteria. Achieving uniformity in input shares, on the other hand, is not difficult, but it does not guarantee consistent implementation at each function's output. Various solutions, such as re-masking [14], increasing the number of input shares [16], reducing the number of output shares [1], and so on, have been proposed to create functions that maintain uniformity in input and output shares. The first approach is used in this study.

1.3 Robust Probing Model

There are various implementation vulnerabilities that an attacker can use to perform the SCA against a masking scheme. such as a lack of composability (typically caused by an insufficient refreshing of the shares) can reduce the security order in the probing model, glitches (combinatorial recombinations), transition-based leakages in the circuit (memory recombinations), and potential couplings (routing recombinations). Even if security is ensured in the former cases, achieving the noise condition may be challenging (the physical noise of the operations exploited and the number of operations exploited) [9].

The **robust probing model** improves on the basic probing model to capture a larger variety of physical attacks and reduce the mentioned implementation weaknesses [9, 7].

In the (g, t, c)–robust d–probing concept, a gadget is secure if:(g: glitch , t: transition and c: coupling)

- the probes are extended with glitches (iff $g = 1$),
- the probes are extended with transitions (iff $t = 1$),
- the probes are extended with c-couplings (for an integer $c \geq 1$),

Hence, the classical d-probing model is thus the (0, 0, 0)–robust d–probing model.

Although non-completeness and uniformity are sufficient conditions for the composability of first-order glitch-resistant circuits, it is complicated for higher security orders. These two properties alone are not enough to reason about higher-order masked implementations in hardware and the addition of refreshing gadgets is needed for this purpose. Therefore, a threshold circuit is first-order robust probing secure if, for any component function in the circuit, its input values are jointly independent of the circuit's secrets. A threshold circuit composed of layers that include correctness, non-completeness, and uniformity features is first-order robust probing secure [7].

Chapter 2

Specification of Piccolo

2.1 Piccolo: An Ultra-Lightweight Blockcipher

Piccolo consists of a 64-bit plaintext block and key versions of 80-bit and 128-bit. Other features of this Ultra-Lightweight block cipher include repeatable round and the generalized Feistel structure as a round function. In a hardware implementation, this cipher is an optimum architecture (At the time of its design). Piccolo also requires only 60 equivalent gates to implement decryption along with encryption [18].

This cipher's round function (F) is a 16-by-16 mapping with two independent substitution boxes (S-boxes) and a Mixcolumn layer.Figure 2.1- 2.2 and Table 2.1 illustrate the round function architecture and the Piccolo substitution box mapping, respectively.

Also in equation 2.1 , the S-box in Piccolo can be represented using Algebraic Normal Form (ANF) [22](index 1 illustrate msb).

Table 2.1: 4-bit bijective S-box S in hexadecimal form [18]

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	E	4	B	2	3	8	0	9	1	A	7	F	6	C	5	D

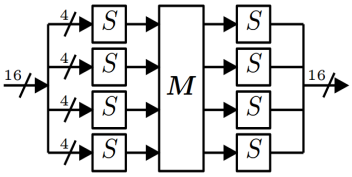

Figure 2.1: Round funcyion of Piccolo [18]

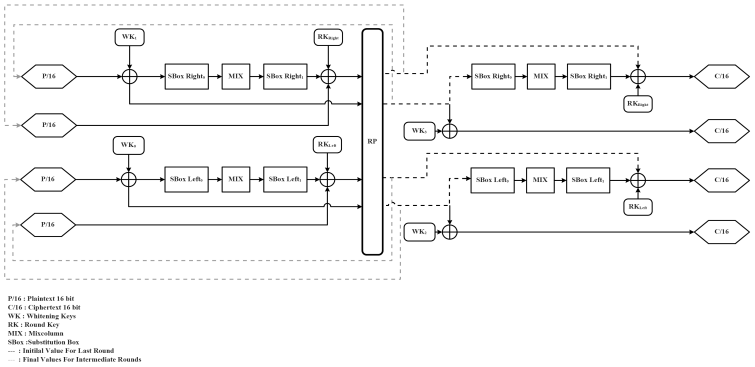

Figure 2.2: Architecture of serial-based Piccolo-128 encryption function

$$
\begin{aligned}
Output_1 &= 1 \oplus Input_1 \oplus Input_2 \oplus Input_4 \oplus Input_1x_2 \\
Output_2 &= 1 \oplus Input_1 \oplus Input_2 \oplus Input_3 \oplus Input_2Input_3 \\
Output_3 &= 1 \oplus Input_1 \oplus Input_4 \oplus Input_1Input_2 \oplus Input_1Input_3 \oplus \\
&\quad Input_2Input_3 \oplus Input_3Input_4 \oplus Input_1Input_2Input_3 \\
Output_4 &= Input_1 \oplus Input_2 \oplus Input_3 \oplus Input_1Input_3 \oplus Input_1Input_4 \oplus \\
&\quad Input_2Input_4 \oplus Input_3Input_4 \oplus Input_1Input_2Input_3 \oplus \\
&\quad Input_2Input_3Input_4
\end{aligned}
\tag{2.1}
$$

Chapter 3

First Order Masking of Piccolo

The polynomial degree of Piccolo's S-box is 3. The following equation 3.1 is generated for the minimal number of input and output shares based on the public connections mentioned in the threshold implementation for a gadget (encryption functions).

$$S_{in} \geq 3 \times (1) + 1 = 4 \qquad , \qquad S_{out} \geq \binom{4}{3} = 4 \tag{3.1}$$

Hence, the Piccolo substitution layers require at least four input and four output shares for the first order(d=1) threshold implementation. Four random shares of x, y, z, and w are considered for each S-box input bit. The symbol $Input$ in equation 3.2 represents the MSB to LSB bits of 4-bit input.

$$\begin{aligned} Input_1 &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \\ Input_2 &= y_1 \oplus y_2 \oplus y_3 \oplus y_4 \\ Input_3 &= z_1 \oplus z_2 \oplus z_3 \oplus z_4 \\ Input_4 &= w_1 \oplus w_2 \oplus w_3 \oplus w_4 \end{aligned} \tag{3.2}$$

The next section focuses on three main features for a first-order DPA-resistant threshold implementation based on substitution layer relations.

3.1 Correctness, Uniformity and Non-Completeness

According to .1, the ANF Piccolo equation inputs are extended. The first step in TI is the correctness, which states that the XOR of allocated shares must equal the value of the intermediate variable.

By sharing the substitution layer outputs, the non-completeness feature is formed for the TI, at least one share in various inputs must be absent for this feature to be confirmed(see Appendix .2).

To achieve the last feature, we evaluate the output truth table for all possible inputs for 4 bits mapping. For instance, Table 3.1 shows the MSB of output shares:

In table 3.1, there is no uniformity for a bit of output. This is also the case for other bits. We use **fresh randomness** masking to even out the outputs.

To construct the correctness feature, we choose three random shares and create the fourth share from the previous three shares(equation 3.3). Then we add fresh randomness to all of the Appendix .3 equations. As shown in Table 3.1, the output of the S-box will be uniform for all inputs , in this case, aslo we Repeated the same approach for the other outputs ($Output_{1,2}, Output_{2,2}, ..., Output_{4,4}$).

$$\begin{aligned} &FR_2 \xleftarrow{\$} \{0,1\}^4, \quad FR_3 \xleftarrow{\$} \{0,1\}^4, \quad FR_4 \xleftarrow{\$} \{0,1\}^4 \\ &FR_1 = FR_2 \oplus FR_3 \oplus FR_4 \end{aligned} \tag{3.3}$$

3.2 Robust Probing security

The concept of uniformity is composable. That is, if both functions are uniform, the combination of them is uniform. If the attacker probes a circuit element in the d-probing model that has the qualities of correctness, uniformity, and non-completeness, the maximum leakage information he obtains is less than the number of shares of the intermediate variables. The different round inputs will be independent of each other

Table 3.1: The number of occurrences of S-box outputs for all possible inputs(MSB)

$Inputs_{1,2,3,4}$	$Output_{1,1}, Output_{2,1}, Output_{3,1}, Output_{4,1}$															
msb→lsb	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	0	384	384	0	1152	0	0	128	1152	0	0	128	0	384	384	0
0001	1152	0	0	128	0	384	384	0	0	384	384	0	1152	0	0	128
0010	0	384	384	0	1152	0	0	128	1152	0	0	128	0	384	384	0
0011	1152	0	0	128	0	384	384	0	0	384	384	0	1152	0	0	128
0100	1152	0	0	128	0	384	384	0	0	384	384	0	1152	0	0	128
0101	0	384	384	0	1152	0	0	128	1152	0	0	128	0	384	384	0
0110	1152	0	0	128	0	384	384	0	0	384	384	0	1152	0	0	128
0111	0	384	384	0	1152	0	0	128	1152	0	0	128	0	384	384	0
1000	1152	0	0	128	0	384	384	0	0	384	384	0	1152	0	0	128
1001	0	384	384	0	1152	0	0	128	1152	0	0	128	0	384	384	0
1010	1152	0	0	128	0	384	384	0	0	384	384	0	1152	0	0	128
1011	0	384	384	0	1152	0	0	128	1152	0	0	128	0	384	384	0
1100	1152	0	0	128	0	384	384	0	0	384	384	0	1152	0	0	128
1101	0	384	384	0	1152	0	0	128	1152	0	0	128	0	384	384	0
1110	1152	0	0	128	0	384	384	0	0	384	384	0	1152	0	0	128
1111	0	384	384	0	1152	0	0	128	1152	0	0	128	0	384	384	0

Table 3.2: The number of occurrences of S-box outputs for all possible inputs(MSB) after unifying .3

$Inputs_{1,2,3,4}$	$Output_{1,1}, Output_{2,1}, Output_{3,1}, Output_{4,1}$															
msb→lsb	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	0	4096	4096	0	4096	0	0	4096	4096	0	0	4096	0	4096	4096	0
0001	4096	0	0	4096	0	4096	4096	0	0	4096	4096	0	4096	0	0	4096
0010	0	4096	4096	0	4096	0	0	4096	4096	0	0	4096	0	4096	4096	0
0011	4096	0	0	4096	0	4096	4096	0	0	4096	4096	0	4096	0	0	4096
0100	4096	0	0	4096	0	4096	4096	0	0	4096	4096	0	4096	0	0	4096
0101	0	4096	4096	0	4096	0	0	4096	4096	0	0	4096	0	4096	4096	0
0110	4096	0	0	4096	0	4096	4096	0	0	4096	4096	0	4096	0	0	4096
0111	0	4096	4096	0	4096	0	0	4096	4096	0	0	4096	0	4096	4096	0
1000	4096	0	0	4096	0	4096	4096	0	0	4096	4096	0	4096	0	0	4096
1001	0	4096	4096	0	4096	0	0	4096	4096	0	0	4096	0	4096	4096	0
1010	4096	0	0	4096	0	4096	4096	0	0	4096	4096	0	4096	0	0	4096
1011	0	4096	4096	0	4096	0	0	4096	4096	0	0	4096	0	4096	4096	0
1100	4096	0	0	4096	0	4096	4096	0	0	4096	4096	0	4096	0	0	4096
1101	0	4096	4096	0	4096	0	0	4096	4096	0	0	4096	0	4096	4096	0
1110	4096	0	0	4096	0	4096	4096	0	0	4096	4096	0	4096	0	0	4096
1111	0	4096	4096	0	4096	0	0	4096	4096	0	0	4096	0	4096	4096	0

when fresh randomness is provided. Hence, because the inputs enter the circuit components, the leak information is provided to the attacker independently, and retrieving the secret information will be unachievable without forming links between the intermediary variables [7]. Also, non-completeness and uniformity are sufficient conditions for the composability of first-order glitch-resistant circuits.

As a result, a TI scheme consisting of layers with the features of correctness, non-completeness, and uniformity, is resistant to the first-order robust probing.

Python programming is used to perform the computations in this section.

Chapter 4

Hardware Implementation

4.1 Piccolo No Protection

Appendix .4 is the unprotected implementation of Piccolo that required 69 registers, 293 multiplexers, and 128 XOR gates. It also has a maximum combinational path latency of 0.954 nanoseconds. In the next section, the TI Piccolo is implemented by allocating four shares to the nonlinear layer's inputs and outputs using the VHDL in Xilinx ISE.

4.2 TI Implementation of Piccolo-128

Based on the descriptions in Chapter 3, the Piccolo TI was designed in VHDL step by step. All linear components in this work were implemented in the same approach for all four shares. The inputs-outputs were shared according to the equations in Appendix .2 and .3 for the nonlinear component, which generally forms the substitution layer. The serial architecture for the TI Piccolo-128 with four shares is depicted in Figure 4.1.

Table 4.1 compares the Piccolo cipher in an unprotected situation with first-order TI in terms of implementation area and execution time. By allocating different shares than the unprotected version, the TI has a more cost implementation for the designer. This study did not consider TI optimization methods in various aspects. Also, there is

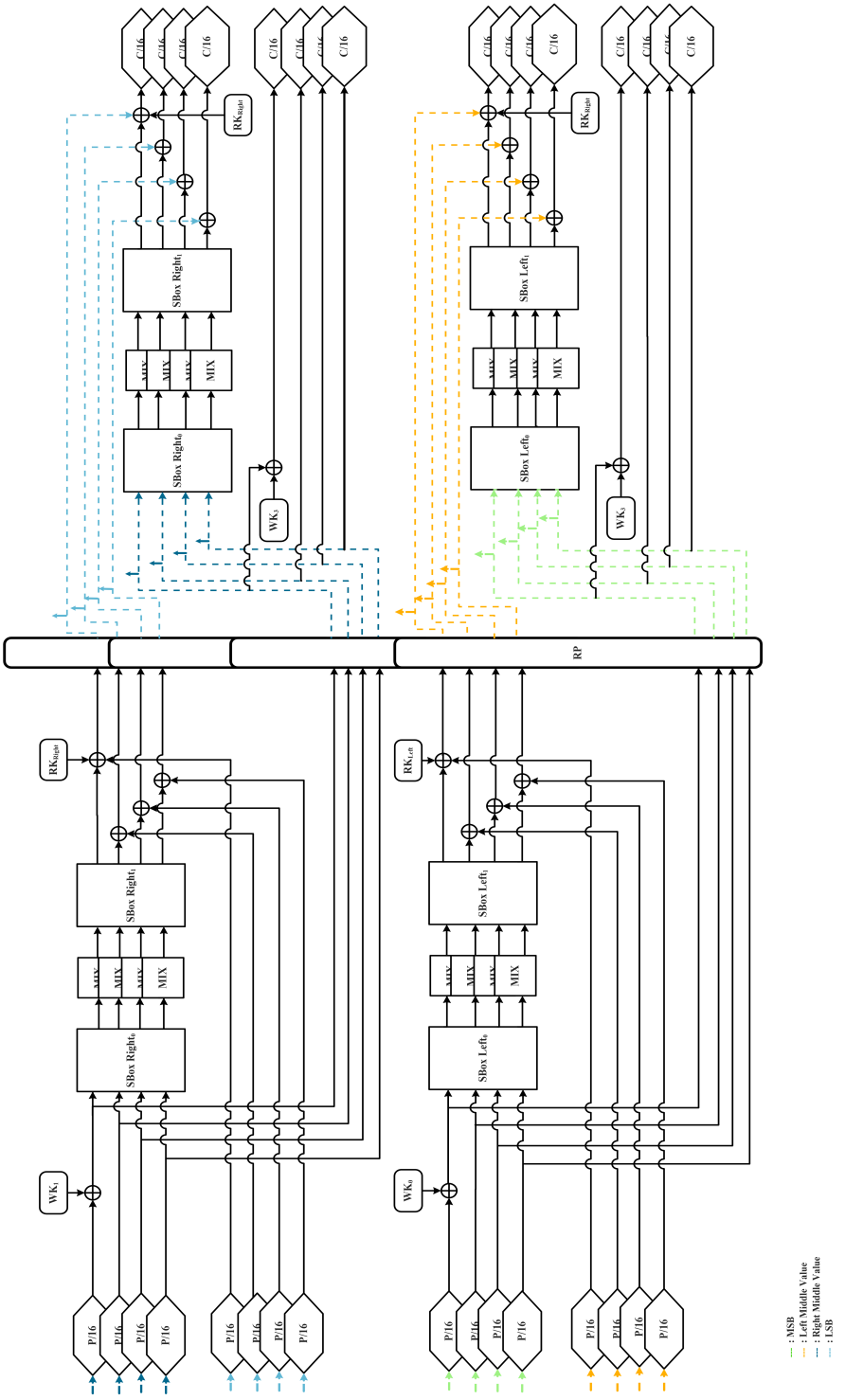

Figure 4.1: The architecture of serial-based TI Piccolo with 4 shares

Table 4.1: Comparison

Implementation factors	unprotected	first-order TI
Register	69	261
Multiplexer	293	677
Xor	128	688
MCPD	0.954 ns	5.679 ns

no claim to security for higher-order DPA, but it may be regarded as a future research topic.

Conclusion

Piccolo is an ultra-lightweight block cipher with 64-bit plaintext blocks and 128-bit and 64-bit versions of the key. Piccolo was presented at CHES in 2011. In this study, the Piccolo-128 first-order TI was performed. The nonlinear layer's inputs and outputs were getting masked with four shares. Python language evaluated the theoretical results, and VHDL was employed in the hardware implementation section. This implementation is resistant to first-order robust probing since it has three features: correctness, uniformity, and non-completeness. After allocating four shares, the cost of implementation, including area and processing time, has increased significantly; however, this is due to a lack of design optimization in input-output sharing, which might be an attractive topic for future research.

Bibliography

[1] Begül Bilgin. “Threshold implementations: as countermeasure against higher-order differential power analysis”. In: (2015).

[2] Begül Bilgin et al. “Trade-offs for threshold implementations illustrated on AES”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.7 (2015), pp. 1188–1200.

[3] Bilgin, Begül et al. “Higher-order threshold implementations”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2014, pp. 326–343.

[4] Dušan Božilov, Miroslav Knežević, and Ventzislav Nikov. “Threshold implementations of prince: the cost of physical security”. In: *NIST Lightweight Cryptography Workshop*. 2016.

[5] Suresh Chari et al. “Towards sound approaches to counteract power-analysis attacks”. In: *Annual International Cryptology Conference*. Springer. 1999, pp. 398–412.

[6] Thomas De Cnudde et al. “Higher-order glitch resistant implementation of the PRESENT S-box”. In: *International Conference on Cryptography and Information Security in the Balkans*. Springer. 2014, pp. 75–93.

[7] Siemen Dhooghe, Svetla Nikova, and Vincent Rijmen. “Threshold implementations in the robust probing model”. In: *Proceedings of ACM Workshop on Theory of Implementation Security Workshop*. 2019, pp. 30–37.

[8] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. “Making masking security proofs concrete”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2015, pp. 401–429.

[9] Sebastian Faust et al. “Composable masking schemes in the presence of physical defaults & the robust probing model”. In: (2018).

[10] Paul Kocher, Joshua Jaffe, and Benjamin Jun. “Differential power analysis”. In: *Annual international cryptology conference*. Springer. 1999, pp. 388–397.

[11] Paul C Kocher. “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems”. In: *Annual International Cryptology Conference*. Springer. 1996, pp. 104–113.

[12] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Vol. 31. Springer Science & Business Media, 2008.

[13] Stefan Mangard, Thomas Popp, and Berndt M Gammel. “Side-channel leakage of masked CMOS gates”. In: *Cryptographers’ Track at the RSA Conference*. Springer. 2005, pp. 351–365.

[14] Amir Moradi et al. “Pushing the limits: A very compact and a threshold implementation of AES”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2011, pp. 69–88.

[15] Nicolai Müller, Thorben Moos, and Amir Moradi. “Low-Latency Hardware Masking of PRINCE”. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2021, pp. 148–167.

[16] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. “Threshold implementations against side-channel attacks and glitches”. In: *International conference on information and communications security*. Springer. 2006, pp. 529–545.

[17] Axel Poschmann et al. “Side-channel resistant crypto for less than 2,300 GE”. In: *Journal of Cryptology* 24.2 (2011), pp. 322–345.

[18] Kyoji Shibutani et al. “Piccolo: an ultra-lightweight blockcipher”. In: *International workshop on cryptographic hardware and embedded systems*. Springer. 2011, pp. 342–357.

[19] Hadi Soleimany. *Advanced Cryptography Course*. Cyberspace Research Institute , Shahid Beheshti University, 2022. URL: `http://facultymembers.sbu.ac.ir/h_soleimany/advanced-cryptography-course`.

[20] Sara Zarei. *Provide the optimal approach for implementation the keccak function on hardware while ensuring security against high-order power analysis attacks.* Master Thesis, Cyberspace Research Institute , Shahid Beheshti University, 2020.

[21] Sara Zarei et al. “Low-latency Keccak at any arbitrary order”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), pp. 388–411.

[22] Fan Zhang et al. “Improved algebraic fault analysis: A case study on piccolo and applications to other lightweight block ciphers”. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2013, pp. 62–79.

Appendices

.1 Correctness

The fourth share of each input is calculated by XORing the first three shares and the plaintext. According to this point, the Correctness feature is confirmed at different levels of the TI.

$$Output_1 = 1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus y_1 \oplus y_2 \oplus y_3 \oplus y_4 \oplus w_1 \oplus w_2 \oplus w_3 \oplus w_4 (x_1y_1) \oplus (x_1y_2) \oplus (x_1y_3) \oplus (x_1y_4) \oplus (x_2y_1) \oplus (x_2y_2) \oplus (x_2y_3) \oplus (x_2y_4) \oplus (x_3y_1) \oplus (x_3y_2) \oplus (x_3y_3) \oplus (x_3y_4) \oplus (x_4y_1) \oplus (x_4y_2) \oplus (x_4y_3) \oplus (x_4y_4)$$

$$Output_2 = 1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus y_1 \oplus y_2 \oplus y_3 \oplus y_4 \oplus z_1 \oplus z_2 \oplus z_3 \oplus z_4 (y_1z_1) \oplus (y_1z_2) \oplus (y_1z_3) \oplus (y_1z_4) \oplus (y_2z_1) \oplus (y_2z_2) \oplus (y_2z_3) \oplus (y_2z_4) \oplus (y_3z_1) \oplus (y_3z_2) \oplus (y_3z_3) \oplus (y_3z_4) \oplus (y_4z_1) \oplus (y_4z_2) \oplus (y_4z_3) \oplus (y_4z_4)$$

$$Output_3 = 1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus w_1 \oplus w_2 \oplus w_3 \oplus w_4 \oplus$$
$$(x_1y_1) \oplus (x_1y_2) \oplus (x_1y_3) \oplus (x_1y_4) \oplus (x_2y_1) \oplus (x_2y_2) \oplus (x_2y_3) \oplus (x_2y_4) \oplus (x_3y_1) \oplus (x_3y_2) \oplus (x_3y_3) \oplus (x_3y_4) \oplus (x_4y_1) \oplus (x_4y_2) \oplus (x_4y_3) \oplus (x_4y_4) \oplus (x_1z_1) \oplus (x_1z_2) \oplus (x_1z_3) \oplus (x_1z_4) \oplus (x_2z_1) \oplus (x_2z_2) \oplus (x_2z_3) \oplus (x_2z_4) \oplus (x_3z_1) \oplus (x_3z_2) \oplus (x_3z_3) \oplus (x_3z_4) \oplus (x_4z_1) \oplus (x_4z_2) \oplus (x_4z_3) \oplus (x_4z_4) \oplus (y_1z_1) \oplus (y_1z_2) \oplus (y_1z_3) \oplus (y_1z_4) \oplus (y_2z_1) \oplus (y_2z_2) \oplus (y_2z_3) \oplus (y_2z_4) \oplus (y_3z_1) \oplus (y_3z_2) \oplus (y_3z_3) \oplus (y_3z_4) \oplus (y_4z_1) \oplus (y_4z_2) \oplus (y_4z_3) \oplus (y_4z_4) \oplus$$
$$(z_1w_1) \oplus (z_1w_2) \oplus (z_1w_3) \oplus (z_1w_4) \oplus (z_2w_1) \oplus (z_2w_2) \oplus (z_2w_3) \oplus (z_2w_4) \oplus (z_3w_1) \oplus (z_3w_2) \oplus (z_3w_3) \oplus (z_3w_4) \oplus (z_4w_1) \oplus (z_4w_2) \oplus (z_4w_3) \oplus (z_4w_4) \oplus$$
$$(x_1y_1z_1) \oplus (x_1y_2z_1) \oplus (x_1y_3z_1) \oplus (x_1y_4z_1) \oplus (x_2y_1z_1) \oplus (x_2y_2z_1) \oplus (x_2y_3z_1) \oplus (x_2y_4z_1) \oplus (x_3y_1z_1) \oplus (x_3y_2z_1) \oplus (x_3y_3z_1) \oplus (x_3y_4z_1) \oplus (x_4y_1z_1) \oplus (x_4y_2z_1) \oplus (x_4y_3z_1) \oplus (x_4y_4z_1) \oplus (x_1y_1z_2) \oplus (x_1y_2z_2) \oplus (x_1y_3z_2) \oplus (x_1y_4z_2) \oplus (x_2y_1z_2) \oplus (x_2y_2z_2) \oplus (x_2y_3z_2) \oplus (x_2y_4z_2) \oplus (x_3y_1z_2) \oplus (x_3y_2z_2) \oplus (x_3y_3z_2) \oplus (x_3y_4z_2) \oplus (x_4y_1z_2) \oplus (x_4y_2z_2) \oplus (x_4y_3z_2) \oplus (x_4y_4z_2) \oplus (x_1y_1z_3) \oplus (x_1y_2z_3) \oplus (x_1y_3z_3) \oplus (x_1y_4z_3) \oplus (x_2y_1z_3) \oplus (x_2y_2z_3) \oplus (x_2y_3z_3) \oplus (x_2y_4z_3) \oplus (x_3y_1z_3) \oplus (x_3y_2z_3) \oplus (x_3y_3z_3) \oplus (x_3y_4z_3) \oplus (x_4y_1z_3) \oplus (x_4y_2z_3) \oplus (x_4y_3z_3) \oplus (x_4y_4z_3) \oplus (x_1y_1z_4) \oplus (x_1y_2z_4) \oplus (x_1y_3z_4) \oplus (x_1y_4z_4) \oplus (x_2y_1z_4) \oplus (x_2y_2z_4) \oplus (x_2y_3z_4) \oplus (x_2y_4z_4) \oplus (x_3y_1z_4) \oplus (x_3y_2z_4) \oplus (x_3y_3z_4) \oplus (x_3y_4z_4) \oplus (x_4y_1z_4) \oplus (x_4y_2z_4) \oplus (x_4y_3z_4) \oplus (x_4y_4z_4)$$

$$Output_4 = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus y_1 \oplus y_2 \oplus y_3 \oplus y_4 \oplus z_1 \oplus z_2 \oplus z_3 \oplus z_4$$
$$(x_1z_1) \oplus (x_1z_2) \oplus (x_1z_3) \oplus (x_1z_4) \oplus (x_2z_1) \oplus (x_2z_2) \oplus (x_2z_3) \oplus (x_2z_4) \oplus (x_3z_1) \oplus (x_3z_2) \oplus (x_3z_3) \oplus (x_3z_4) \oplus (x_4z_1) \oplus (x_4z_2) \oplus (x_4z_3) \oplus (x_4z_4) \oplus (x_1w_1) \oplus (x_1w_2) \oplus (x_1w_3) \oplus (x_1w_4) \oplus$$

$(x_2w_1) \oplus (x_2w_2) \oplus (x_2w_3) \oplus (x_2w_4) \oplus (x_3w_1) \oplus (x_3w_2) \oplus (x_3w_3) \oplus (x_3w_4) \oplus (x_4w_1) \oplus$
$(x_4w_2) \oplus (x_4w_3) \oplus (x_4w_4) \oplus (y_1w_1) \oplus (y_1w_2) \oplus (y_1w_3) \oplus (y_1w_4) \oplus (y_2w_1) \oplus (y_2w_2) \oplus$
$(y_2w_3) \oplus (y_2w_4) \oplus (y_3w_1) \oplus (y_3w_2) \oplus (y_3w_3) \oplus (y_3w_4) \oplus (y_4w_1) \oplus (y_4w_2) \oplus (y_4w_3) \oplus$
$(y_4w_4) \oplus (z_1w_1) \oplus (z_1w_2) \oplus (z_1w_3) \oplus (z_1w_4) \oplus (z_2w_1) \oplus (z_2w_2) \oplus (z_2w_3) \oplus (z_2w_4) \oplus$
$(z_3w_1) \oplus (z_3w_2) \oplus (z_3w_3) \oplus (z_3w_4) \oplus (z_4w_1) \oplus (z_4w_2) \oplus (z_4w_3) \oplus (z_4w_4) \oplus (x_1y_1z_1) \oplus$
$(x_1y_2z_1) \oplus (x_1y_3z_1) \oplus (x_1y_4z_1) \oplus (x_2y_1z_1) \oplus (x_2y_2z_1) \oplus (x_2y_3z_1) \oplus (x_2y_4z_1) \oplus (x_3y_1z_1) \oplus$
$(x_3y_2z_1) \oplus (x_3y_3z_1) \oplus (x_3y_4z_1) \oplus (x_4y_1z_1) \oplus (x_4y_2z_1) \oplus (x_4y_3z_1) \oplus (x_4y_4z_1) \oplus (x_1y_1z_2) \oplus$
$(x_1y_2z_2) \oplus (x_1y_3z_2) \oplus (x_1y_4z_2) \oplus (x_2y_1z_2) \oplus (x_2y_2z_2) \oplus (x_2y_3z_2) \oplus (x_2y_4z_2) \oplus (x_3y_1z_2) \oplus$
$(x_3y_2z_2) \oplus (x_3y_3z_2) \oplus (x_3y_4z_2) \oplus (x_4y_1z_2) \oplus (x_4y_2z_2) \oplus (x_4y_3z_2) \oplus (x_4y_4z_2) \oplus (x_1y_1z_3) \oplus$
$(x_1y_2z_3) \oplus (x_1y_3z_3) \oplus (x_1y_4z_3) \oplus (x_2y_1z_3) \oplus (x_2y_2z_3) \oplus (x_2y_3z_3) \oplus (x_2y_4z_3) \oplus (x_3y_1z_3) \oplus$
$(x_3y_2z_3) \oplus (x_3y_3z_3) \oplus (x_3y_4z_3) \oplus (x_4y_1z_3) \oplus (x_4y_2z_3) \oplus (x_4y_3z_3) \oplus (x_4y_4z_3) \oplus (x_1y_1z_4) \oplus$
$(x_1y_2z_4) \oplus (x_1y_3z_4) \oplus (x_1y_4z_4) \oplus (x_2y_1z_4) \oplus (x_2y_2z_4) \oplus (x_2y_3z_4) \oplus (x_2y_4z_4) \oplus (x_3y_1z_4) \oplus$
$(x_3y_2z_4) \oplus (x_3y_3z_4) \oplus (x_3y_4z_4) \oplus (x_4y_1z_4) \oplus (x_4y_2z_4) \oplus (x_4y_3z_4) \oplus (x_4y_4z_4) \oplus (y_1z_1w_1) \oplus$
$(y_1z_2w_1) \oplus (y_1z_3w_1) \oplus (y_1z_4w_1) \oplus (y_2z_1w_1) \oplus (y_2z_2w_1) \oplus (y_2z_3w_1) \oplus (y_2z_4w_1) \oplus$
$(y_3z_1w_1) \oplus (y_3z_2w_1) \oplus (y_3z_3w_1) \oplus (y_3z_4w_1) \oplus (y_4z_1w_1) \oplus (y_4z_2w_1) \oplus (y_4z_3w_1) \oplus$
$(y_4z_4w_1) \oplus (y_1z_1w_2) \oplus (y_1z_2w_2) \oplus (y_1z_3w_2) \oplus (y_1z_4w_2) \oplus (y_2z_1w_2) \oplus (y_2z_2w_2) \oplus$
$(y_2z_3w_2) \oplus (y_2z_4w_2) \oplus (y_3z_1w_2) \oplus (y_3z_2w_2) \oplus (y_3z_3w_2) \oplus (y_3z_4w_2) \oplus (y_4z_1w_2) \oplus$
$(y_4z_2w_2) \oplus (y_4z_3w_2) \oplus (y_4z_4w_2) \oplus (y_1z_1w_3) \oplus (y_1z_2w_3) \oplus (y_1z_3w_3) \oplus (y_1z_4w_3) \oplus$
$(y_2z_1w_3) \oplus (y_2z_2w_3) \oplus (y_2z_3w_3) \oplus (y_2z_4w_3) \oplus (y_3z_1w_3) \oplus (y_3z_2w_3) \oplus (y_3z_3w_3) \oplus$
$(y_3z_4w_3) \oplus (y_4z_1w_3) \oplus (y_4z_2w_3) \oplus (y_4z_3w_3) \oplus (y_4z_4w_3) \oplus (y_1z_1w_4) \oplus (y_1z_2w_4) \oplus$
$(y_1z_3w_4) \oplus (y_1z_4w_4) \oplus (y_2z_1w_4) \oplus (y_2z_2w_4) \oplus (y_2z_3w_4) \oplus (y_2z_4w_4) \oplus (y_3z_1w_4) \oplus$
$(y_3z_2w_4) \oplus (y_3z_3w_4) \oplus (y_3z_4w_4) \oplus (y_4z_1w_4) \oplus (y_4z_2w_4) \oplus (y_4z_3w_4) \oplus (y_4z_4w_4)$

.2 Non-completeness

To share the outputs, we use the $Output_{s,b}$, where s is the share number and b is the bit value of the outputs.

For MSB of output we have the following four shares:

$Output_{1,1} = 1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus y_2 \oplus y_3 \oplus y_4 \oplus w_2 \oplus w_3 \oplus w_4(x_2y_2) \oplus (x_2y_3) \oplus (x_2y_4) \oplus$
$(x_3y_2) \oplus (x_3y_3) \oplus (x_3y_4) \oplus (x_4y_2) \oplus (x_4y_3) \oplus (x_4y_4)$

$Output_{2,1} = x_1 \oplus y_1 \oplus w_1 \oplus (x_1y_1) \oplus (x_1y_3) \oplus (x_1y_4) \oplus (x_3y_1) \oplus (x_4y_1) \oplus$

$Output_{3,1} = (x_1y_2)$

$Output_{4,1} = (x_2y_1)$

For the second bit we have the following four shares output:

$Output_{1,2} = 1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus y_2 \oplus y_3 \oplus y_4 \oplus z_2 \oplus z_3 \oplus z_4(y_2z_2) \oplus (y_2z_3) \oplus (y_2z_4) \oplus$
$(y_3z_2) \oplus (y_3z_3) \oplus (y_3z_4) \oplus (y_4z_2) \oplus (y_4z_3) \oplus (y_4z_4)$

$Output_{2,2} = x_1 \oplus y_1 \oplus z_1(y_1z_1) \oplus (y_1z_3) \oplus (y_1z_4) \oplus (y_3z_1) \oplus (y_4z_1)$

$Output_{3,2} = (y_1z_2)$

$Output_{4,2} = (y_2z_1)$

For the third bit, we have the following four shares:

$Output_{1,3} = 1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus w_2 \oplus w_3 \oplus w_4 \oplus (x_2y_2) \oplus (x_2y_3) \oplus (x_2y_4) \oplus (x_3y_2) \oplus$
$(x_3y_3) \oplus (x_3y_4) \oplus (x_4y_2) \oplus (x_4y_3) \oplus (x_4y_4) \oplus (x_2z_2) \oplus (x_2z_3) \oplus (x_2z_4) \oplus (x_3z_2) \oplus (x_3z_3) \oplus$
$(x_3z_4) \oplus (x_4z_2) \oplus (x_4z_3) \oplus (x_4z_4) \oplus (y_2z_2) \oplus (y_2z_3) \oplus (y_2z_4) \oplus (y_3z_2) \oplus (y_3z_3) \oplus (y_3z_4) \oplus$
$(y_4z_2) \oplus (y_4z_3) \oplus (y_4z_4) \oplus (z_2w_2) \oplus (z_2w_3) \oplus (z_2w_4) \oplus (z_3w_2) \oplus (z_3w_3) \oplus (z_3w_4) \oplus$
$(z_4w_2) \oplus (z_4w_3) \oplus (z_4w_4) \oplus (x_2y_2z_2) \oplus (x_2y_3z_2) \oplus (x_2y_4z_2) \oplus (x_3y_2z_2) \oplus (x_3y_3z_2) \oplus$
$(x_3y_4z_2) \oplus (x_4y_2z_2) \oplus (x_4y_3z_2) \oplus (x_4y_4z_2) \oplus (x_2y_2z_3) \oplus (x_2y_3z_3) \oplus (x_2y_4z_3) \oplus (x_3y_2z_3) \oplus$
$(x_3y_3z_3) \oplus (x_3y_4z_3) \oplus (x_4y_2z_3) \oplus (x_4y_3z_3) \oplus (x_4y_4z_3) \oplus (x_2y_2z_4) \oplus (x_2y_3z_4) \oplus (x_2y_4z_4) \oplus$
$(x_3y_2z_4) \oplus (x_3y_3z_4) \oplus (x_3y_4z_4) \oplus (x_4y_2z_4) \oplus (x_4y_3z_4) \oplus (x_4y_4z_4)$

$Output_{2,3} = x_1 \oplus w_1 \oplus (x_1y_1) \oplus (x_1y_3) \oplus (x_1y_4) \oplus (x_3y_1) \oplus (x_4y_1) \oplus (x_1z_1) \oplus (x_1z_3) \oplus$
$(x_1z_4) \oplus (x_3z_1) \oplus (x_4z_1) \oplus (y_1z_1) \oplus (y_1z_3) \oplus (y_1z_4) \oplus (y_3z_1) \oplus (y_4z_1) \oplus (z_1w_1) \oplus$
$(z_1w_3) \oplus (z_1w_4) \oplus (z_3w_1) \oplus (z_4w_1) \oplus (x_1y_1z_1) \oplus (x_1y_3z_1) \oplus (x_1y_4z_1) \oplus (x_3y_1z_1) \oplus$
$(x_3y_3z_1) \oplus (x_3y_4z_1) \oplus (x_4y_1z_1) \oplus (x_4y_3z_1) \oplus (x_4y_4z_1) \oplus (x_1y_1z_3) \oplus (x_1y_3z_3) \oplus (x_1y_4z_3) \oplus$
$(x_3y_1z_3) \oplus (x_4y_1z_3) \oplus (x_1y_1z_4) \oplus (x_1y_3z_4) \oplus (x_1y_4z_4) \oplus (x_3y_1z_4) \oplus (x_4y_1z_4)$

$Output_{3,3} = (x_1y_2) \oplus (x_2y_1) \oplus (x_1z_2) \oplus (x_2z_1) \oplus (y_1z_2) \oplus (y_2z_1) \oplus (z_1w_2) \oplus (z_2w_1) \oplus$
$(x_1y_2z_1) \oplus (x_2y_1z_1) \oplus (x_2y_2z_1) \oplus (x_2y_4z_1) \oplus (x_4y_2z_1) \oplus (x_1y_1z_2) \oplus (x_1y_2z_2) \oplus (x_1y_4z_2) \oplus$
$(x_2y_1z_2) \oplus (x_4y_1z_2) \oplus (x_1y_2z_4) \oplus (x_2y_1z_4)$

$Output_{4,3} = (x_2y_3z_1) \oplus (x_3y_2z_1) \oplus (x_1y_3z_2) \oplus (x_3y_1z_2) \oplus (x_1y_2z_3) \oplus (x_2y_1z_3)$

For LSB of output we have the following four shares:

$$Output_{1,4} = x_2 \oplus x_3 \oplus x_4 \oplus y_2 \oplus y_3 \oplus y_4 \oplus z_2 \oplus z_3 \oplus z_4 \oplus (x_2z_2) \oplus (x_2z_3) \oplus (x_2z_4) \oplus (x_3z_2) \oplus (x_3z_3) \oplus (x_3z_4) \oplus (x_4z_2) \oplus (x_4z_3) \oplus (x_4z_4) \oplus (x_2w_2) \oplus (x_2w_3) \oplus (x_2w_4) \oplus (x_3w_2) \oplus (x_3w_3) \oplus (x_3w_4) \oplus (x_4w_2) \oplus (x_4w_3) \oplus (x_4w_4) \oplus (y_2w_2) \oplus (y_2w_3) \oplus (y_2w_4) \oplus (y_3w_2) \oplus (y_3w_3) \oplus (y_3w_4) \oplus (y_4w_2) \oplus (y_4w_3) \oplus (y_4w_4) \oplus (z_2w_2) \oplus (z_2w_3) \oplus (z_2w_4) \oplus (z_3w_2) \oplus (z_3w_3) \oplus (z_3w_4) \oplus (z_4w_2) \oplus (z_4w_3) \oplus (z_4w_4) \oplus (x_2y_2z_2) \oplus (x_2y_3z_2) \oplus (x_2y_4z_2) \oplus (x_3y_2z_2) \oplus (x_3y_3z_2) \oplus (x_3y_4z_2) \oplus (x_4y_2z_2) \oplus (x_4y_3z_2) \oplus (x_4y_4z_2) \oplus (x_2y_2z_3) \oplus (x_2y_3z_3) \oplus (x_2y_4z_3) \oplus (x_3y_2z_3) \oplus (x_3y_3z_3) \oplus (x_3y_4z_3) \oplus (x_4y_2z_3) \oplus (x_4y_3z_3) \oplus (x_4y_4z_3) \oplus (x_2y_2z_4) \oplus (x_2y_3z_4) \oplus (x_2y_4z_4) \oplus (x_3y_2z_4) \oplus (x_3y_3z_4) \oplus (x_3y_4z_4) \oplus (x_4y_2z_4) \oplus (x_4y_3z_4) \oplus (x_4y_4z_4) \oplus (y_2z_2w_2) \oplus (y_2z_3w_2) \oplus (y_2z_4w_2) \oplus (y_3z_2w_2) \oplus (y_3z_3w_2) \oplus (y_3z_4w_2) \oplus (y_4z_2w_2) \oplus (y_4z_3w_2) \oplus (y_4z_4w_2) \oplus (y_2z_2w_3) \oplus (y_2z_3w_3) \oplus (y_2z_4w_3) \oplus (y_3z_2w_3) \oplus (y_3z_3w_3) \oplus (y_3z_4w_3) \oplus (y_4z_2w_3) \oplus (y_4z_3w_3) \oplus (y_4z_4w_3) \oplus (y_2z_2w_4) \oplus (y_2z_3w_4) \oplus (y_2z_4w_4) \oplus (y_3z_2w_4) \oplus (y_3z_3w_4) \oplus (y_3z_4w_4) \oplus (y_4z_2w_4) \oplus (y_4z_3w_4) \oplus (y_4z_4w_4)$$

$$Output_{2,4} = x_1 \oplus y_1 \oplus z_1 \oplus (x_1z_1) \oplus (x_1z_3) \oplus (x_1z_4) \oplus (x_3z_1) \oplus (x_4z_1) \oplus (x_1w_1) \oplus (x_1w_3) \oplus (x_1w_4) \oplus (x_3w_1) \oplus (x_4w_1) \oplus (y_1w_1) \oplus (y_1w_3) \oplus (y_1w_4) \oplus (y_3w_1) \oplus (y_4w_1) \oplus (z_1w_1) \oplus (z_1w_3) \oplus (z_1w_4) \oplus (z_3w_1) \oplus (z_4w_1) \oplus (x_1y_1z_1) \oplus (x_1y_3z_1) \oplus (x_1y_4z_1) \oplus (x_3y_1z_1) \oplus (x_3y_3z_1) \oplus (x_3y_4z_1) \oplus (x_4y_1z_1) \oplus (x_4y_3z_1) \oplus (x_4y_4z_1) \oplus (x_1y_1z_3) \oplus (x_1y_3z_3) \oplus (x_1y_4z_3) \oplus (x_3y_1z_3) \oplus (x_4y_1z_3) \oplus (x_1y_1z_4) \oplus (x_1y_3z_4) \oplus (x_1y_4z_4) \oplus (x_3y_1z_4) \oplus (x_4y_1z_4) \oplus (y_1z_1w_1) \oplus (y_1z_3w_1) \oplus (y_1z_4w_1) \oplus (y_3z_1w_1) \oplus (y_3z_3w_1) \oplus (y_3z_4w_1) \oplus (y_4z_1w_1) \oplus (y_4z_3w_1) \oplus (y_4z_4w_1) \oplus (y_1z_1w_3) \oplus (y_1z_3w_3) \oplus (y_1z_4w_3) \oplus (y_3z_1w_3) \oplus (y_4z_1w_3) \oplus (y_1z_1w_4) \oplus (y_1z_3w_4) \oplus (y_1z_4w_4) \oplus (y_3z_1w_4) \oplus (y_4z_1w_4)$$

$$Output_{3,4} = (x_1z_2) \oplus (x_2z_1) \oplus (x_1w_2) \oplus (x_2w_1) \oplus (y_1w_2) \oplus (y_2w_1) \oplus (z_1w_2) \oplus (z_2w_1) \oplus (x_1y_2z_1) \oplus (x_2y_1z_1) \oplus (x_2y_2z_1) \oplus (x_2y_4z_1) \oplus (x_4y_2z_1) \oplus (x_1y_1z_2) \oplus (x_1y_2z_2) \oplus (x_1y_4z_2) \oplus (x_2y_1z_2) \oplus (x_4y_1z_2) \oplus (x_1y_2z_4) \oplus (x_2y_1z_4) \oplus (y_1z_2w_1) \oplus (y_2z_1w_1) \oplus (y_2z_2w_1) \oplus (y_2z_4w_1) \oplus (y_4z_2w_1) \oplus (y_1z_1w_2) \oplus (y_1z_2w_2) \oplus (y_1z_4w_2) \oplus (y_2z_1w_2) \oplus (y_4z_1w_2) \oplus (y_1z_2w_4) \oplus (y_2z_1w_4)$$

$$Output_{4,4} = (x_2y_3z_1) \oplus (x_3y_2z_1) \oplus (x_1y_3z_2) \oplus (x_3y_1z_2) \oplus (x_1y_2z_3) \oplus (x_2y_1z_3) \oplus (y_2z_3w_1) \oplus (y_3z_2w_1) \oplus (y_1z_3w_2) \oplus (y_3z_1w_2) \oplus (y_1z_2w_3) \oplus (y_2z_1w_3)$$

.3 Uniformity

We utilize the fresh randomness equation 3.3 to satisfy the uniformity:

$Output_{1,1} \Leftarrow Output_{1,1} \oplus FR_{1,1}$

$Output_{2,1} \Leftarrow Output_{2,1} \oplus FR_{2,1}$

$Output_{3,1} \Leftarrow Output_{3,1} \oplus FR_{3,1}$

$Output_{4,1} \Leftarrow Output_{4,1} \oplus FR_{4,1}$

$Output_{1,2} \Leftarrow Output_{1,2} \oplus FR_{1,2}$

$Output_{2,2} \Leftarrow Output_{2,2} \oplus FR_{2,2}$

$Output_{3,2} \Leftarrow Output_{3,2} \oplus FR_{3,2}$

$Output_{4,2} \Leftarrow Output_{4,2} \oplus FR_{4,2}$

$Output_{1,3} \Leftarrow Output_{1,3} \oplus FR_{1,3}$

$Output_{2,3} \Leftarrow Output_{2,3} \oplus FR_{2,3}$

$Output_{3,3} \Leftarrow Output_{3,3} \oplus FR_{3,3}$

$Output_{4,3} \Leftarrow Output_{4,3} \oplus FR_{4,3}$

$Output_{1,4} \Leftarrow Output_{1,4} \oplus FR_{1,4}$

$Output_{2,4} \Leftarrow Output_{2,4} \oplus FR_{2,4}$

$Output_{3,4} \Leftarrow Output_{3,4} \oplus FR_{3,4}$

$Output_{4,4} \Leftarrow Output_{4,4} \oplus FR_{4,4}$

.4 Piccolo-128 unprotected

`https://github.com/emsec/ImpeccableCircuits/tree/master/Piccolo`

.5 Test Vector

A

$Plaintext$ = 0123456789abcdef

key = 00112233445566778899aabbccddeeff

$Input_1$ = 56752289bbefecd8

$Input_2$ = 5066448810aacc45

$Input_3$ = 5274efaaff441236

$Input_4$ = 5544ccccddaaff44

$Output_1$ = 4ff245deb58984b1

$Output_2$ = 3538ab70064a8ed9

$Output_3$ = 9e4bbb9dd86873c8

$Output_4$ = cc4422bbffccaa55

$Ciphertext$ = 5ec42cea657b89ff

B

$Plaintext$ = aaaaaaaaaaaaaaaa

key = 4455aaccbbff115544ff123456aaddff

$Input_1$ = eeeeeeeeeeeeeeee

$Input_2$ = 0123456789abcdef

$Input_3$ = fedcba9876543210

$Input_4$ = bbbbbbbbbbbbbbbb

$Output_1$ = ab816da4f17b781e

$Output_2$ = 62628a20626202a8

$Output_3$ = 9d9d8a209d9d0da8

$Output_4$ = bbbb4444bbbb4444

$Ciphertext$ = 580ff43f6e31f6fe