PROCEEDINGS OF SCIENCE

# FPGA Optimal Implementation of PRINCE against Power Analysis Attacks

**Yi Zou** [1]
*College of Computer Science and Technology, Hengyang Normal University*
*Hunan Provincial Key Laboratory of Intelligent Information Processing and Application*
*Hengyang, 421002, China*
*E-mail:* 51674074@qq.com

**Lang Li**[2]
*College of Computer Science and Technology, Hengyang Normal University*
*Hengyang, 421002, China*
*E-mail:* lilang911@126.com

The algorithm PRINCE proposed in ASIACRYPT 2012 is a lightweight cipher, currently suitable for the implementation on RFID Smart Card of the IoT. It's studied to achieve the optimal implementation of PRINCE encryption algorithm, and to add a fixed random mask to enhance the anti power attacking ability of PRINCE. In this paper, we constructed the same operations into a module called PrinceRound, and used counter to control the PrinceRound module, while the same round operation runs repeatedly. As a result, it can save registers and effectively reduce the amount of computation. Furthermore, we used a special method by adding the random mask to S-box. The experimental results show that the optimized PRINCE occupies area 9.8% fewer than original PRINCE on FPGA and the encryption with a fixed random mask can run correctly. Last but not least, our research is the first one about the implementation of PRINCE algorithm on FPGA, and can serve as a reference for further application of IOT encryption.

*ISCC 2017*
*16-17 December 2017*
*Guangzhou, China*

[1]Yi Zou(1983-),M.S.,Lecturer, Her research interests include embedded system and information security.

[2]This research is supported by the Scientific Research Fund of Hunan Provincial Education Department with Grant(15C0203), Industry university research project of Hengyang Normal University (No. 14CXYY02), the National Natural Science Foundation of China under Grant (No.61572174), the Science and Technology Plan Project of Hunan Province (No.2016TP1020) and Young Backbone Teacher of HengYang Normal University.





PoS(ISCC 2017)039

## 1.Introduction

With the rapid development of the Internet of Things (IoT), great changes have taken place in all aspects of people's lives in recent years. Many lightweight block ciphers have been proposed to provide cipher blocks for resource constrained devices such as RFID tags. Among the best studied ciphers are PRESENT, KATAN,LED and PRINT Cipher [1-4].

PRINCE is a novel lightweight block cipher which was proposed by Borgho in 2012[5]. It is optimized with respect to latency when implemented in hardware. This is the first lightweight block cipher which gives priority to latency with block length 64 bits and the key 128 bits. There are few research papers about PRINCE cryptographic algorithms but there is no research paper about optimization of the PRINCE cipher algorithm and resisting power analysis.

It is very important to reduce area resources when the cipher is implemented on resource constrained chips. At the same time, in order to obtain the area optimization, the anti attack ability of the cipher is correspondingly declining. So it is a very meaningful to find out how to make the lightweight cipher more secure, occupy less resource and run faster.

This paper is organized as follows: Section 2 briefly describes the PRINCE block cipher; Section 3 elaborates on area optimization of PRINCE; in Section 4, resisting power analysis for PRINCE encryption algorithm is proposed; Section 5 discusses the experimental results, and finally the conclusion is presented in the last section.

## 2.Description of PRINCE

PRINCE is a 64-bit block cipher with a 128-bit key. The key is split into two parts of 64 bits each, k = k0||k1,and extended to 192 bits by the mapping (k0||k1)—> (k0||k0'||k1) := (k0|| (k0>>>1) ⊕ (k0 >>63)||k1). During the encryption, the first two subkeys $k_0$ and $k'_0$ are used as whitening keys, while the third subkey $k_1$ is the key for a 12-round block cipher referred to as PRINCEcore. The high level structure of PRINCE is demonstrated in Fig. 1.

The 12-round process of PRINCEcore is depicted in Fig. 2. A typical round of PRINCEcore consists of an S-box layer, a linear layer and an addition layer. The intermediate computation result called state, is usually represented by a 64-bit vector or a 16-nibble vector.


**Figure 1:** The structure of PRINCE

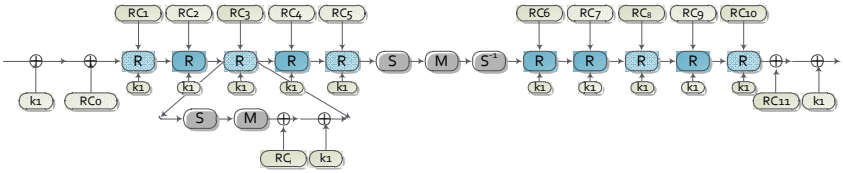


**Figure 2:**PRINCEcore

S-box layer: The cipher uses a 4-bit S-box which is given in Table.1. We denote the S-box and its inverse by S and $S^{-1}$ respectively.

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

**Table 1:**S-box of PRINCE

## 3.The Area Optimization and Verification of PRINCE

In the area optimization method, the PRINCE algorithm mainly has five modules: MatrixMutil, ShiftRow, SubCell, PrinceRound, and the main program Prince. Five modules are described as follows:

MatrixMutil: the column vector of matrix multiplication (the intermediate state is the column vector);

ShiftRow: including the row shift and reverse row shift, through the control signal to choose the corresponding results as output;

SubCell: including S-box replacement and reverse S-box replacement, through the control signal to choose the corresponding results as output;

PrinceRound: complete a round wheel transformation of Prince algorithm

The main program Prince: The module which controls all modules.

### 3.1 The Optimization of MatrixMutil and Experiment

In MatrixMutil, we found that according to the characteristics of the matrix, the element on each line is a corresponding element of the corresponding column vector, the result is obtained by adding these elements. Using this method, we only need to do 64*3 XOR on MatrixMutil, effectively reducing the computational complexity. The Verilog HDL code is given as follows.

```
module MatrixMutil(res,state);
  input [0:63]  state;
  output[0:63]  res;
assign res={ s[ 4]^s[8]^s[12],s[1]^s[ 9]^s[13],s[ 2]^s[ 6]^s[14],s[ 3]^s[ 7]^s[11],
  ……
                s[48]^s[56]^s[60],s[49]^s[53]^s[61],s[50]^s[54]^s[58],s[55]^s[59]^s[63]
  };
endmodule
```

The simulation results of MatrixMutil by Modelsim 6.1f are as follows.

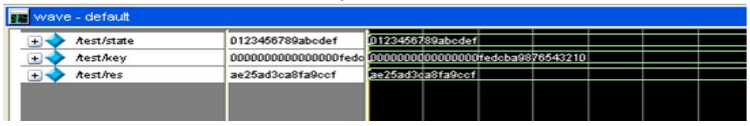


**Figure 3:** Simulation results of MatrixMutil module

### 3.2 Optimization of Repeated Round Modules and Experiment

The same operation module of the PRINCE encryption is placed in the round operation called PrinceRoundso as to realize the repeated calling. It can effectively reduce the area when the encryption is running on hardware. Thus, it is feasible to apply encryption into sensors and other nodes.

In PrinceRound module, the operation order of the corresponding module was controlled by the control signal, which can implement encryption/decryption operations within the module, and it can reduce the number of registers. The Verilog HDL code of main programming is described as follows.

```
assign  tag=(r<=5)?1:0;
```

```
assign  t[0]=tag? state:tem[1], ……t[3]=tag?tem[2]:state,
result=tag?tem[3]:tem[0];
MatrixMutil  MM(tem[0],t[0]);
ShiftRow  SR(tem[1],t[1],tag);
assign  tem[2]= t[2] ^key ^RC[r];
SubCell  SC(tem[3],t[3],tag);
```

Simulation results for the PRINCE round module are shown as Fig. 4.

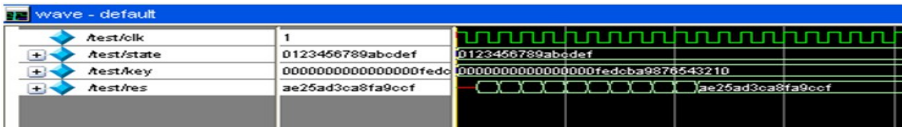


**Figure 4:** Simulation Results for the PRINCE Round Module

### 3.3 Main Program of Prince

The realization of the Prince master control program is to use the counter which controls PrinceRound module to run transformation round 10 times. The code is shown as follows.

```
always @(posedge clk) begin
 cnt  <= (cnt^10)? cnt+1: cnt;
 res  <= ready ? ( (cnt) ?te:t_SC ) :res;
 ready  <= (cnt^10)? 1:0;
 end
PrinceRound PR(t_res,res,key[64:127],cnt);
```

Owing to the method of continuous assignment and the clock signal controlled by the calculator to update, the encryption requires only 12 clock cycles.

| plaintext | k0 | k1 | ciphertext |
|---|---|---|---|
| 0000000000000000 | 0000000000000000 | 0000000000000000 | 818665aa0d02dfa |
| ffffffffffffffff | 0000000000000000 | 0000000000000000 | 604ae6ca03c20ada |
| 0000000000000000 | ffffffffffffffff | 0000000000000000 | 9fb51935fc3df524 |
| 0000000000000000 | 0000000000000000 | ffffffffffffffff | 78a54cbe737bb7ef |
| 0123456789abcdef | 0000000000000000 | fedcba9876543210 | ae25ad3ca8fa9ccf |

**Table 2:**Test Vector of PRINCE


**Figure 5:** Simulation Results of Main Program Prince

The test vector in Table 2 is the vector used in Paper [5], and we also used the same test values on the fifth group in Paper [5]. The simulation results can be seen in Fig. 5, the plaintext for input is 012345789abcdef, the key for input is 0000000000000000 and fedcba9876543210, the ciphertext is ae25ad3ca8fa9ccf. So, the method of optimization is correct.

## 4.Algorithm of Resisting Power Analysis for PRINCE

The random mask is considered for the S-box transform and inverse S-box transform of PRINCE encryption algorithm. Due to the special design of the PRINCE algorithm, it also

needs to do an S-box transform and inverse S-box transformation in the control section. In order to construct S-boxes with random mask, a large amount of memory is needed. So we use a special way to avoid consuming more memory. Before the S-box transform, the intermediate state is XOR with a random array once, then after S-box transform, the result XOR with a random array runs again. The modified code is suitable for completely random mask, partial random mask, and fixed random mask. The algorithm is shown as follows:

---

```
Algorithm 1:Add(state,key0); Add(state,key1); Add(state,RC[0]);
            Add(state,R);SubCell(state,R);Add(state,R);
            for(i=1;i<=5;i++){
             M_layer(state); M_layer(R);
             ShiftRow(state); ShiftRow(R); Add(state,RC[i]); Add(state,key1);
             Add(state,R);// add random mask
             SubCell(state,R);
             Add(state,R);//compensation mask}
             M_layer(state); M_layer(R);
             for(;i<=10;i++){ …
             r_ShiftRow(state);r_ShiftRow(R);
             M_layer(state); M_layer(R);…}//the next 5 rounds
```

---

From Fig. 6, with the input of plaintext and key, the output of the PRINCE algorithm which added a random mask can verify that the random mask is correct.


**Figure 6:** Verification Result of PRINCE Random Mask Algorithm in VC

## 5.Experiment on FPGA and Analysis

### 5.1 Area Comparison

Among the above method, the optimized PRINCE algorithm, un-optimized PRINCE algorithm and the PRINCE random mask algorithm are run on FPGA, synthesized by ISE13.2 and downloaded on Xilinx Virtex-5 LX50T FPGA. The result of the un-optimized PRINCE algorithm on FPGA is shown in Fig. 7. The result of optimized is shown in Fig.8. The result of PRINCE random mask algorithm is shown in Fig. 9 respectively. Table 3 is the comparison of logic resource usage on FPGA.

```
Design Summary:
Number of errors:      0
Number of warnings:   66
Slice Logic Utilization:
  Number of Slice Registers:                5,433 out of  28,800   18%
    Number used as Flip Flops:              5,417
    Number used as Latch-thrus:                16
  Number of Slice LUTs:                     5,317 out of  28,800   18%
    Number used as logic:                   5,145 out of  28,800   17%
      Number using O6 output only:          4,897
      Number using O5 output only:             74
      Number using O5 and O6:                 174
    Number used as Memory:                    155 out of   7,680    2%
      Number used as Dual Port RAM:            64
        Number using O5 and O6:                64
      Number used as Shift Register:           91
        Number using O6 output only:           90
        Number using O5 output only:            1
    Number used as exclusive route-thru:       17
  Number of route-thrus:                      108
    Number using O6 output only:               88
    Number using O5 output only:               17
    Number using O5 and O6:                     3
```

**Figure 7:**Results of the Un-optimized PRINCE Algorithm on FPGA

```
Design Summary:
Number of errors:      0
Number of warnings:   66
Slice Logic Utilization:
  Number of Slice Registers:                5,342 out of  28,800   18%
    Number used as Flip Flops:              5,326
    Number used as Latch-thrus:                16
  Number of Slice LUTs:                     6,382 out of  28,800   22%
    Number used as logic:                   6,213 out of  28,800   21%
      Number using O6 output only:          5,965
      Number using O5 output only:             75
      Number using O5 and O6:                 173
    Number used as Memory:                    155 out of   7,680    2%
      Number used as Dual Port RAM:            64
        Number using O5 and O6:                64
      Number used as Shift Register:           91
        Number using O6 output only:           90
        Number using O5 output only:            1
    Number used as exclusive route-thru:       14
  Number of route-thrus:                       92
    Number using O6 output only:               85
    Number using O5 output only:                4
    Number using O5 and O6:                     3
```

**Figure 8:** Result of Optimized PRINCE Algorithm on FPGA

```
Design Summary:
Number of errors:      0
Number of warnings:   66
Slice Logic Utilization:
  Number of Slice Registers:                5,497 out of  28,800   19%
    Number used as Flip Flops:              5,481
    Number used as Latch-thrus:                16
  Number of Slice LUTs:                     5,562 out of  28,800   19%
    Number used as logic:                   5,393 out of  28,800   18%
      Number using O6 output only:          5,144
      Number using O5 output only:             76
      Number using O5 and O6:                 173
    Number used as Memory:                    155 out of   7,680    2%
      Number used as Dual Port RAM:            64
        Number using O5 and O6:                64
      Number used as Shift Register:           91
        Number using O6 output only:           90
        Number using O5 output only:            1
    Number used as exclusive route-thru:       14
  Number of route-thrus:                       98
    Number using O6 output only:               85
    Number using O5 output only:               10
    Number using O5 and O6:                     3
```

**Figure 9:** Results of PRINCE Random Mask Algorithm on FPGA

| PRINCE | Slice | LUT | Flip Flops |
|---|---|---|---|
| optimized | 5342 | 5317 | 5326 |
| un-optimized | 5433 | 6382 | 5417 |
| Random Mask | 5497 | 5562 | 5481 |

**Table 3:** Comparison of Logic Resource Usage on FPGA

According to the compared data, the optimized PRINCE has advantages in area resource consumption before optimization. The area resource is reduced by 9.8%.

### 5.2 Speed Comparison

Throughput is equal System clock frequency * length of block / clock of encryption and decryption. The length of cipher block is 64, and clock of encryption and decryption is 12. The clock frequency of the optimized PRINCE algorithm is 102.459, so

$$Throughput = 102.459 * 64/12 = 546.448(Mbps) \cdot$$

The clock frequency of un-optimized PRINCE algorithm is 109.939, so

$$Throughput = 100.939 * 64/12 = 538.43(Mbps) \cdot$$

The clock frequency of PRINCE random mask algorithm is 100.472, so

$$Throughput = 100.472 * 64/12 = 535.850(Mbps) \cdot$$

From the above results, we can see that the optimized PRINCE with less area and the encryption speed is faster, the throughput is increased by 1.5%, and it can be testified that we added a fixed value.

## 6. Conclusion

This paper proposed a method based on area optimization for PRINCE, and realized the algorithm of resisting power analysis for PRINCE based on Verilog HDL language and running on FPGA. The experimental results have shown that random mask added correctly, and the algorithm of resisting power analysis which is proposed in this paper added fixed random mask value. Theoretically, it has a certain ability of defending attack.

## References

[1] Bogdanov A，Knudsen L R Leander Get a1．*PRESENT*：*Anultra-lightweight block cipher*[C]．Cryptographic Hardware and Embedded Systems (CHES 2007), 2007，LNCS, Vol.4727, pp.450-466．

[2] Canniere De,Dunkelman O,and K nezevic M.*KATAN and KTANTAN-A family of small and efficient hardwareoriented block ciphers*[C].Cryptographic Hardware and Embedded Systems 2009,2009,LNCS, Vol.5747, pp.272-288.

[3] Guo J, Peyrin T, Poschmann A ,et al. *The LED block cipher*[C]. Cryptographic Hardware and Embedded System(CHES 2011). 2011,LNCS, Vol. 326-341.

[4] Lars Knudsen, Gregor Leander, Axel Poschmann and Matthew J. B. Robshaw. *PRINTcipher: A Block Cipher for IC-Printing*[C].CHES 2010, LNCS, Vol.6225, pp.16-32.

[5] Julia Borgho, Anne Canteaut, Tim Guneysu, Elif Bilge Kavun. *PRINCE:A Low-latency Block Cipher for Pervasive Computing Applications*[C]. Asiacrypt 2012. LNCS, Vol. 7658, pp. 208-225.

[6] Cheng Lei,Sun Bing,Li Chao.*Side Channel Cube Attack on PRINCE*[C].Telecom market,2013, 2:107-114.

[7] LI Lang, DU Guo-quan, ZENG Ting, et al.*Research on the PRINCE Algebraic Attack*[J]. MATHEMATICS IN PRACTICE AND THEORY,2015,45(5):153-159.

[8] ZOU Yi, LI Lang, JIAO Ge. *Differential Fault analysis of PRINCE lightweight cryptographic algorithm*[J]. Computer Science, 2017,44(6A):377-379.