# IPcore implementation susceptibility: A case study of Low latency ciphers

Dillibabu Shanmugam
*dillibabu@setsindia.net*

Ravikumar Selvam
*selvamravik@gmail.com*

Suganya Annadurai
*asuganya@setsindia.net*

***Abstract*—Security evaluation of third-party cryptographic IP (Intellectual Property) cores is often ignored due to several reasons including, lack of awareness about its adversity, lack of trust validation methodology otherwise view security as a byproduct. Particularly, the validation of low latency cipher IP core on Internet of Things (IoT) devices is crucial as they may otherwise become vulnerable for information theft. In this paper, we share an (Un)intentional way of cipher implementation as IP core(hard) become susceptible against side channel attack and show how the susceptible implementation can be experimentally exploited to reveal secret key in FPGA using power analysis. In this paper our contributions are: First, we present Look-Up Table (LUT) based unrolled implementation of `PRINCE` [1] block cipher with place and route constraints in FPGA. Second, using power analysis attack we recover 128-bit key of `PRINCE` with complexity of $2^9$. Finally, we conclude the paper with the experimental results.**

***Keywords*—Side channel Attack, Low latency Cipher, Intellectual Property(IP) Core**

## 1. Introduction

As ubiquitous network and constrained devices scaled-up rapidly for human-being betterment, security concerns for information protection also multitude. To address these concerns to an extent, researchers come up with light-weight ciphers [2]. Though ciphers are secure mathematically, implementation attack and issues in adaptation for constrained devices make further exploration on ciphers interms of trade off factors such as low energy (less power consumption), low latency (operates at high speed), less area (single circuit for encryption and decryption). From the beginning to latest design of these ciphers [3] brings many changes for betterment. However, implementation vulnerability prevails continuously in one or other form; unless proper countermeasures is considered.

First, implementation vulnerability against side channel attack is exploited by paul kocher [4] in 1999. Later, countermeasures were proposed such as masking [5] and hiding to counteract. However, attack on countermeasures carried out on each counteract [6]. One such counteract technique [7] as well as better trade-off factor (low latency) is unrolled (single clock execution) implementation, say as in `PRINCE`. Even-then attack on unrolled implementation is explored using t-test and able to retrieve some part of key [8]. In t-test, function execution moment is identified to normalize uneven delay and glitches by appropriate inputs. This clearly shows unrolled implementation is susceptible under this technique. In this paper, we studied the possibility of normalizing the uneven delay and glitches using implementation constraints, has a kind of susceptible implementation (loophole) on unrolled architecture of the cipher. We found that this makes the cipher vulnerable against power analysis attack and successfully reveals all the key bytes. Even though susceptible implementation makes vulnerable, as such every possible is there for this kind of implementation, (Un)intentional in the third party IP cores [9] to steal the secret information. This might possible be consider as a Hardware Trojan(HT) with following characteristics. First, physical characteristic: Change conventional place and route to constraints place and route, as a structure change. Second in activation characteristic, trigger signal is always high; whenever cipher execution starts. Finally in action characteristics, leak secret information via side channels. Therefore strong need has raised to validate the IP core before usage.

Researchers have been exploring on low latency ciphers for betterment (resistant against crypt-analysis as well as efficient trade-off for constraint device), and have come up with better ciphers such as, `PRINCE` [1], `MIDORI` (low energy) [10], `QARMA`,`MANTIS` (Tweakable block cipher) [11], [12]. On the other hand to counteract partial DPA on `PRINCE`, first and last round Threshold Implementation(TI) is suggested in [13].

Our contributions in this paper are as follows.
First, we show FPGA (LUT) architecture for unrolled version of PRINCE block cipher with place and route constraints. Then, we demonstrate the power analysis attack to recover 128-bit key of `PRINCE` with complexity of $2 * (16 * 2^4) = 2^9$. Further, we conclude the paper with the experimental results.

Organization of the Paper. We share susceptible implementation of low latency cipher, `PRINCE` in Section 2. We described, how the implementation is vulnerable against DPA in Section 3 and 4. Finally, in section 5, paper is concluded.

## 2. Implementation of PRINCE

### 2.1. Brief on PRINCE

PRINCE is a symmetric block cipher based on a SPN structure. Its alpha reflective property makes single circuit suitable for both encryption and decryption. Further, the algorithm is proposed for unrolled (computes cipher in single clock cycle) implementation, that means, where application needs low latency encryption schemes. PRINCE consists of following functions such as pre-key whitening, round functions, and post-key whitening. PRINCE is a 64-bit block cipher with a 128-bit key. The key is split into two parts of 64 bits each, represented as $K0$ and $K1$. Each round function consists of four sub-functions: S-Layer (S), M-Layer (M), AddRound-Constants and AddRoundKey. The first 5 rounds perform the sub-functions in the forward (above) order, while the last 5 rounds perform their inverses in the reverse order. There is an intermediate function block consisting of S-Layer, $M'$ Layer and inverse S-Layer ($S^{-1}$) between the first and last 5 rounds.

### 2.2. FPGA implementation of PRINCE

The design of PRINCE facilitates to be very speed and compact in terms of resource requirement, especially if implemented in hardware. In general, ciphers are implemented using logic gates. In this paper, PRINCE cipher is implemented completely in LUT. All possible combinational inputs for each function is analyzed and its output could be precomputed as a LUT. In this implementation, the key whitening operations are combined with the Key K1 and appropriate round constant. This has been mapped to single LUT in the beginning and the end of the cipher.Similarly, LUT value is calculated for S-box function and M-layer function which are given in the table Table 1 and Table 2.

TABLE 1. LUT VALUES OF PRINCE FUNCTIONS (INITIAL-XOR,MIX-COLUMN,ROUND-XOR)

| Initial Xor | $M$ | $K1$ | $K0$ | $IX = M \oplus K1 \oplus K0$ |
|---|---|---|---|---|
| Mixcolumn | S3 | S2 | S1 | $MO = S3 \oplus S2 \oplus S1$ |
| Add_Key_Constant | MO | RC | K1 | $R = MO \oplus RC \oplus K1$ |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 |
| 4 | 0 | 1 | 1 | 0 |
| 5 | 1 | 0 | 0 | 1 |
| 6 | 1 | 0 | 1 | 0 |
| 7 | 1 | 1 | 0 | 0 |
| 8 | 1 | 1 | 1 | 1 |
| | | | | $LUT(hex) = 96$ |

The cipher is implemented on FPGA(Virtex2vp7) using Verilog hardware description language. Each SLICE in FPGA has two LUTs(G and F); in that, 2245 F-LUT SLICE is used for implementation. However both LUTs can be used for optimized implementation. Syntax for LUT initialization and implementation are given as below.

TABLE 2. LUT VALUES OF PRINCE S-BOX FUNCTION

| S-BOX | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| S.No | IX4 | IX3 | IX2 | IX1 | S4 | S3 | S2 | S1 |
| | | | | | 3 | 2 | F | 7 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 2 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| | | | | | 7 | 2 | 1 | C |
| 5 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 7 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 8 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| | | | | | 4 | 3 | 3 | 2 |
| 9 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 10 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 11 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 12 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| | | | | | 5 | F | 1 | 6 |
| 13 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 14 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 15 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 16 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| LUT | | | | | 5473 | F322 | 131F | 62C7 |

**Key whitening(Initial Xor):**
Initialization: defparam IX63.INIT = 8'H96;
LUT3:IX63(.O(IX[63]),.I0(M[63]),.I1(K1[63]),.I2(K0[127]));

**S-box:**
Initialization: defparam R1S63.INIT=16'H5473;
LUT4:R1S63(.O(S[63]),.I3(IX[63]),.I2(IX[62]),.I1(IX[61]),.I0(IX[60]));

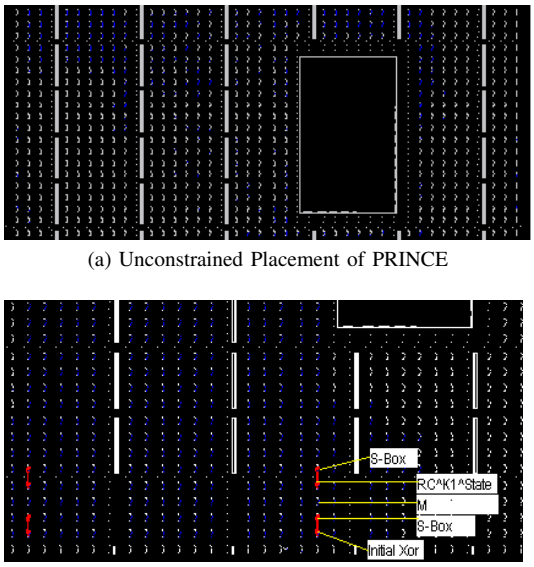
**Mixcloumn:**
Initialization: defparam R1M63.INIT = 8'H96;
LUT3:R1M63(.O(MO[63]),.I0(S[59]),.I1(S[55]),.I2(S[51]));

**RC $\oplus$ K1 $\oplus$ state:**
Initialization: defparam FZ0.INIT=8'H96;
LUT3:R1KRC63(.O(R1O[63]),.I0(MO[63]),.I1(K1[63]),.I2(RC[63]));

### 2.3. (Un)Constrained Place and Route of the Implementation

Normally, place and route will be taken care by Electronic Design Automation (EDA) tools. However, tool provides option for constraints in PAR implementation of the design as well. In this paper both constrained and unconstrained implementation of PRINCE are done and compared their experiments results. First, PRINCE LUT based implementation with normal PAR is done, which routes LUT functions in random manner as shown in the Figure 1a. This end-up in uneven wire-delay between two functions because of some input(s) to LUT functions could be delayed, while others are on-time. Glitches are prone in this style of implementation. This will eventually reflects in power consumption of the circuit and becomes difficult for DPA as in [8]. On the other-hand, implementation of LUT based PRINCE cipher

with PAR constraint routes functions (LUTs) in sequential manner as shown in figure 1b.




(a) Unconstrained Placement of PRINCE

(b) Constrained Placement of PRINCE

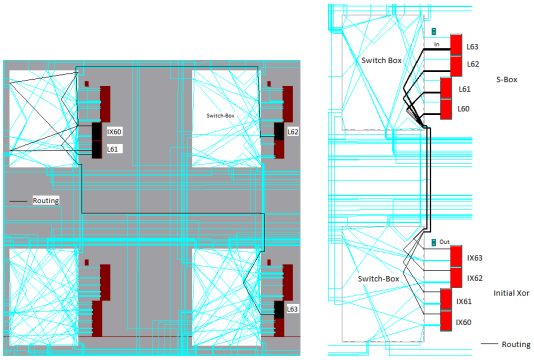
Figure 1. (Un)constrained Placement of PRINCE

This will enforce almost even wire-delay between two functions, since all the LUT inputs arrives on-time. This reduce the glitches in the circuit. Therefore power consumption of the circuit is almost proportional to data handled by the function at the moment, which is vulnerable against DPA. The syntax for implementation constraint on place and route is follows.//synthesis attribute RLOC of $IX63$ is "$X0Y0$".

## 3. Analysis on Implementation Vulnerability

In round based implementation, register is used to store and update intermediate values of the algorithms. The specific register is targeted to reveal the secret information using DPA attack. In this line of attack round based implementation of PRINCE is exploited in [14]. However, this will not work for unrolled implementation of PRINCE. Since their is no intermediate register, all the functional circuit values are updated on the fly through interconnects, such as switch-box and wires.This makes execution of a function on slightly different note of time and also difficult to interpret exact execution time of the particular fiunctional unit. But, the Welch's t-test [15] is useful to find the point of execution (function execution time) appropriately to do power analysis on that point. This is explored in the paper [8]; however, only partial keys are retrieved.

Reason behind this, implementation functional circuits (LUTs) are placed in random manner and interconnected by wires, switches for routing. An intermediate function as a LUT receive inputs from different LUT output, these inputs arrives on different note of time, which cause delay in execution of the function. In meantime glitches propagate in the LUT and varies power consumption of the circuit, which makes the Welch's t-test unsuccessful after certain extent. Therefore in this paper, PRINCE LUT implementation with PAR constraint is explored to reduce the random delay. By PAR constraints subsequent functional LUTs were placed close to each other (row-wise on FPGA), which in turn reduce route distance between them. This makes less glitch propagation into a function(LUT). Consequently, LUT output will not fluctuates randomly which in turns consumes data-dependent power consumption.




(a) Unconstrained Route (b) Constrained Route

Figure 2. (Un)constrained Route of PRINCE

In this paragraph how PAR constraints is applied on PRINCE is described. First, placement of PRINCE functional LUT is elaborated. Initial xor function (Message $\oplus K1 \oplus K0$) is placed in second row from bottom (marked in red color) and then first round S-box function is placed in third row, Mixcolumn and shift-row function in fourth row, and then xor function (Round constant $\oplus K1 \oplus$state value) in sixth row. The same process is used for each round functions in sequential manner as shown in the Figure 1b. Similarly PRINCE functional LUT routing is elaborated as follow. For instance, delay between 4-bit initial xor function and 4-bit first round S-box function is almost even wire length and close to each other as shown in the Figure 2b; whereas delay between same 4-bit initial xor function and 4-bit first round S-box function in unconstrained PAR implementation has uneven wire routing, scattered over the FPGAas shown in the Figure 2a. Therefore, unconstrained PAR implementation becomes difficult to attack; whereas constraint PAR implementation is vulnerable against side channel attack.

## 4. DPA on PRINCE

A DPA attack can be summarized in the following five steps:

- Choose an intermediate point of interest of the executed algorithm.
- Measure the power consumption.
- Calculate hypothetical intermediate values.
- Map intermediate values to hypothetical power consumption values, $(P_{hyp})$.

- Statistically compare the hypothetical power values with the measured power traces.

The above five steps is elaborated on PRINCE as follow. **Step 1: Point of Interest (POI)** In PRINCE, S-box is a nonlinear function, where key and plaintext gets gel together to achieve confusion property. This makes power consumption random for different inputs, which is vital for power analysis. Therefore from DPA perspective POI is S-box. The output of the first round S-box, $POI_1$, is influenced by two 64-bit keys: the whitening key K0 and the round key K1. Therefore, an attack on $POI_1$ will give an adversary an key $K^{'}$ as,

$$K^{'} = K0 \oplus K1 \tag{1}$$

The adversary knowing $K^{'}$ cannot recover the individual key parts K0 and K1, and thus another attack is required on the second round function S-box, $POI_2$, to have $K1$. From that arriving K0 is trivial. Thus in PRINCE, two POI are required to retrieve key completely as shown in the Figure 3.

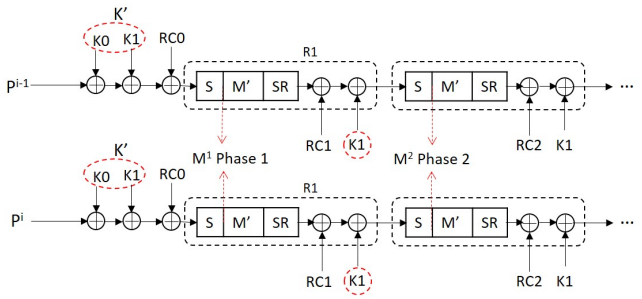



Figure 3. PRINCE Attack phases

**Step 2: Measure power consumption ($P_{msd}$)** Unrolled PRINCE cipher is implemented on FPGA, which is built using Complementary Metal Oxide Semiconductor (CMOS) circuit. The circuit leaks information mostly through dynamic power consumption on execution, which is captured and stored for required number of encryption for analysis. From the Figure 4 it is clear that power consumption raises abruptly then comes down. Measured power traces are stored in matrix form $P_{msd}$(i,T), where i represent $i^{th}$ encryption and T represent total number of points in a power trace.

**Step 3: Calculate hypothetical intermediate values** Power analysis attack works on divide and conquer approach. One feasible way to divide the cipher for analysis needs to be same as its S-box size (4-bit). Therefore Chunk of 4-bit is taken at a time from POI for analysis. All the possible combination of key values (search space) for those 4-bit with corresponding 4-bit of plaintext used for encryption are extracted to generate Hypothetical intermediate value at POI. Hypothetical intermediate value for first, $L_j^{1,i}$ and second, $L_j^{2,i}$ POI are given in the equations 2 and 3

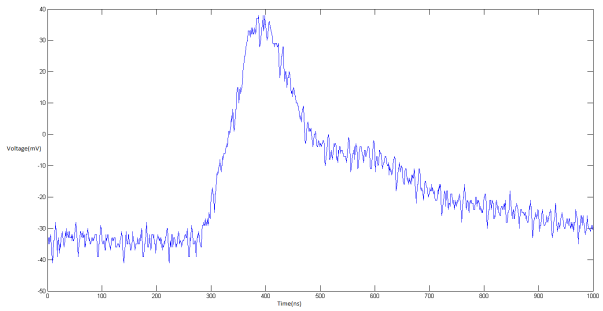



Figure 4. PRINCE cipher power consumption

respectively.

$$L_j^{1,i} = S(p_j^i \oplus K_j^{'} \oplus RC0_j) \tag{2}$$

$$L_j^{2,i} = S(SR(M'(S(p_j^i \oplus K_j^{'} \oplus RC0_j))) \oplus RC1_j \oplus K1_j) \tag{3}$$

$P_j^i$ is denoted as plaintext of $j^{th}$ nibble of $i^{th}$ encryption. j ranges from $0 \leq j \leq 15$, $K_j^{'} = K0_j \oplus K1_j$, all possible combination of key, $K_j^{'}$ as follow: 0,1, ...15.

**Step 4: Compute hypothetical power consumption** In order to arrive hypothetical power consumption, power model should be realistic to describe power consumption between each and every intermediate stages of the algorithm executed in the hardware module. Hamming distance (HD) model suits very well to describe the power consumption of unrolled PRINCE. Normally, HD is calculated between two state values of the register. In this scenario their is no register concept, since entire cipher is executed in single clock cycle. Therefore, HD is calculated using state value at the POI between present and previous encryption. Hypothetical power consumption at $POI_1$ and $POI_2$ are represented in the equation 4 and 5 respectively.

$$P_{hyp}^{1,i} = HD(L_j^{1,i} \oplus L_j^{1,i-1}) \tag{4}$$

$$P_{hyp}^{2,i} = HD(L_j^{2,i} \oplus L_j^{2,i-1}) \tag{5}$$

$P_{hyp}^i$ denotes the power consumption of $i^{th}$encryption

**Step 5: Correlation between measured ($P_{msd}$) and hypothetical ($P_{hyp}$) power consumption** Pearson's Correlation coefficient is used to correlate between measured power consumption and hypothetical power consumption. Each column of the $P_{msd}$ is correlated with each column of the $P_{hyp}$ to obtain rank matrix. On plot this rank matrix, highest correlation value shows the correct key guess.

$$r(i,j) = \frac{\sum_{i=1}^{n}(P_{msd,i} - \bar{P_{msd,i}})(P_{hyp} - \bar{P_{hyp}})}{\sqrt{\sum_{i=1}^{n}(P_{msd,i} - \bar{P_{msd}})^2}\sqrt{\sum_{i=1}^{n}(P_{hyp,i} - \bar{P_{hyp}})^2}} \tag{6}$$

Here, i and j represent $i^{th}$ row and $j^{th}$ column of the corresponding power consumption matrix.

### 4.1. *Attack description*

Key values used in experiment are tabled as below.

| Nibbles | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| K' | B | 6 | 0 | 0 | 9 | 1 | 9 | 9 | 6 | 4 | 0 | 0 | 1 | 8 | 3 | A |
| K1 | 6 | 9 | 8 | B | 3 | 1 | E | 5 | F | 0 | 6 | B | 4 | 6 | 2 | 9 |
| K0 | D | F | 8 | B | A | 0 | 7 | C | 9 | 4 | 6 | B | 5 | E | 1 | 3 |

*Phase 1 attack at* $POI_1$: First, $16^{th}$ nibble (MSB 4-bit) of all encryption with all possible combination of 4-bit key are used to compute hypothetical intermediate value, $L_j^{1,i}$. Then HD is calculated between $L_{16}^{1,i}$ and $L_{16}^{1,i-1}$ to arrive hypothetical power consumption, $P_{hyp}^1$. Which is correlated with measured power consumption, $P_{msd}$ to obtain rank matrix (r). On plotting the rank matrix, highest correlation value, 0.01628 gives correct key guess, 13-1 = 12(B) as shown in the figure subsection 4.1.

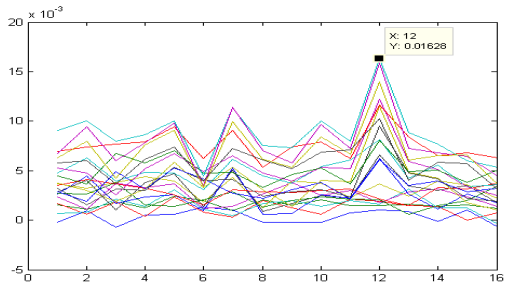



Figure 5. Key guess of MSB nibble, Key $K^{'}$

The same procedure is repeated for all the remaining key of $K^{'}$ and plotted its correct key guess as shown in Figure 6.

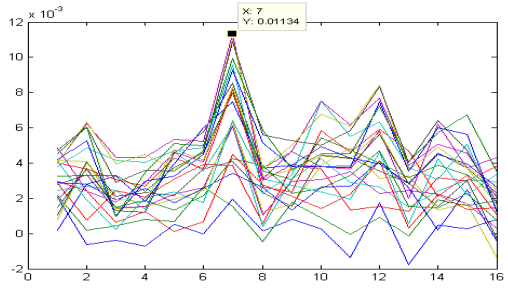
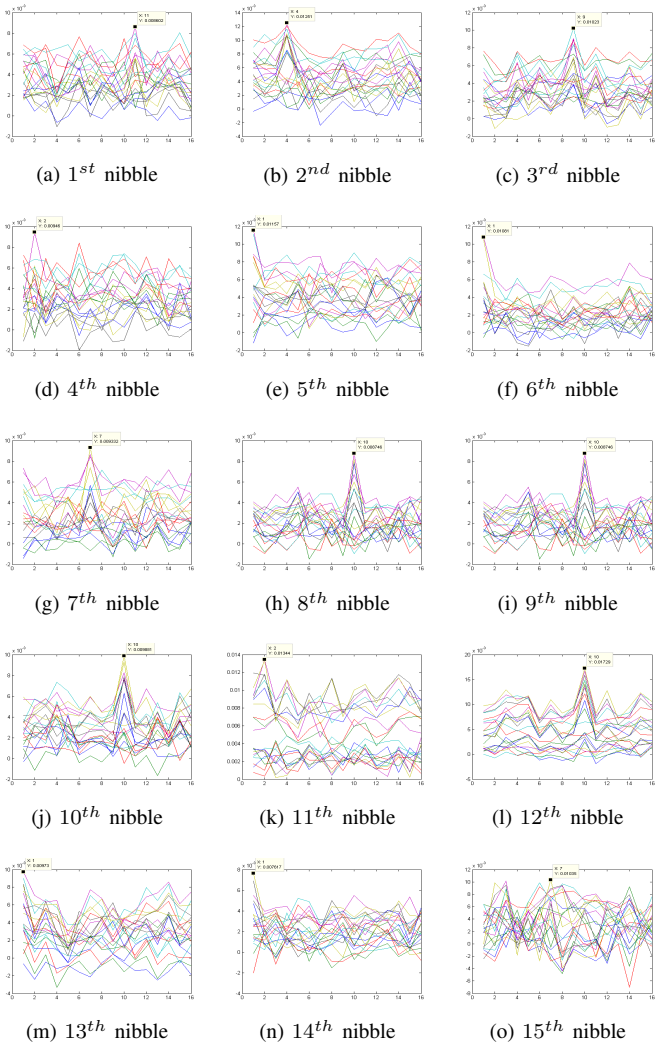



Figure 7. Key guess of MSB nibble Key, K1

*Phase 2 attack at* $POI_2$: Correct key guess retrieved at phase 1 is substituted while arriving hypothetical intermediate value at $POI_2$, $L_{16}^{2,i}$. Then compute hypothetical power consumption, $P_{hyp}^2$, which is correlated with $P_{msd}$ to retrieve the nibble of key, $K1_{16}$ as shown in the Figure 7. The same procedure is repeated for all the remaining key of K1 and plotted its correct key guess as shown in Figure 8.



(a) $1^{st}$ nibble (b) $2^{nd}$ nibble (c) $3^{rd}$ nibble

(d) $4^{th}$ nibble (e) $5^{th}$ nibble (f) $6^{th}$ nibble

(g) $7^{th}$ nibble (h) $8^{th}$ nibble (i) $9^{th}$ nibble

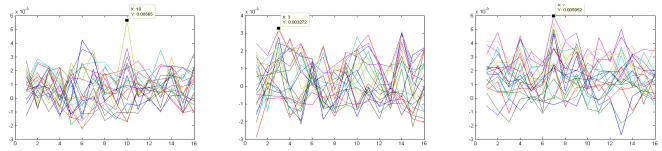
(j) $10^{th}$ nibble (k) $11^{th}$ nibble (l) $12^{th}$ nibble

(m) $13^{th}$ nibble (n) $14^{th}$ nibble (o) $15^{th}$ nibble

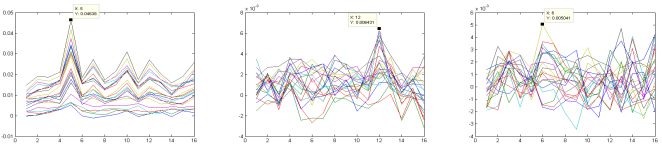
Figure 6. Phase 1 attack of PRINCE : Key, $K^{'}$

### 4.2. Attack complexity

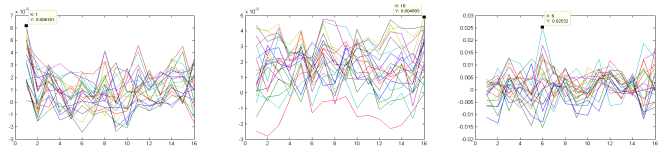
Differential Power analysis(DPA) is divide and conquer approach. That is, instead of trying brute force approach to reveal 128 key bits with complexity of $2^{128}$; key bits are attacked with complexity of $(2^4 * 16) * 2 = 2^9$. Description in detail as follow. To reveal $16^{th}$ nibble of key, $K^{'}$ 4-bit key hypothesis is required for all 16 key possible combination. Therefore attack complexity to retrieve single nibble is $2^4$. For 16 nibbles requires $2^4$ * 16 computation. This is to recovery $K^{'}$ alone. However to find K1 and K0 from K' is non-trivial. Therefore explored similar attack at the second round to reveal K1 with attack complexity $2^4$ * 16. Using K' and K1 easy to arrive K0. Thus, we requires ($2^4$ * 16) * 2 attack complexity. The attack complexity is significantly reduced from $2^{64}$ to $2^9$.
From figure 9, it is clear that 200,000 samples are enough to retrieve $16^{th}$ nibble of K' and K1.



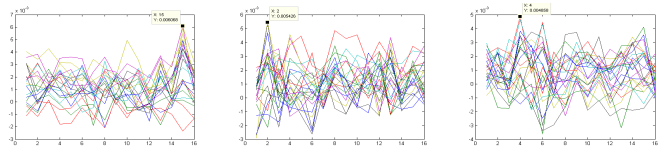
(a) $1^{st}$ nibble (b) $2^{nd}$ nibble (c) $3^{rd}$ nibble



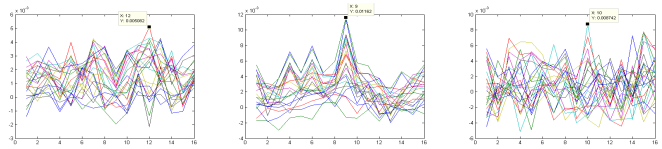
(d) $4^{th}$ nibble (e) $5^{th}$ nibble (f) $6^{th}$ nibble



(g) $7^{th}$ nibble (h) $8^{th}$ nibble (i) $9^{th}$ nibble

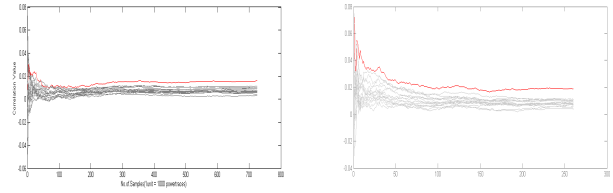


(j) $10^{th}$ nibble (k) $11^{th}$ nibble (l) $12^{th}$ nibble



(m) $13^{th}$ nibble (n) $14^{th}$ nibble (o) $15^{th}$ nibble

Figure 8. Phase 2 attack of PRINCE : Key, K1



(a) $16^{th}$ nibble of K' (b) $16^{th}$ nibble of K1

Figure 9. Attack on PRINCE

However 700,000 samples are required for other nibbles. This can be reduced by combining LUT constraint and `Welch's t-test`. We would like to explore in our future work.

## 5. Conclusion

In this paper, we studied and explored the possible implementation vulnerability of low latency crypto-graphic cipher, `PRINCE` as an IP core. Likewise, the attack can also be extent to ciphers `MIDORI`, `QARMA` which necessitate proper verification and validation of IP core in future. Threshold Implementation (TI) as a countermeasure for first and last round is proposed in [13] to increase resistant against DPA. However, its security needs to be validated experimentally under susceptible implementation before commercial usage.

## References


[1] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger *et al.*, "Prince–a low-latency block cipher for pervasive computing applications," in *Advances in Cryptology–ASIACRYPT 2012*. Springer, 2012, pp. 208–225.

[2] A. Y. Poschmann, "Lightweight cryptography: cryptographic engineering for a pervasive world," in *PH. D. THESIS*. Citeseer, 2009.

[3] https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers.

[4] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference*. Springer, 1999, pp. 388–397.

[5] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen, "A side-channel analysis resistant description of the aes s-box," in *International Workshop on Fast Software Encryption*. Springer, 2005, pp. 413–423.

[6] E. Peeters, F.-X. Standaert, N. Donckers, and J.-J. Quisquater, "Improved higher-order side-channel attacks with fpga experiments," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2005, pp. 309–323.

[7] S. Bhasin, S. Guilley, L. Sauvage, and J.-L. Danger, "Unrolling cryptographic circuits: A simple countermeasure against side-channel attacks," in *Proceedings of the 2010 International Conference on Topics in Cryptology*, ser. CT-RSA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 195–207. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-11925-5_14

[8] V. Yli-Mäyry, N. Homma, and T. Aoki, "Improved power analysis on unrolled architecture and its application to prince block cipher," in *Lightweight Cryptography for Security and Privacy*. Springer, 2015, pp. 148–163.

[9] http://scialert.net/abstract/?doi=rjit.2016.17.28.

[10] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni, "Midori: a block cipher for low energy," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2014, pp. 411–436.

[11] R. Avanzi, "The qarma block cipher family – almost mds matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes," Cryptology ePrint Archive, Report 2016/444, 2016, http://eprint.iacr.org/2016/444.

[12] C. Beierle, J. Jean, S. Klbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The skinny family of block ciphers and its low-latency variant mantis," Cryptology ePrint Archive, Report 2016/660, 2016, http://eprint.iacr.org/2016/660.

[13] A. Moradi and T. Schneider, "Side-channel analysis protection and low-latency in action - case study of prince and midori," Cryptology ePrint Archive, Report 2016/481, 2016, http://eprint.iacr.org/2016/481.

[14] R. Selvam, D. Shanmugam, and S. Annadurai, "Vulnerability analysis of PRINCE and RECTANGLE using CPA," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security, CPSS 2015, Singapore, Republic of Singapore, April 14 - March 14, 2015*, J. Zhou and D. Jones, Eds. ACM, 2015, pp. 81–87. [Online]. Available: http://doi.acm.org/10.1145/2732198.2732207

[15] B. L. Welch, "The generalization ofstudent's' problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1/2, pp. 28–35, 1947.