# Indian Institute of Technology (IIT-Kharagpur)

**SPRING Semester, 2022**
**COMPUTER SCIENCE AND ENGINEERING**

**CS60004: Hardware Security**

**End Semester Examination**

**Full Marks: 50**

**Time allowed: 1 hours 45 minutes**

**1.** Consider Fig 1(a), which shows the $i^{th}$ stage of APUF (Arbiter PUF). We consider an alternative PUF design using a similar cascading of switches followed by the same arbiter logic, as in APUF. The alternative, is called as PAPUF. Its $i^{th}$ stage is shown in Fig 1(b). Architecturally, the classic APUF is similar to the PAPUF, with only the stages (switching elements) modified.

We wish to build a linear additive mathematical model for PAPUF. In this context answer the following questions:

(a) As shown in Fig 1(b), every stage has 4 delay elements, denoted as $p_i, r_i, q_i$ and $s_i$. Write expressions for the propagation delay of the input trigger signal at the output of the $(i+1)^{th}$ stage. Hence, enumerate the delay difference between the top and bottom paths, assuming that the challenge bit in the $(i+1)^{th}$ stage is $\mathbf{c}[i+1] \in \{+1, -1\}$.

(8 marks)

(b) Establish a compact representation for the final delay difference after the $n^{th}$ stage assuming that the challenge is an $n$-bit vector where each entry is $+1$ or $-1$. The final expression should be in terms of a feature vector $\boldsymbol{\phi}$ (in terms of the challenge bits) and a weight vector $\mathbf{w}$ (in terms of the delay units).

(4 marks)

(c) Comment on how the linear model for the PAPUF differs from that of the classic APUF.

(3 marks)


(a) APUF's $i$-th stage (b) PAPUF's $i$-th stage

Figure 1: Switching stages of classic APUF and PAPUF. Depending on the challenge bit $\mathbf{c}[i] \in \{+1, -1\}$, a pair of delay elements is selected.

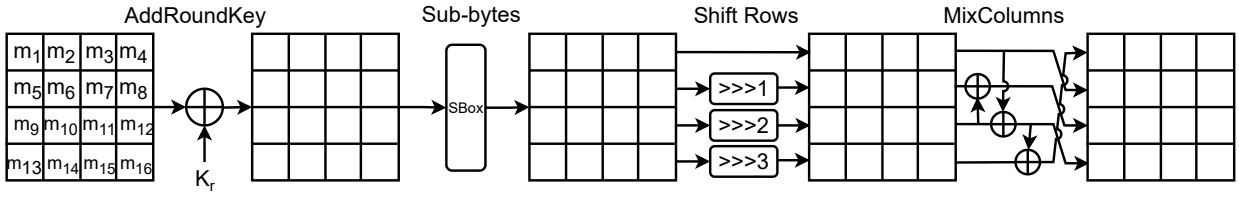


Figure 2: AES-SKINNY Round Operations

**2.** Due to a fitness regime, AES became skinny and got named as AES-SKINNY. It takes a 64-bit plaintext $\mathbf{m} = m_1, m_2, \cdots, m_{16}$, and a 64-bit key as input where $m_i$ is a nibble and computes the ciphertext after R=20 rounds. Similar to AES, the plaintext and the state matrix can be represented as a $4 \times 4$ matrix, albeit the elements are nibbles rather than bytes (and hence can be considered as elements of $GF(2^4)$). A round function of AES-SKINNY consists of four sub-operations namely, `AddRoundKey, SubBytes, ShiftRows` and `MixColumns` as shown in Figure 2. Like AES the last round does not have `MixColumns` step. However, there are some differences: The `SBoxes` in the `SubBytes` operation are $4\times4$ mappings. Moreover the `ShiftRows` and `MixColumns` operations are changed slightly as compared to AES. Here, the `ShiftRows` operation performs right-rotation of the cells in the state matrix (as shown in Figure 2). The second, third and fourth rows are shifted by 1, 2 and 3 cells respectively. The `MixColumns` operation involves multiplication of the state matrix with the matrix $\mathbf{M}$, i.e $\mathbf{X}^{r+1} = \mathbf{M}.\mathbf{X}^r$ where $\mathbf{X}^r$ is the state matrix. The matrix $\mathbf{M}$ is given by:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Additionally, the new `SBox` has the property that the average number of values $x$ satisfying the equation, $\texttt{SBox}^{-1}(x) \oplus \texttt{SBox}^{-1}(x \oplus \alpha) = \beta$, for a given choice of $\alpha, \beta$ is 1. Note, here $x, \alpha, \beta$ are all nibbles.

You need to perform a Differential Fault Attack (DFA) on AES-SKINNY as we did in class on AES and retrieve the last round key. In this regard, answer the following questions.

(a) Suggest a conducive fault model - bit, byte, nibble to perform a fault attack? Explain briefly. (2 marks)

(b) Given R rounds, in which round should the fault be injected so that the fault propagates to all bytes in the ciphertext? Show the fault propagation across different rounds. (5 marks)

(c) Explain how DFA will work assuming the fault is injected in the first cell of the state matrix at the appropriate round. Derive the fault differential equations to determine the last round key. Estimate the complexity of the remaining search space of the key due to a single fault injection. (8 marks)

**3.** (a) A 'not-so-secured' application authenticates legitimate users by means of 4 digit numerical ***pins***. The following code snippet shows the pin authentication mechanism.

```
char* inp_pin = argv[1];
char* g_mem = mmap(0x00, 10);
int wrong_key = 0;
for (int i = 0; i < 4; i++) {
    if (inp_pin[i] == *(mem + i))
        continue;
    else
        wrong_key++;
}
if (wrong_key)  printf(''You entered Wrong key!'');
else     printf(''Welcome! You are logged in.'');

```

The pin entered by the user is stored in a buffer named `inp_pin`. The entered pin is then checked digit by digit with the actual pin stored in memory. The application is running on an 8-bit system and has a specially designed cache that has 10 lines and each line can store 8 bits of data. In other words, the cacheline size is 8 bits and there are 10 such cachelines. The correct pin is stored in contiguous memory locations starting from address 0x00 where each digit is stored as 8-bit number. An adversary wants to leak the secret pin by performing `Prime+Probe` attack. It primes(fills up) the entire cache with arbitrary data and lets the victim or user to enter the correct pin. After the login operation, the adversary probes the cache and measures the time required to access each of the 10 cache lines.

(i) Explain how the adversary can determine the digits of the pin by the above attack scenario mentioned. (4 marks)

(ii) For a 4-digit pin, the brute force approach requires key space complexity of $10^4$. What is the key space complexity for the above attack? (2 marks)

(b) We know that the Square and multiply algorithm to implement modular exponentiation $M^d \text{mod } N$ is vulnerable to simple power attack due to the variation of power consumptions due to squaring and multiplication operations. The Montgomery Ladder algorithm proposes a balanced exponentiation technique which helps to mitigate such attacks. The exponent $d$ is encoded as a bitstring $(d_0, d_1, \cdots, d_{n-1})$ where $d_0$ is the leading one. Complete the following code snippet.

**Algorithm 1:** Montgomery Ladder algorithm

```
1  R_0 ← 1;
2  R_1 ← M;
3  for i = 1 to n − 1 do
4  |   if d_i == 0 then
5  |   |   R_1 ← ------(i)------;
6  |   |   R_0 ← ------(ii)------;
7  |   end
8  |   else if d_i == 1 then
9  |   |   R_0 ← ------(iii)------;
10 |   |   R_1 ← ------(iv)------;
11 |   end
12 end
13 return R_0;
```

(4 marks)

(c) Perform Template attack on Algorithm 1 by considering the modulus ($N$) to be 17 and the base ($M$) to be 13, while the value of $n = 4$. Note that the MSB is 1 by definition.

The attacker builds templates on a known device for the power consumptions due to the computation of the modular exponentiation using the Montgomery's ladder. The template consists of the tuple $T_1, T_2, T_3$, where each $T_i$ denotes the power consumption while processing the $i^{th}$ iteration. In the table, the column heads indicate the various 3-bit key choices starting from bit positions 1 to 3.

The template for the unknown exponent of interest is $T_1 = 0.827$, $T_2 = 1.060$ and $T_3 = 0.990$. Find the 3 bit exponent by performing Least Mean Square Test using the below template (refer to Table 1). Hint: The formula for Mean Square Test: $[(t_1 - m_1)^2 + (t_2 - m_2)^2 + (t_3 - m_3)^2]^{1/2}$ (10 marks)

| | **000** | **001** | **010** | **011** | **100** | **101** | **110** | **111** |
|---|---|---|---|---|---|---|---|---|
| $\mathbf{T_1}$ | 1.033 | 1.046 | 0.855 | 0.868 | 0.835 | 1.146 | 0.7690 | 0.934 |
| $\mathbf{T_2}$ | 1.033 | 1.046 | 2.855 | 2.868 | 0.835 | 1.146 | 0.769 | 0.934 |
| $\mathbf{T_3}$ | 1.033 | 3.046 | 0.855 | 0.868 | 0.835 | 3.146 | 0.769 | 0.934 |

Table 1: Power Template for 3-bit Exponents