Evaluation of A Memory Protection Design Against Replay Attack

January 3, 2014

Abstract

Security becomes increasingly important to computing systems. Proof of the security of a design is even critical. This paper presents an approach to evaluating the data authentication design for memory protection in the embedded systems.

1 Introduction

Security becomes increasingly critical in embedded systems. Most embedded systems consist of secure processor chips and insecure off-chip memory components. To protect the system from attacks on the insecure component, data in the off-chip memory often need to be encrypted and authenticated. Use of data tagging is a common approach to protect data integrity, where data in the memory is attached with a tag and the tag value is checked each time the data is fetched and used by the processor; if the tag value is changed, the data is deemed as tampered and invalid.

There are two typical attacks on such a system:

- *Random Tag Attack*, where the true data/tag pair is replaced with a randomly or purposely picked data and a forged tag.
- *Replay Attack*, where a previously observed valid data and tag pair are replayed. The valid copy can be from the same or a different memory location.

To counter such attacks, existing protection approaches basically use nonce to control the tag generation, with which the tag value is not only dependent on the data to be tagged but also is sensitive to the data's memory location and assess time, as abstracted in Figure 1.

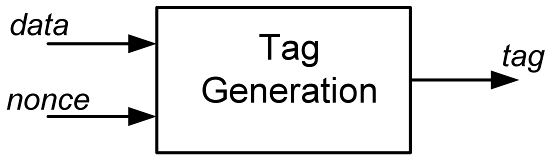

Figure 1: Tag Generated with Nonce

In this paper, we evaluate the security offered by such a design, especially with the focus on the design, $CETD(CostEffectiveTagDesign)$, which was proposed in [1] for embedded systems.

The main contributions made in this paper are:

2 CETD Design Overview

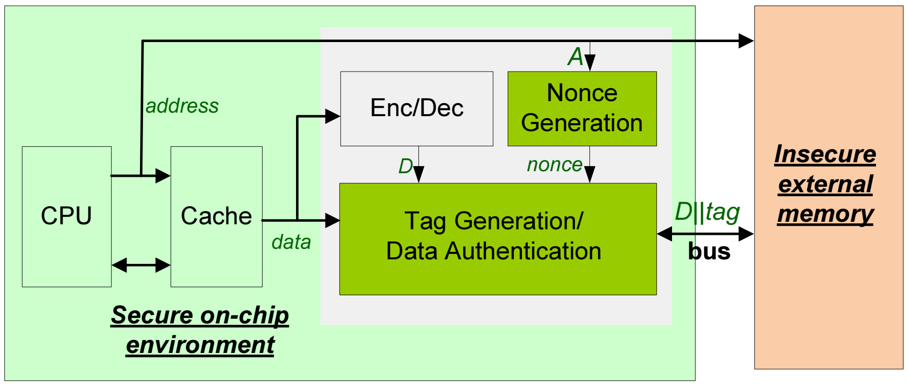

Figure 2: CETD System Overview

Figure 2 shows an overview of the system that has been proposed in [1]. It targets a system that has a secure processor chip and insecure off-chip memory. The off-chip memory can be accessed by attackers and the memory data can be read and altered. Therefore, the plaintext data is encrypted and the encrypted data is tagged. The encrypted data D and its tag are stored in the memory. When a data is required, its encrypted value and the related tag are fetched from the memory into the processor chip, then the tag value of the fetched data is re-calculated and compared with the tag obtained from the memory. If both values are the same, the data is authenticated and can be further decrypted for use. Otherwise, the data should be discarded.

The tag generation process is given in Figure 3(a), where the input encrypted data is divided into blocks with the size same as that of the tag. The process consists of three steps: 1) block segment shuffle 2) block bit rotation, and 3) block XOR.

The block segment shuffle is randomly choosing a pair of blocks and swapping bit-segments between the two blocks, as demonstrated in Figure 3(b), where 2-bit segments in blocks D_i and D_j are swapped. There can be many rounds of swaps. For each round, the swap can be performed on a different block pair and with a different segment size, and the segment position in the block can be also different. The block shuffle operation breaks the block boundaries, helping to counter the slice attack [1]. The second step, block rotation, further randomizes the input. The rotation is performed on individual blocks. An example is given in Figure 3(c), where a block is left-rotate-shifted 3 bits. For each rotation, the shift distance can be varied. In the final XOR step, blocks from the rotation are merged into one block as a tag value.

The operations in both the block shuffle and block rotation steps are controlled by a nonce value.

For each memory location, there are a dedicated counter and an associated random number. The nonce is generated by a block cipher that takes the input the access memory address A, the associated random number R, and the counter value C, as shown in Figure 4(a). The counter value is incremented for each update operation to the same location; when the counter reaches to its maximum, it will wrap around and a new random number is used, as demonstrated in Figure 4(b), which shows how the counter value and random number are changed for a sequence of memory update operations to memory location $A1$. The nonce value varies with the memory location and its counter value, as illustrated in Figure 4(c). Therefore, each tag generation is expected to use a separate nonce.

With the design shown in Figure 2, the possible attacks related to the tag design are of two types:

- *Forged Tag Attack*, where the genuine data/tag pair is replaced with a

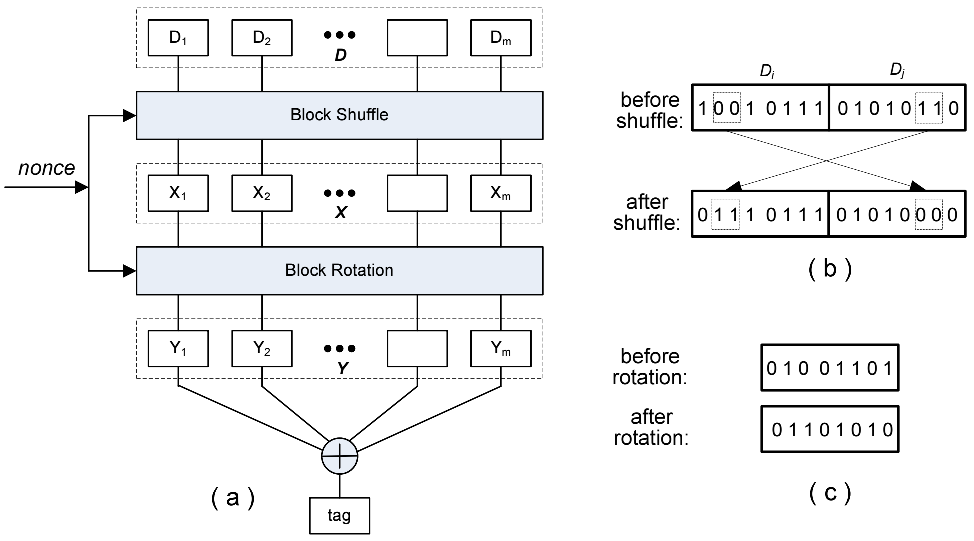

Figure 3: Tag Generation Process (a) diagram (b) shuffle example (c) rotation example

different data value and a forged tag, and

- *Replay Attack*, where the genuine data/tag pair is replaced with a previously observed valid data/tag pair.

The forged tag attack is basically related to the randomness of the tag value and the tag generation process, which has been discussed in [1] and will be further studied in a separate work. Here, our focus is on the replay attack, as will be presented in the next section.

3 Security Against Replay Attack

The replay attack includes the replay of data from previously observed data for the same memory location (i.e., the data was observed before but is now not in the memory) or the replay of data from other memory location (the data can still be there).

Based on the given design, the tag generation can be abstracted as a two-level function, f:

$$\begin{cases} tag & = f(D, nonce), \\ nonce & = enc(A, C, R); \end{cases} \tag{1}$$

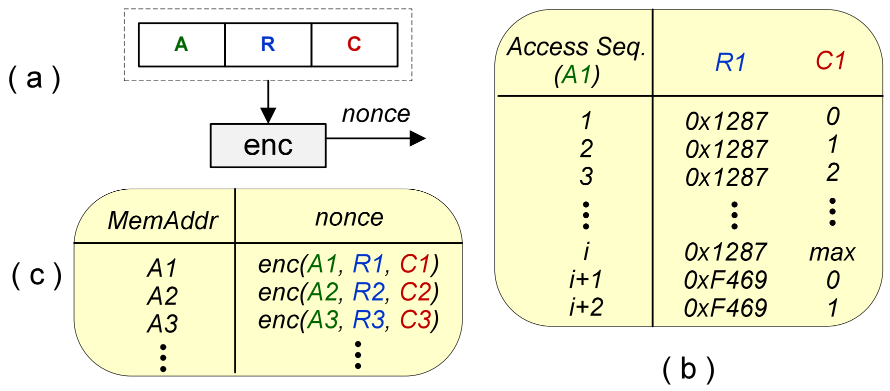

Figure 4: Nonce Generation (a) structural overview (b) change of random number (c) dedicated nonce values

where D is the data being tagged, A and C the memory location and the counter value associated with D, and R the related random number.

Since both D, and *nonce* are generated by a block cipher (such as AES), we assume the data (D) and the related nonce, are random and uniform distributed over their value spaces.

We use D_r and T_r to denote a replayed data and its old valid tag. When D_r is replayed, the on-chip authentication will calculate a tag value for D_r with the current (new) nonce and then compare the new tag with T_r, which was produced previously based on an old nonce. If both are same, the replayed data passes the authentication and the replay attack succeeds.

Therefore, we have the following security definition for the target system:

Statement 1: *The design is breakable by a replay attack if the tag, of the replayed data D_r, calculated with a new nonce is the same as that with the old nonce.* Namely, the design can be compromised if following is true:

$$f(D_r, enc(A_{new}, C_{new}, R_{new})) = f(D_r, enc(A_{old}, C_{old}, R_{old})), \tag{2}$$

which can be further divided into three collisions, and the design can be attacked by any of the three:

- **Collision** $CLS_{(C,R)}$ on the counter and random value pair (C, R), where a memory location uses a same pair of counter and random values at two different times; namely $(A, C_{new}, R_{new}) = (A, C_{old}, R_{old})$;

- **Collision** $CLS_{(enc)}$ on encryption function enc, where same nonce values were created with different A and/or different (C,R); namely, $enc(A_{new}, C_{new}, R_{new}) = enc(A_{old}, C_{old}, R_{old})$;
- **Collision** $CLS_{(tag-gen)}$ on the tag generation process, where the same tag values were generated with different nonces for a given replayed data; namely, $f(D_r, nonce_{new}) = f(D_r, nonce_{old})$, and $nonce_{new} \neq nonce_{old}$.

The first two types of collisions result in the collision of nonces; they are therefore collectively called **nonce collision** CLS_{nonce}.

Both the nonce collision and the tag generation process collision, and the success probability of the related replay attack are discussed in the following two subsections.

3.1 Nonce Collision

As stated above, the nonce collisions come from two types of collisions: $CLS_{(C,R)}$ and $CLS_{(enc)}$.

Assume the block cipher size is S_b, the counter size is S_{ct}, the random number size is S_{rn}, and the memory address size is S_A. All sizes are in bits. We have

$$S_b = S_{ct} + S_{rn} + S_A. \tag{3}$$

Collision $CLS_{(C,R)}$ allows for attacks of replaying data from a same memory location. The collision $CLS_{(C,R)}$ is only possible when a random number repeats a previous used value (i.e.,**random number collision**), as illustrated in Figure 5, where random number R_j repeats R_i. During the random number collision, if data at the memory location A_p at time t_{p1} is replayed at t_{p2} ($C_{p1} = C_{p2}$), the replay attack will not be detected by the system.

Assume the random number is uniform distributed. Given a replay, the probability of the $CLS_{(C,R)}$ collision is

$$Pr(CLS_{(C,R)}) = \frac{1}{2^{S_{rn}+S_{ct}}}, \tag{4}$$

where S_{rn} is the random number size and S_{ct} the counter size.

Collision CLS_{enc} is purely determined by the randomness of the cipher block. With a good block cipher, such as AES, if inputs are different, the

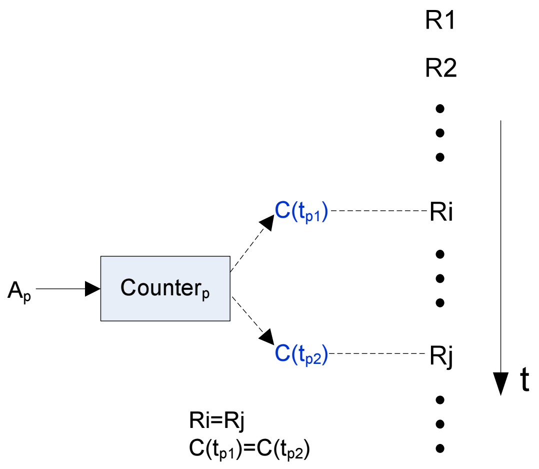

Figure 5: (C,R) Collision

outputs are different. Therefore, the collision probability of encryption is 0, namely,

$$Pr(CLS_{(enc)}) = 0. \tag{5}$$

The probability of nonce collision is

$$Pr(CLS_{(nonce)}) = Pr(CLS_{(C,R)}) + Pr(CLS_{(enc)}) = \frac{1}{2^{S_{rn}+S_{ct}}}. \tag{6}$$

With the nonce collision (i.e. $nonce_{new} = nonce_{old}$), the new tag of the replay data is the same as the old tag, and the related success probability

$$Pr(replay|CLS_{(nonce)}) = 1. \tag{7}$$

Therefore, the probability of the replay success based on the nonce collision is

$$\begin{aligned} Pr(replay_{(nonce)}) &= Pr(replay|CLS_{(nonce)}) * Pr(CLS_{(nonce)}) \\ &= \frac{1}{2^{S_{rn}+S_{ct}}} \\ &= \frac{1}{2^{S_b-A}}, \end{aligned} \tag{8}$$

where A is the memory space.

Formula (8) shows that given the size of the block cipher, the size of the memory should be controlled within a limit to reduce the nonce collision probability, hence the related replay attack.

3.2 Tag Generation Collision, $CLS_{(tag-gen)}$

The replay attack related to collision $CLS_{(tag-gen)}$ here is purely related to the three steps in the tag generation process: block shuffle, rotation, and XOR block merge operation. Refer to Figure 3, the inputs to the three steps are D, X and Y blocks. D is the data to be tagged, which is produced by a block cipher and is deemed as random; X is the internal output from shuffle, and Y is the output from the rotation.

The shuffle and rotation are controlled by the random nonce value. A nonce consists of multiple fields. We call the field to control shuffle the **s-field**, and the field to control block rotation, the **r-field**.

Based on the classification of the collision given before, here for the tag collision we consider two tag generations for a same input D_r but with different nonces: nonce1 and nonce2, $nonce1 \neq nonce2$. There are three possible cases that the two nonces can be different:

- CASE 1: s-field different, r-field same
- CASE 2: s-field same, r-field different
- CASE 3: s-field different, r-field different

In each case, the resulting X/Y blocks either are altered or remain unchanged as compared to their input block values. If unchanged, tag collisions will happen; If altered, tag collision can be regarded as random.

A few symbols that will be used in the following discussion are explained first here.

Assume the size of s-field and r-field are S_s and S_r, respectively; The size of a block is n bits. Since input data D is generated with the block cipher, the size of D is the size of block cipher, S_b. When the data is divided into m blocks, we have

$$n = \frac{S_b}{m}, \tag{9}$$

where S_b is the input data size.

Often n is chosen as a power of 2. Here we also treat n as

$$n = 2^f, \tag{10}$$

where f is a positive integer.

For m blocks, each block can be rotate-shifted up to n bits, and the size of r-field is

$$S_r = m * log_2 n; \tag{11}$$

Similarly, each shuffle can choose the segment size up to n bits and the segment in each of the two blocks can be positioned in n possible locations, therefore, the size of the s-field is

$$S_s = (3 log_2 n) * r, \tag{12}$$

where r is the number of shuffle rounds.

The success rate of replay in each case is discussed below.

3.2.1 CASE 1

The probability of CASE 1 (with different s-field values but a same r-field) is

$$Pr(CASE1) = \frac{2^{S_s} * (2^{S_s} - 1) * 2^{S_r}}{2^{2(S_s+S_r)}} = \frac{2^{S_s} - 1}{2^{S_s+S_r}}, \tag{13}$$

where the denominator $2^{2(S_s+S_r)}$ is the total control space for two tag generations; $2^{S_s} * (2^{S_s} - 1)$ and 2^{S_r} in the numerator are the possibilities to find two different s-field values and one r-field value. When, the s-field is large enough ($2^{S_s} >> 1$),

$$Pr(CASE1) \approx \frac{1}{2^{S_r}} = \frac{1}{2^{m * log_2 n}} = \frac{1}{n * 2^m}. \tag{14}$$

Consider one round shuffle between two blocks. For a k-bit segment in one block, there are n possible k-segments in the second block. The probability of finding a matched segment in the second block is $n * \frac{1}{2^k}$. The probability of the two matched segments to be selected in the shuffle operation is $\frac{1}{n^2}$. The probability of the output of a shuffle is the same as its input is

$$n * \frac{1}{2^k} * \frac{1}{n^2} = \frac{1}{n * 2^k}.$$

Since k is in a range of from 0 to $n-1$, the probability of retaining the same input value after r rounds of shuffle is

$$Pr(X = D|CASE1) = (\sum_{k=0}^{n-1} \frac{1}{n * 2^k})^r = \frac{2^r}{n^r}(1 - \frac{1}{2^n})^r. \tag{15}$$

When n is big enough ($\frac{1}{2^n} << 1$),

$$Pr(X = D|CASE1) \approx (2/n)^r. \tag{16}$$

Since the control values for rotation are same in CASE 1, the success probability of the replay attack under CASE 1 is therefore

$$\begin{aligned} Pr(replay_{(TG-case1)}) &= Pr(X = D|CASE1) * Pr(CASE1) \\ &\approx \frac{1}{n^{r+1} * 2^{m-r}}, \end{aligned} \tag{17}$$

where m is the number of blocks, n the block size in bits, and r the shuffle rounds. From Formula (9), the above formula can be written as

$$Pr(replay_{(TG-case1)}) \approx \frac{1}{n^{r+1} * 2^{S_b/n-r}}, \tag{18}$$

where S_b is the size of the block cipher. Formula (18) shows the probability is a convex function of n. We can find values of n and r for the minimum replay success probability based on CASE 1.

3.2.2 CASE 2

The probability of CASE 2 (with same s-field and different r-field) is

$$Pr(CASE2) = \frac{2^{S_s} * 2^{S_r} * (2^{S_r} - 1)}{2^{2(S_s+S_r)}} = \frac{2^{S_r} - 1}{2^{S_s+S_r}}. \tag{19}$$

When the r-field is big enough ($2^{S_r} >> 1$),

$$Pr(CASE2) =\approx \frac{1}{2^{S_s}} = \frac{1}{2^{3*(log_2 n)r}} = \frac{1}{n * 2^{3r}}. \tag{20}$$

In CASE 2, for two tag generations, the shuffle always keeps its inputs; only the rotation may produce different outputs due to different r-field values. Assume the difference of r-field values in the two tag generations is d for an input block. If the output blocks from both rotations are same, we need the input block to remain same after it is rotate-shifted d bits. For this, we have the following statement:

Statement 2: *For a block to remain same after the rotation of d bits, the block should contain repeated patterns. With d, one can find a p, such that*

$d = 2^p(2q+1)$. *The size of the repeated pattern is* 2^p *bits.*

Based on Statement 2, we can then derive the success probability of the replay attack under CASE 2 is

$$\begin{aligned} Pr(replay_{(TG-case2)}) &= Pr(Y = D|CASE2) * Pr(CASE2) \\ &\approx \frac{1}{2^{mn/2}} \frac{1}{n * 2^{3r}} = \frac{1}{n * 2^{3r+mn/2}} \qquad (21) \\ &= \frac{1}{n * 2^{3r+S_b/2}}, \qquad (22) \end{aligned}$$

where r is the shuffle rounds, and n the block size. From Formula (22), we can see that increasing n and r helps to counter the CASE2-based replay attacks.

The proof of Statement 2 and the derivation of Formula (22) are detailed in Appendix.

3.2.3 CASE 3

For CASE 3, both r-field and s-field are different. The probability of CASE 3 is

$$\begin{aligned} Pr(CASE3) &= 1 - (Pr(CASE1) + Pr(CASE3)) \\ &= \frac{2}{2^{S_s+S_r}} \\ &= \frac{2}{n^2 * 2^{3r+S_b/n}}. \qquad (23) \end{aligned}$$

With CASE 3, we take the shuffle and rotation results as random. The tag collision is mainly determined by the XOR reduction operation, which is discussed below.

We denote Y blocks to the XOR operation for tag Tag_a as Y_{a1}, Y_{a2}, $\cdots$, Y_{am}. If two tags, Tag_a and Tag_b, collided with a same value, we would have

$$Tag_a \oplus Tag_b = 0; \qquad (24)$$

Namely,

$$(\sum_{i=1}^{m} \oplus Y_{ai}) \oplus (\sum_{i=1}^{m} \oplus Y_{bi}) = 0, \qquad (25)$$

where $\sum_{i=1}^{m} \oplus$ stands for the bit-wise XOR of all listed block values.

The $2m$ Y blocks in Formula (25) can be grouped based on their values. The blocks with a same value are put in one set. For the set with an Even Number of Blocks, denoted as **ENB set**, their XOR results in 0 and those ENB sets can be cancelled in the XOR operation in Formula (25). We further merge these ENB sets into one group, denoted as $\widetilde{S}_0$, and leave the rest of sets (with the odd number of blocks, and called **ONB sets**) unchanged. Assume the number of the ONB sets is k; Note, k must be an even number, otherwise the total number of blocks from all sets will become odd. Also, k should be smaller or equal to $2m$. $k = 2$ means there are two sets each with an odd number of blocks, and $k = 2m$ means all $2m$ blocks are distinct from each other and each block forms one set. We use $\widetilde{S_k}$ to hold the k distinct blocks from the ONB sets. Figure 6 shows an example of, where input Y blocks are given in Figure 6(a) with their values in decimal being shown below each block. Based on their values, the eight blocks are grouped into five sets, as shown in figure (b). The first two sets each of two blocks go to group $\widetilde{S}_0$, the rest of two sets form another group $\widetilde{S}_2$ that contains two distinct blocks.

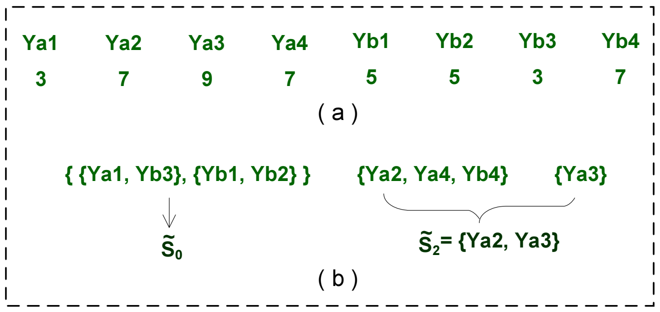

Figure 6: Example of Block Grouping (a) input Y blocks (b) four sets, two groups

The tag collision can be considered on the two basic sub cases (under CASE 3). We therefore name them as Case 31 and Case 32.

- Case 31: $\widetilde{S_k} \neq \phi, k \neq 0$, namely, the input Y blocks form some ONB sets $\widetilde{S_k}$ with each set having an odd number blocks, namely, and
- Case 32: $\widetilde{S_k} = \phi, k \neq 0$, namely, there are no ONB sets, all input Y blocks just form set $\widetilde{S}_0$.

Both the sub-cases may lead to tag collisions and the probability of two tag collisions under CASES 3 is determined by

$$\begin{aligned} & Pr(Taga = Tagb|CASE3) \\ = \; & Pr(Tag_a = Tag_b|Case31)Pr(Case31) + \\ & +Pr(Tag_a = Tag_b|Case32)Pr(Case32). \end{aligned} \tag{26}$$

And we can obtain the upper bound of the success probability of replay from CASE3 as

$$\begin{aligned} Pr(replay_{(TG-case3)}) &= Pr(Taga = Tagb|CASE3) * Pr(CASE3) \\ &\leq (\frac{1}{2^n - 2m + 1} + \frac{m}{2^{mn}}) * \frac{2}{n^2 * 2^{3r+S_b/n}} \\ &= (\frac{1}{2^n - 2S_b/n + 1} + \frac{S_b}{n * 2^{S_b/2}}) * \frac{2}{n^2 * 2^{3r+S_b/n}}. \end{aligned} \tag{27}$$

The detail of the derivation is given in Appendix.

The upper bound of the collision probability from the tag generation process is

$$\begin{aligned} Pr_{CLS(tag-gen)} &= Pr(replay_{(TG-case1)}) + Pr(replay_{(TG-case2)}) + Pr(replay_{(TG-case3)}) \\ &\leq \frac{1}{n^{r+1} * 2^{(S_b/n-r)}} + \frac{1}{n * 2^{3r+S_b/2}} + \\ &\quad +(\frac{1}{2^n - 2S_b/n + 1} + \frac{S_b}{n * 2^{S_b/2}}) * \frac{2}{n^2 * 2^{3r+S_b/n}} \end{aligned} \tag{28}$$

4 Simulations and Discussions

Based on the study shown in the previous section. We calculate the probability of replay attacks under different cases. The results are given below.

4.1 Simulation Setup and Results

5 Conclusions

6 Appendix

6.1 Proof of Statement 2

Proof:

For the block rotation, we can perceive a block as a bit wheel. The rotation of a block is turning the bit wheel. We use two wheels to demonstrate the d-bit rotation, as shown in Figure 7, where the inner wheel is the block before rotation, denoted by D_b, and the outer wheel for the block after rotation, denoted by D_a.

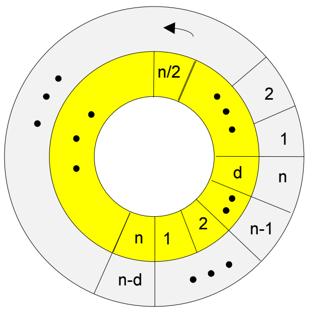

Figure 7: Block Rotated d Bits

Before rotation, the two blocks are same, namely,

$$D_b[i] = D_a[i], \qquad i = 1, \cdots, n; \tag{29}$$

where n is the block size.

For the block remaining same after rotation of d bits, we need

$$D_b[i+d] = D_a[i], \qquad i = 1, \cdots, n. \tag{30}$$

Therefore, we have **the first inference**:

$$D_b[i+d] = D_b[i], \qquad i = 1, \cdots, n. \tag{31}$$

Similarly, from Formula (29) $D_b[i+d] = D_a[i+d]$, and from Formula (30) $D_a[i+d] = D_b[i+d+d]$; hence we have **the second inference**:

$$D_b[i+d+d] = D_b[i], \qquad i = 1, \cdots, n. \tag{32}$$

We can repeat the process for different $D_b[i+h*d], h = 0, 1, \cdots$ and have a **normalized inference**:

$$D_b[i+h*d] = D_b[i], \quad i = 1, \cdots, n; \quad h = 0, 1, \cdots . \tag{33}$$

Since the block size is n bits, when $i + h * d > n$, the bit position should be wrapped around and its modulo of n should be used. Trying each h value, we can trace all bits in the block that have a same value as bit i. In fact, we can find a minimal h value, h_m, such that tracing h from 0 to h_m, we can find all bits with the same value, and $(i + h_m * d) modn = i$. We group those bit-s $(D_b[i+h*d], h = 0, 1, \cdots, h_m)$ into one group, called **identical-bit group**.

Now let us determine how many such bit groups exist for a given d, $d > 0$. Since any positive integer can be represented as the product of an odd number and a power of 2, we can write d as

$$d = 2^p(2q + 1), \tag{34}$$

where p and q are non-negative integers.

For an identical-bit group, $(i + h * d)$ mod n, $0 \leq h \leq h_m$, there is an integer g, such that

$$i + h_m * d - g * n = i, \tag{35}$$

Namely,

$$h_m * d - g * n = 0. \tag{36}$$

That is

$$h_m = g * n/d. \tag{37}$$

Assume here $n = 2^f$, as was given in Formula (10). From Formula (34), the above formula can be written as

$$h_m = \frac{g * 2^f}{2^p(2q + 1)}, \tag{38}$$

where $f > p$ since $n > d$. To ensure h_m to be an integer, g should be an odd number $k(2q+1), k = 1, 2, \cdots$, and $g = (2q+1)$ corresponds to the minimal h_m value:

$$h_m = \frac{(2q + 1) * 2^f}{2^p(2q + 1)} = 2^{f-p}. \tag{39}$$

Formula (39) gives the size of each bit group (how many bits in an identical-bit group); therefore, there are $n/h_m = 2^f/h_m = 2^p$ distinct groups.

When d is an odd number, $p = 0$, there is only one distinct group. When d is an even number, we will show that the adjacent 2^p bits in the block (D_a) are each in different identical-bit groups. Namely, we want to prove that

$$(i + h * d) modn \neq i + x, \tag{40}$$

for $1 \leq x < 2^p$.

To prove Formula (40), we apply the approach of *proofs by contradiction.* Assume there is an integer, h', $h' < h_m$, and an integer g' such that

$$(i + h' * d) mod n = i + h' * d - g'n = i + x. \tag{41}$$

Then

$$h' = \frac{x + g'n}{d} = \frac{x + g' * 2^f}{2^p(2q+1)} = \frac{x/2^p + g' * 2^{f-p}}{2q+1},$$

Since $x < 2^p$, h' is not an integer. Therefore, for any $x < 2^p$, Formula (41) is not true. Hence any adjacent 2^p bits in the block should be in different distinct groups. The 2^p groups hold repeated 2^p-bit segments.

We can, therefore, conclude that **For a block to remain same after the rotation of d bits, the block should contain repeated patterns. With d, one can find a p, such that $d = 2^p(2q + 1)$. The size of the repeated pattern is 2^p bits**.

End of Proof.

To demonstrate the repeat patterns with different d values, we use the 16-bit block as an example. When $d = 3$, $p = 0$; there is only one distinct group and the repeat pattern is of 1 bit. When $d = 6 = 2^1 * 3$, $p = 1$; there are two (2^1) distinct groups and the repeat pattern is of 2 bits, as shown in Figure 8, where bits in one distinct group is highlighted in a same color.

6.2 Derivation of Formula (22)

For all possible values in a block, the values with repeated patterns lead to unchanged block after rotation, which in turn causes the tag collision.

Now let us count the possible pattern values, v_i, for i-bit pattern.

For 1-bit pattern, there are two possible pattern values: 0 or 1; $v_1 = 2$.

For 2-bit pattern, there are 2^2 pattern values: '00','01','10','11'. Among them, '00' and '11' are actually covered by the 1-bit pattern; therefore, $v_2 = 2^2 - v_1 = 2$.

For 3-bit pattern, the possible pattern values are '000, 001, 010, 011, 100, 101, 110, 111';, excluding repeated patterns in the series, $v_3 = 2^3 - v_1 = 6$.

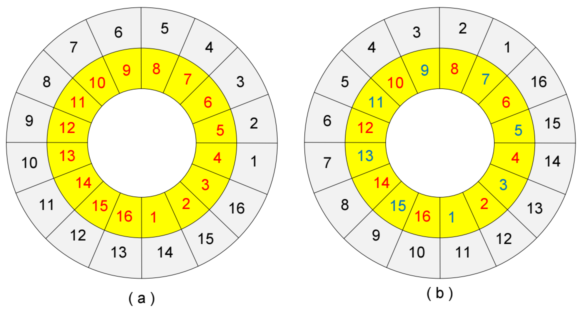

Figure 8: Block Rotated with Different Distance (a) d=3, 1-bit pattern (b)d=6, 2-bit pattern

For 4-bit pattern, the pattern values are: "0000, 0001, 0010, 0011, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111"; $v_4 = 2^4 - v_1 - v_2$.

With the similar way, we can find that for the k-bit pattern, $k = 1, \cdots, n/2$. There are 2^k possible values and

$$v_k = \begin{cases} 2^k - v_1 & \text{if k is odd and } k > 1; \\ 2^k - v_1 - v_{k/2} & \text{if k is even and } k > 2. \end{cases} \tag{42}$$

The total number of possible pattern values is

$$\begin{aligned} M &= \sum_{k=1}^{n/2} v_k && (43) \\ &< (\sum_{k=0}^{n/2} (2^k - v_1)) = 2^{n/2} - n, && (44) \end{aligned}$$

where n is the block size.

For each repeat pattern, the probability of the matching control r-field value is

The probability of values with repeated patterns over the whole value space is $M/2^n$. The probability of a block remaining same after the rotation is $\frac{2^{n/2}-n}{2^n} < \frac{1}{2^{n/2}}$ Therefore, the probability of m blocks to remain same after rotation can be approximated as,

$$Pr(Y = D|CASE2) = (\frac{1}{2^{n/2}})^m = \frac{1}{2^{mn/2}}. \tag{45}$$

The success probability of the replay attack under CASE 2 is

$$\begin{aligned} Pr(replay_{(TG-case2)}) &= Pr(Y = C|CASE2) * Pr(CASE2) \\ &\approx \frac{1}{2^{mn/2}} \frac{1}{n * 2^{3r}} = \frac{1}{n * 2^{3r+mn/2}}, \end{aligned} \tag{46}$$

where r is the shuffle rounds, n the block size, and m the number of blocks.

6.3 Derivation of Formula (27)

With Case 31, for Formula (25) to be true, the XOR of the k blocks should be 0, that is

$$\sum_{Y_i \in \widetilde{S_k}} \oplus Y_i = 0. \tag{47}$$

The number of all possible combinations of distinct k blocks in the n-bit value domain is

$$2^n \cdot (2^n - 1) \cdot (2^n - 2) \cdots (2^n - k + 1).$$

Without lose of generality, we form k-blocks that satisfy Formula (47) as follows. Randomly choose a (k-1) distinct blocks in the n-bit domain. The XOR result of the (k-1) blocks is either one of these (k-1) blocks or a different block value. If the result is one of the (k-1) blocks, the selection is discarded since they cannot form a set of k distinct blocks; Otherwise the (k-1) blocks together with the block from their XOR result are selected as a distinct k-block set.

As can be seen, there are $2^n \cdot (2^n - 1) \cdot (2^n - 2) \cdots (2^n - k + 2)$ different (k-1)-distinct blocks, some of them can lead to k-distinct blocks, while some may not. Therefore, the number of possible k-distinct blocks is smaller than this $2^n \cdot (2^n - 1) \cdot (2^n - 2) \cdots (2^n - k + 2)$. The probability of tag collision under Case 31 with k-distinct blocks is

$$Pr(Tag_a = Tag_b|(Case31(k)) \leq \frac{2^n \cdot (2^n - 1) \cdot (2^n - 2) \cdots (2^n - k + 2)}{2^n \cdot (2^n - 1) \cdot (2^n - 2) \cdots (2^n - k + 1)} = \frac{1}{2^n - k + 1}. \tag{48}$$

Since k can be any even number between 2 and $2m$, the probability of two tag collision is

$$\begin{aligned}
& Pr(Tag_a = Tag_b|Case31)Pr(Case31) \\
= & \sum_{k=2,4,\ldots,2m} Pr(Tag_a = Tag_b|(Case31(k))Pr(Case31(k)) \\
\leq & \sum_{k=2,4,\ldots,2m} 1/(2^n - k + 1)Pr(Case31(k)) \\
\leq & \sum_{k=2,4,\ldots,2m} 1/(2^n - 2m + 1)Pr(Case31(k)) \\
\leq & \ 1/(2^n - 2m + 1).
\end{aligned} \tag{49}$$

For Case 32, the tag collision is 100%, namely

$$Pr(Tag_a = Tag_b|Case32) = 1. \tag{50}$$

We only need to find out the probability of Case 32. Again we group blocks in their values. Assume there are k sets; each set has even number of identical blocks.
When $k = 1$, there is only one set that has $2m$ identical block values. The number of possible values, n_1, is

$$n_1 = 2^n.$$

When $k = 2$, there are two distinct blocks, the number of possible block value combinations, n_2, is

$$n_2 = 2^n * (2^n - 1).$$

When $k = m$, there are m distinct sets, each having two identical blocks. The possible combinations are

$$n_m = 2^n * (2^n - 1) * \cdots * (2^n - m + 1).$$

The total number of possibilities for Case 32, $N2$, is

$$\begin{aligned} N2 &= \sum_{i=1}^{m} n_i \\ &= \sum_{i=1}^{m} 2^n * ((2^n - 1) * \cdots * (2^n - i + 1)) \\ &\leq \sum_{i=1}^{m} (2^n)^m = m * (2^n)^m. \end{aligned}$$

Therefore, the probability of Case 32 can be given as

$$Pr(Case2) = \frac{N2}{(2^n)^{2m}} \leq \frac{m * (2^n)^m}{(2^n)^{2m}} = \frac{m}{2^{mn}}. \qquad (51)$$

The probability of the tag collision under CASE 3 is therefore

$$Pr(Taga = Tagb|CASE3) \leq \frac{1}{2^n - 2m + 1} + \frac{m}{2^{mn}}, \qquad (52)$$

where m is the number of blocks and the n the tag size.

The probability of CASE 3 is smaller than 1. Therefore the success probability of replay from CASE3 is

$$Pr_{rply-case3} \leq Pr(Taga = Tagb|CASE3) \leq \frac{1}{2^n - 2m + 1} + \frac{m}{2^{mn}}. \qquad (53)$$

Formula (27) shows that when the input block values are uniform-randomly distributed, the tag collision probability is inversely, in the exponential rate, related to the tag size. When the tag size is big enough, the tag collision probability is close to zero. For example, when m=2, n=64, the probability is reaching to $1/2^{64}$ (and vanishingly small).

References

[1] M. Hong, H. Guo, and X. Hu. A cost-effective tag design for memory data authentication in embedded systems. In *Proceedings of the 2012 international conference on Compilers, architectures and synthesis for embedded systems*, 2012.