Forgery and Key-Recovery Attacks on CAESAR Candidate Marble

Thomas Fuhr, Gaëtan Leurent, Valentin Suder

Forgery and Key-Recovery Attacks on CAESAR Candidate Marble

Thomas Fuhr Gaëtan Leurent Valentin Suder

January 11, 2015

The CAESAR competition, which started in 2014, aims at providing a new standard of authenticated encryption. In this paper, we perform an analysis of the candidate Marble. We show a generic attack on the Marble mode of operation (independent of the E transformations), where we recover the whitening key L, and perform forgeries using 2^{64} chosen plaintext queries. Considering the specific internal primitives used in Marble (composed of 4 AES rounds), we also show how to recover the secret key using 2^{32} additional decryption queries, in the decryption-misuse setting (where we can decipher plaintexts without valid tags).

Keywords: CAESAR competition, authenticated encryption, Marble, forgery, key-recovery.

1 Introduction

The purpose of an *Authenticated Encryption* scheme is to provide both privacy and integrity with a single cryptographic algorithm. The main goal of the CAESAR competition, launched in 2014, is the design of Authenticated Encryption schemes which are designed for different applications and environments in order to provide alternatives to current options such as AES-GCM [**?**]. This resulted in the submission of 57 candidates that demand to be analyzed carefully. In this paper, we provide a security analysis of the AES-based candidate Marble [**?**].

Marble has very strong security claims: it claims to offer 128-bit security even after large amount of data have been encrypted with the same key. In addition, Marble doesn't use nonces, and the security claim even holds if unverified plaintext are released, *i.e.* the adversary can request the decryption of a ciphertext C without knowing a valid tag corresponding to C (decryption-misuse oracle).

The only result presented so far on Marble is a cipher-text distinguisher with complexity 2^{64}, similar to the distinguisher against the counter mode [**?**].

Attack	Data	Time
Recover L	2^{64} CP	2^{64}
Forgery	2^{64} CP	2^{64}
Key-recovery	2^{64} CP + 2^{32} CC[1]	2^{64}

Table 1: Our results.

Our results In this paper we show that Marble is not as secure as claimed by its author. Namely, we describe algorithms that perform the following attacks.

First, we found a technique to recover the whitening key L derived from the main encryption key K, using 2^{65} chosen plaintexts. When L is known the security offered by Marble crumbles and we show a forgery attack using a single extra encryption query. This shows that Marble is no longer secure when the amount of data encrypted with a key exceeds the birthday bound. Once L is known, it is possible to recover K using decryption-misuse queries, *i.e.* obtain plaintexts from ciphertexts without a valid tag. Our attack requires about 2^{33} queries and its time complexity is 2^{64}. All the queries contain a small number of message blocks.

Our results are summarised in Table 1.

Outline of the paper In Section 2, we give a short description of the Marble authenticated encryption algorithm. In section 3, we show how to recover L and describe our forgery attack. Finally, we demonstrate in Section 4 how to recover the encryption key K from decryption-misuse queries.

2 Description of Marble

Marble is an authenticated encryption algorithm designed by Guo [**?**] with key-length and tag-length of both 128 bits. A plaintext and its associated data are divided into blocks of 128 bits and are then proceeded consecutively. Its internal permutation is based on a modified version of the AES block cipher. Unlike other authenticated encryption algorithms, Marble does not require a nonce.

An overview of Marble is depicted in Figure 1. The Marble mode of operation makes use of 2 128-bit chaining variables s_1 and s_2, initialised with constants $const_1$ and $const_2$. The associated data and the plaintext are padded independently, so both resulting fields A and P can be divided into 128 bit blocks. We do not describe the padding function here, as it does not affect our attacks. We will denote a message to encrypt by $(AD \mid M)$, where AD is a vector containing l_A 128-bit blocks of associated data and M is a vector containing l_M 128-bit blocks of plaintexts.

The internal primitive used is a modified block cipher, as intermediate values of the block are combined with the incoming chaining variables. Formally, the primitive uses 3

[1]The chosen ciphertexts use the decryption-misuse model.

internal keyed permutations E_1, E_2 and E_3 and processes 128-bit blocks as follows. On input (P, s_1, s_2), (C, s'_1, s'_2) is defined as

$$\begin{aligned} X &= E_{1K}(P) \\ (X', s'_1) &= (3X + s_1, X + s_1) \\ Y &= E_{2K}(X') \\ (Y', s'_2) &= (3Y + s_2, Y + s_2) \\ C &= E_{3K}(Y') \end{aligned}$$

Note that additions and multiplications are performed in the binary Galois Field $GF(2^{128})$ defined by the primitive polynomial $x^{128} + x^7 + x^2 + x + 1$. Furthermore, polynomials $\Sigma a_i X^i$ are denoted by the integers $\Sigma a_i 2^i$. Therefore, please note that additions and multiplications on such objects have to be interpreted as operations in the binary field (and not on the integer ring) and have to be handled carefully.

In the case of Marble, each one of the three permutations E_1, E_2 and E_3, is composed with 4 full-round of AES (*i.e.* SubByte, ShiftRow, MixColumn and AddRoundKey). One can notice that no key addition is performed at the beginning of those permiutations. 12 subkeys are therefore required. A 128-bit master key K is derived into 11 subkeys using the AES-128 key-schedule algorithm. The master key itself is used as the 12th subkey. For more information about the AES block cipher, we refer to [**?**].

The Marble encryption then works as follows. First, a mask L is derived from the key K by encrypting a constant $const_3$ (which also sets key-dependent $s_1[0], s_2[0]$). For each associated data block A_i ($i \geq 1$), a pre-whitening key is defined as $2^{i-1}3^2L$. For each plaintext block M_i, a pre-whitening key and a post-whitening key are defined as 2^iL and $2^{i-1}3L$. These blocks are processed iteratively, starting with associated data, as follows:

1. Addition (*i.e.* xor) of the pre-whitening key;
2. Application of the internal primitive;
3. For plaintext blocks, application of the post whitening key, which results in ciphertext blocks.

Finally, the tag is computed by encrypting an extra block defined as the sum of all plaintext blocks and all encrypted additional data blocks, with pre-whitening key $2^{\ell_M}7L$ and post-whitening key $2^{\ell_M-1}3L$.

3 A universal forgery attack on Marble

In this section, we first describe a method to find the mask L using about 2^{65} chosen plaintext queries. Then, we use this knowledge to compute forgeries. Our attack enables to modify the associated data field in a way that affects neither the ciphertext nor the authentication tag. It can therefore be used to compute universal forgeries in a chosen plaintext setting.

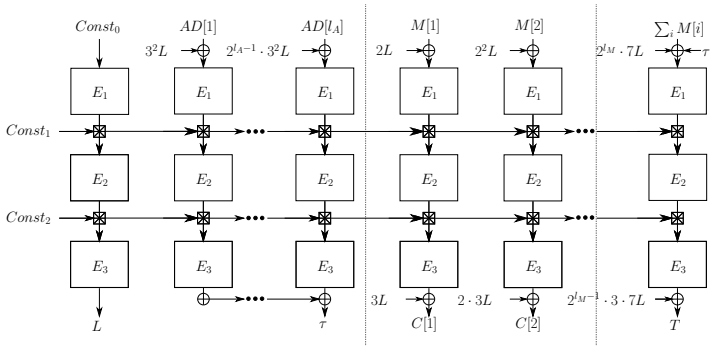


Figure 1: General design of Marble.

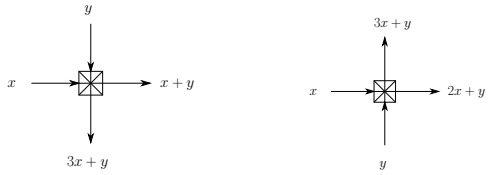


Figure 2: The `TRANS` operation.

3.1 Recover the mask L

The main idea of the attack is to build a pair of message $M \neq M'$ so that the inputs to the E_1 functions are the same for both messages. This is possible if M and M' have the same total length, but the associated data and message parts have different lengths. When the inputs to E_1 collide, all the intermediate computations collide, and we can detect this event on the resulting ciphertexts. Please note that as different multiples of L are used for post-whitening, this operation is more tricky than detecting a collision on ciphertexts. In the following we use 2 blocks of AD and 1 block of message for M, but 1 block of AD and 2 blocks of message for M'.

More precisely, we encrypt messages M_α and M'_β, for different values $\alpha, \beta \in \mathbb{F}_{2^{128}}$, defined as follows (where $A \in \mathbb{F}_{2^{128}}$ is a constant value):

- $M_\alpha = (AD[1], AD[2] \mid M[1]) = (A, 8\alpha \mid 6\alpha)$;
- $M'_\beta = (AD[1] \mid M'[1], M'[2]) = (A \mid 8\beta, 6\beta)$.

In the following, we consider sets of 2^{64} values α and β so that $\alpha \oplus \beta$ covers all values in $\mathbb{F}_{2^{128}}$. The inputs to the E_1 layer will be respectively (we note that $3^2 = 5$ in $\mathbb{F}_{2^{128}}$):

$$\begin{array}{llll} x_1 = A \oplus 5L & x_2 = 8\alpha \oplus 10L & x_3 = 6\alpha \oplus 2L & \text{for } M_\alpha \\ x'_1 = A \oplus 5L & x'_2 = 8\beta \oplus 2L & x'_3 = 6\beta \oplus 4L & \text{for } M'_\beta \end{array}$$

In particular, we have:

$$x_1 \oplus x'_1 = 0 \qquad x_2 \oplus x'_2 = 8 \cdot (\alpha \oplus \beta \oplus L) \qquad x_3 \oplus x'_3 = 6 \cdot (\alpha \oplus \beta \oplus L)$$

Therefore, the inputs to E_1 collide when $\alpha \oplus \beta = L$.

We denote the output of the E_3 layer as y_i (respectively y'_i), and the corresponding ciphertexts as $C_\alpha[1]$ (respectively $(C'_\beta[1], C'_\beta[2])$). We have:

$$C_\alpha[1] = y_3 \oplus 3L \qquad C'_\beta[1] = y'_2 \oplus 3L \qquad C'_\beta[2] = y'_3 \oplus 6L$$

In particular, if $\alpha \oplus \beta = L$, we have $x_i = x'_i$ for $i \leq 3$, therefore $y_i = y'_i$ for $i \leq 3$, and $C_\alpha[1] \oplus C'_\beta[2] = 5L$ (since $3 \oplus 6 = 5$). In order to detect this event efficiently we match the set of values $\{C_\alpha[1] \oplus 5\alpha\}$ and $\{C'_\beta[2] \oplus 5\beta\}$. When $\alpha \oplus \beta = L$, we have a match, and we can easily filter false positives using a new message pair with a different value of the constant A. The full algorithm is given by Algorithm 1, using 2^{65} short encryption queries.

Algorithm 1 Recover L from an encryption oracle $\mathcal{E}$.

```
H ← ∅                                                    ▷ H is a hash table
for α ∈ {0, 1, ..., 2^64 − 1} do
    (C[1] | T) ← E(0, 8α | 6α)
    H{C[1] ⊕ 5α} ← α
end for
for β ∈ {0, 2^64, ..., 2^128 − 2^64} do
    (C'[1], C'[2] | T) ← E(0 | 8β, 6β)
    if H{C'[2] ⊕ 5β} exists then
        α ← H{C'[2] ⊕ 5β}
        (D[1] | T) ← E(1, 8α | 6α)
        (D'[1], D'[2] | T) ← E(1 | 8β, 6β)
        if D[1] ⊕ 5α = D'[2] ⊕ 5β then
            return α ⊕ β
        end if
    end if
end for
```

3.2 Computing forgeries on Marble without whitening keys

Once we have retrieved L, we can consider a simplified description of Marble where the masks are removed, as depicted in Figure 3. In its mask-less description, Marble possesses an interesting property as described in Figure 4: a series of identical input blocks X has a periodic effect on the internal state.

Indeed, if we let $E_1(X) = u$, $E_2(3S_1 \oplus u) = v$ and $E_2(3S_1 \oplus 2u) = w$, it is easy to see that after encrypting 4 blocks X, the internal states S_1 and S_2 remain unchanged. Furthermore, if we use a series of 8 consecutive identical associated data blocks X, the effect on τ also cancels out. This leads to a universal forgery attack: for any associated data AD and plaintext M, the adversary computes the masked value B of a chunk of 8 identical blocks of associated data after AD and queries the encryption oracle on $((A, B) \mid M)$. The answer $(C \mid T)$ to that query is also a valid ciphertext for

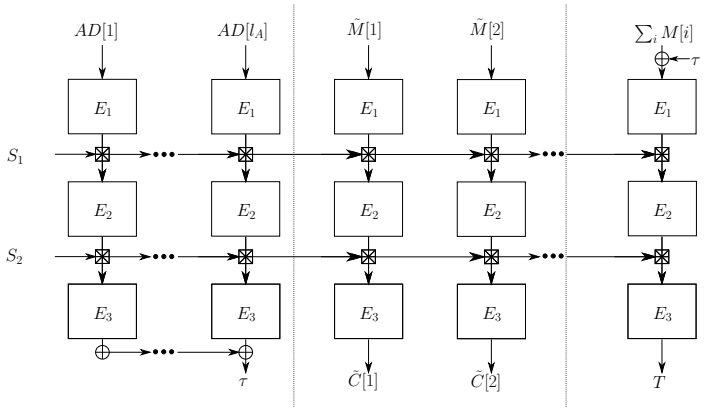


Figure 3: Mask-less description of Marble. S_1 and S_2 are unknown key-dependent values.

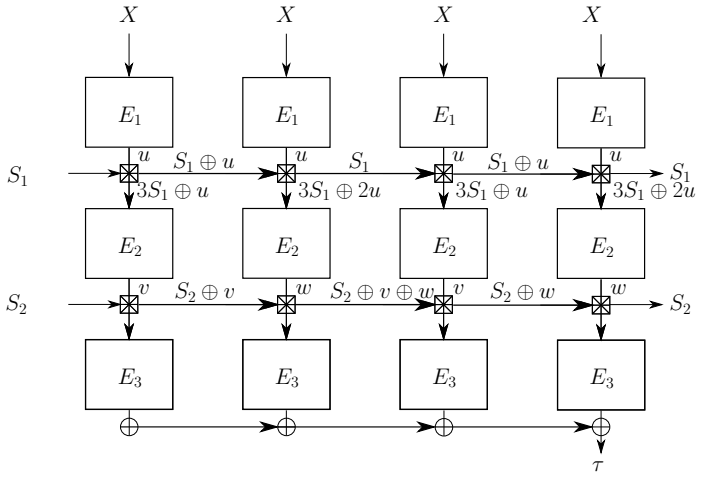


Figure 4: Collision on the internal state of the associated data.

$(AD \mid M)$, therefore the adversary can return $(C \mid T)$ as a forgery. The attack is given as Algorithm 2.

4 Key-recovery attack

We now show how to recover the master key once the mask L has been determined. In order to simplify the description of the attack, we now focus on the maskless variant of Marble; however the full attack can easily be adapted to the full version of Marble with a known mask.

The main idea is to collect pairs of messages with a fixed difference in some internal state variables. This will allow us to attack a reduced cipher composed by 4 AES rounds followed by 4 inverse AES rounds rather than a 12-round AES (see details below). Moreover, we apply this strategy to E_1 rather that to E_3 because the whitening key of

Algorithm 2 Compute the ciphertext $(C \mid T)$ from $(AD \mid M)$ using known L.

$B \leftarrow \left(2^l \cdot 3^2 \cdot L\right)_{l=l_A}^{l_A+7}$
$(C \mid T) \leftarrow \mathcal{AE}_K((AD, B) \mid M)$ ▷ Encryption oracle call
return $(C \mid T)$

E_1 is directly derived from L. Since L is known, the first AES round of E_1 is key-independent. Therefore we can peel it off, and attack only 3 forward rounds and 3 inverse rounds. However, this requires us to use decryption queries, but we can't forge valid tags for an arbitrary ciphertext yet, so we use a decryption-misuse oracle.

4.1 Gathering pairs

The first step is to collect pairs of plaintext blocks that have the same difference in the S_1 lane (after the permutation E_1). In order to construct such plaintexts, we build pairs of ciphertexts with specific differences and values. More precisely, we consider pairs of messages as follows (with the same associated datas AD):

$$\widetilde{C}_x = (AD \mid 0, 0^{120}, 0, 0, 0, 0, 0, 0, 0, 0, x) \qquad C_x[i] = \widetilde{C}_x[i] \oplus 2^{i-1} \cdot 3 \cdot L$$
$$\widetilde{C}'_x = (AD \mid 1, 0^{120}, 1, 0, 0, 0, 0, 1, 1, 1, x) \qquad C'_x[i] = \widetilde{C}'_x[i] \oplus 2^{i-1} \cdot 3 \cdot L$$

where 0 and 1 are constant one-block values and x takes a different value for each pair. We decrypt these pairs and we collect the final plaintext blocks.

We now study the differences in the S_2 lane (before the permutation E_3). Using the definition of the `TRANS` operation as given in Figure 2, S_2 is updated as follows during decryption:

$$S_2[i+1] = 2 \cdot S_2[i] \oplus E_3^{-1}(\widetilde{C}[i])$$

With the messages C_x and C'_x, we have

$$S_2[129] = 2^{129} \cdot S_2[0] \oplus (1 \oplus 2 \oplus \cdots \oplus 2^{128}) \cdot A,$$
$$S'_2[129] = 2^{129} \cdot S_2[0] \oplus (1 \oplus 2 \oplus \cdots \oplus 2^{128}) \cdot A \oplus (2^0 \oplus 2^1 \oplus 2^2 \oplus 2^7 \oplus 2^{128}) \cdot (A \oplus B),$$

where $A = E_3^{-1}(0)$ and $B = E_3^{-1}(1)$. Since $2^{128} = 2^0 \oplus 2^1 \oplus 2^2 \oplus 2^7$, we have $S_2[129] = S'_2[129]$. This is shown in Figure 4, where $\delta = A \oplus B$.

We now consider the final plaintext block given by the decryption oracle.

$$\begin{aligned}
\widetilde{P}_x[130] &= P_x[130] \oplus 2^{130} \cdot L \\
&= E_1^{-1}\left(E_2^{-1}\left(E_3^{-1}(x) \oplus 3 \cdot S_2[129]\right) \oplus 3 \cdot S_1[129]\right) \\
\widetilde{P}'_x[130] &= P'_x[130] \oplus 2^{130} \cdot L \\
&= E_1^{-1}\left(E_2^{-1}\left(E_3^{-1}(x) \oplus 3 \cdot S_2[129]\right) \oplus 3 \cdot S'_1[129]\right)
\end{aligned}$$

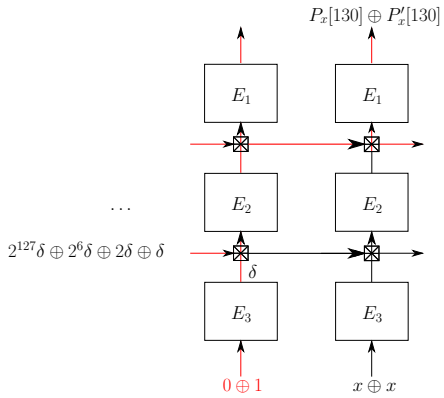
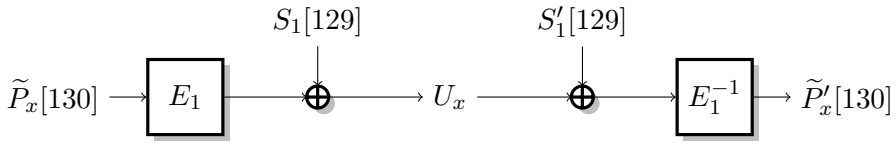


Figure 5: Difference propagation in decryption. A red arrow means that there is a fixed unknown difference. A black arrow means that the difference is null.

With $U_x = E_2^{-1}\left(E_3^{-1}(x) \oplus 3 \cdot S_2[129]\right)$, we have

$$\widetilde{P}_x[130] = E_1^{-1}(U_x) \oplus 3 \cdot S_1[129]$$
$$\widetilde{P}'_x[130] = E_1^{-1}(U_x) \oplus 3 \cdot S'_1[129]$$

Therefore, the pair $\widetilde{P}_x[130], \widetilde{P}'_x[130]$ can be seen as a plaintext/ciphertext pair for a cipher with 4 AES rounds, a middle key $S_1[129] \oplus S'_1[129]$, and 4 inverse AES rounds:

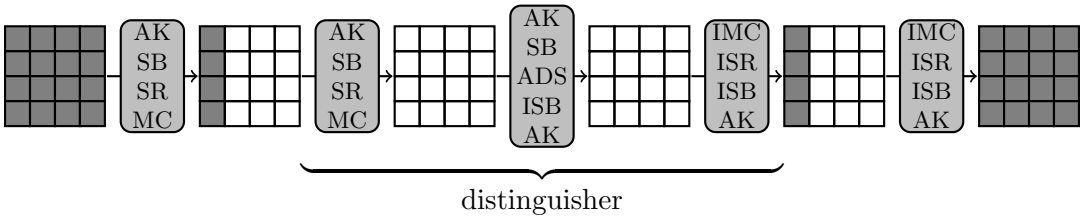


In addition, we can peel off the outer rounds because there is no whitening key in E_1.

4.2 Extracting the key

We must now extract the key of a reduced cipher with 3 AES rounds, and 3 inverse AES rounds. First, we notice that the middle ShiftRow and MixColumn operations can be removed, if we transform the middle key. Therefore the middle rounds are reduced to bytewise operations: AddRoundKey, SubByte, AddDeltaS, InverseSubByte, AddRoundKey. This can be seen as a key-dependent Sbox layer.

The first step of our attack is to guess a diagonal of the first round key, which allows to compute a column after the first round and before the last round. Next we build a distinguisher for the middle rounds.



The middle rounds have only one MixColumn operation, and one InverseMixColumn without byte shuffling in between. Therefore it can be seen as four parallel 32-bit functions, acting on each diagonal (similar to the Super-SBox technique [**?**]). For each function, 1 input byte and 1 output byte are known. We now focus on a single function F. Intuitively, if we have many partial input-output pairs, we should detect some correlation between the inputs and output. This can be formalized using linear cryptanalysis, but the best linear trail is not the same for every key because of the key-dependant middle SBoxes. Instead, we focus on a property that does not require to know in advance which values are correlated, and works for any function F.

We denote the known input byte as α and the known output byte as β, and we use 2^{16} counter $c_{\alpha,\beta}$ to count how many times each pair (α, β) occurs with D availabe data. If the key guess is correct, there is some correlation between α and β, which results in a higher value for some counters. In order to detect this, we compute the sample variance s^2 of the 2^{16} counters:

$$s^2 = 2^{-16} \sum_{\alpha,\beta} (c_{\alpha,\beta} - \overline{c_{\alpha,\beta}})^2, \quad \text{where } \overline{c_{\alpha,\beta}} = 2^{-16} \sum_{\alpha,\beta} c_{\alpha,\beta}.$$

We expect that s^2 is higher when the key guess is correct.

More precisely, we model the counters using random variables $C_{\alpha,\beta}$, and the sample variance as S^2 for a wrong key guess, and S_*^2 for the right key. Our goal is to show that when D is large enough, we have $\Pr[S^2 > S_*^2]$ negligible, *i.e.* the correct key is ranked first.

Wrong key guess. If the key guess is wrong, we expect that α and β behave as independent random variables. Therefore, the counters follow a multinomial distribution with parameters $p = 2^{-16}$ and $n = D$. In particular, the average value of the counters is $\mathrm{E}[C_{\alpha,\beta}] = 2^{-16}D$ and the variance is $\mathrm{Var}[C_{\alpha,\beta}] = 2^{-16}(1 - 2^{-16})D \approx 2^{-16}D$, while the covariance is $\mathrm{Var}[C_{\alpha,\beta}, C_{\alpha',\beta'}] = -2^{-32}D$ for $(\alpha, \beta) \neq (\alpha', \beta')$.

Correct key. Let us now assume that the key guess is correct, *i.e.* the pairs (α, β) are valid partial input/output for F. We focus on a single counter $C_{\alpha,\beta}$, and we consider F as a random permutation. We first count the number of inputs/output pairs to F matching (α, β):

$$X_{\alpha,\beta} = |\{a, b, c \colon F(\alpha, a, b, c) = (\beta, ?, ?, ?)\}|.$$

We have $\mathrm{E}[X_{\alpha,\beta}] = 2^{16}$ for a random permutation and we further conjecture that $\mathrm{Var}[X_{\alpha,\beta}] = 2^{16}$. Next we study $Y_{\alpha,a,b,c}$ the number of times each value α, a, b, c is reached

with D samples. The $Y_{\alpha,a,b,c}$ follow a multinomial distribution, with: $\mathrm{E}[Y_{\alpha,a,b,c}] = 2^{-32}D$, $\mathrm{Var}[Y_{\alpha,a,b,c}] = 2^{-32}(1-2^{-32})D$, $\mathrm{Var}[Y_{\alpha,a,b,c}, Y_{\alpha',a',b',c'}] = -2^{-64}D$.

Finally, $C_{\alpha,\beta}$ can be written as the sum of $X_{\alpha,\beta}$ random variables $Y_{\alpha,a,b,c}$. From the law of total expectation and the law of total variance, we have [2]:

$$\begin{aligned}
\mathrm{E}[C_{\alpha,\beta}] &= \mathrm{E}(X_{\alpha,\beta})\,\mathrm{E}(Y_{\alpha,a,b,c}) = 2^{-16}D \\
\mathrm{Var}[C_{\alpha,\beta}] &= \mathrm{E}(X_{\alpha,\beta})\,\mathrm{Var}(Y_{\alpha,a,b,c}) + \mathrm{E}(X_{\alpha,\beta})(\mathrm{E}(X_{\alpha,\beta})-1)\,\mathrm{Var}(Y_{\alpha,a,b,c}, Y_{\alpha',a',b',c'}) \\
&\qquad + \mathrm{E}(Y_{\alpha,a,b,c})^2\,\mathrm{Var}(X_{\alpha,\beta}) \\
&= 2^{-16}(1-2^{-32})D - 2^{-32}(1-2^{-16})D + 2^{-48}D^2 \\
&= 2^{-16}D - 2^{-32}D + 2^{-48}D^2
\end{aligned}$$

Distinguisher. We obtain an efficient distinguisher with $D = 2^{32}$: for a wrong key guess, the variance of the counter is about 2^{16}, but it is about 2^{17} for the right key. In order to evaluate the probability that the correct key is ranked first, we must evaluate how far the sample variance s^2 is from the true variance $\mathrm{Var}[C_{\alpha,\beta}]$.

For a wrong key guess, if we use a single counter and repeat the experiment with 2^{16} independent sets of D plaintexts, each counter $C_{\alpha,\beta}$ follows a binomial distribution with parameters D and $p = 2^{-16}$. If we approximate them as a normal distributions with parameters $\mu = 2^{-16}D$ and $\sigma = \sqrt{2^{-16}(1-2^{16})D}$, we know that the distribution of the sample variance S^2 for a wrong key guess follows a χ^2 distribution [3]:

$$S^2 \sim \frac{\sigma^2}{(n-1)}\chi^2_{n-1} \sim 2^{-32}D\chi^2_{2^{16}-1}$$

In particular, we have $\Pr[S_0^2 > 2^{16} + 2^{12}] < 2^{-90}$, therefore we don't expect that the sample variance of a wrong key is above $2^{16} + 2^{12}$. In practice, we use a single set of D plaintexts, and we evaluate the sample variance of the 2^{16} counters; our experiments show that the distribution is close to a χ^2 distribution (the maximum value of s^2 with 2^{16} samples was $2^{16} + 1420$).

For the right key, we don't have an analytic expression of the distribution of the sample variance, but we can perform experiments. Our experiments show that with very high probability $S_*^2 > 2^{16} + 2^{12}$, as seen in Figure 6. Of our 2^{16} experiments, the minimum value of s_*^2 was $102795 \approx 2^{16}+2^{15}$. Using $D = 2^{32}$, we have a large margin and we expect the attack to work with significantly less data, but recovering L will be the bottleneck of the attack.

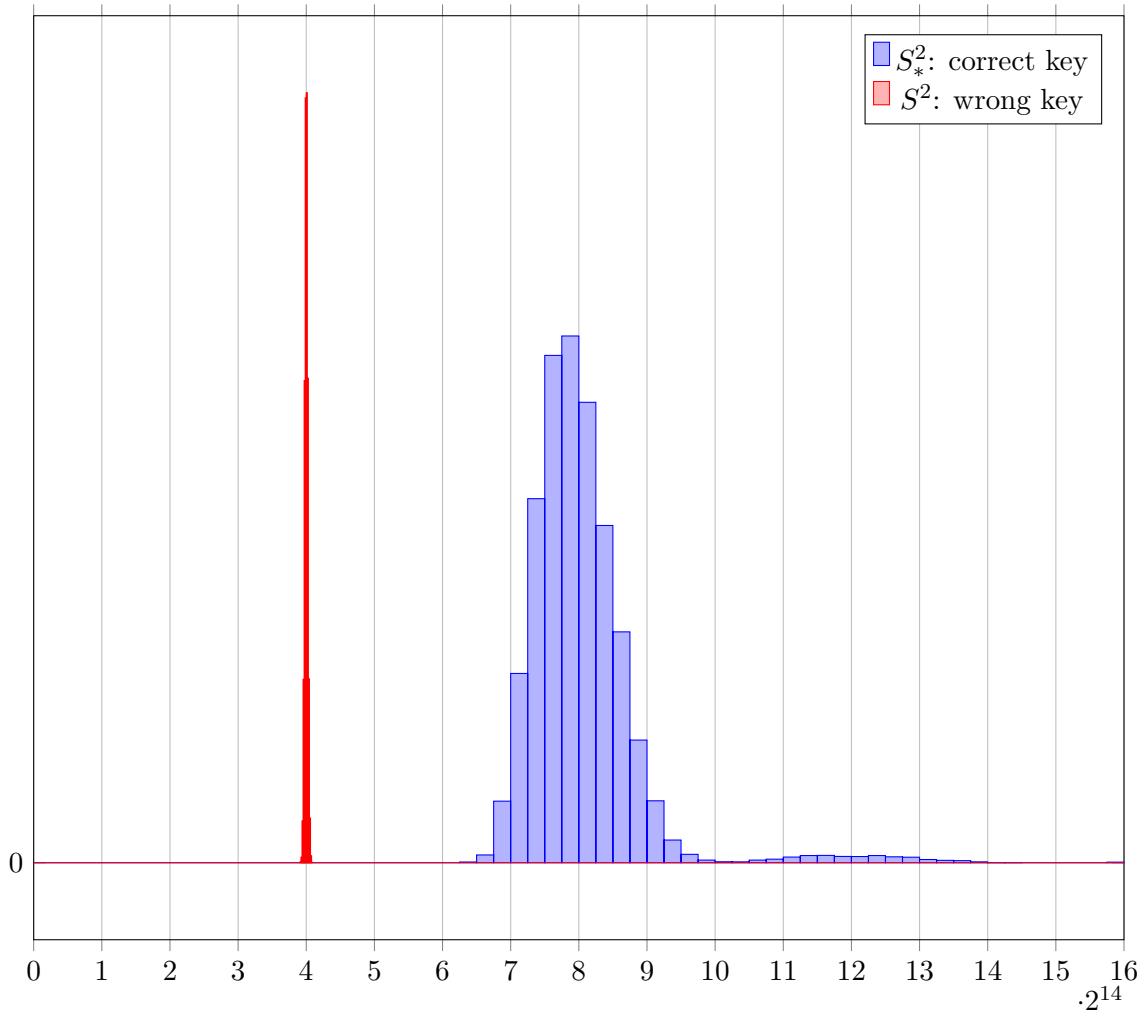
5 Conclusion

Our results show that Marble does not provide the security feature claimed by the author, *i.e.* beyond birthday bound security and decryption-misuse resistance. Its robustness is

[2] We follow the analysis in `http://en.wikipedia.org/wiki/Compound_Poisson_distribution`

[3] See `http://en.wikipedia.org/wiki/Variance#Distribution_of_the_sample_variance`

Figure 6: Experimental results: distribution of the sample variance S^2 and S_*^2 with $D = 2^{32}$ (2^{16} experiments with random keys).



however comparable to those of classical modes of operations for block ciphers, such as GCM or the composition of CBC with HMAC.

Nevertheless, we can conclude from our attacks that once usage requirements are relaxed, the security of Marble collapses faster than the security of other algorithms. Surprisingly, this can even lead a full key recovery.

It seems that adding some domain separation in the state between the processing of the associated data and of the message would avoid our attacks, but a thorough analysis would be necessary to ensure that the resulting construction is secure.

In addition our key-recovery attack suggests that 4 AES rounds in the E functions are insufficient if the adversary can find a shortcut to target two E functions instead of three.