Analysis of the CAESAR Candidate Silver

Jérémy Jean[1], Yu Sasaki[1,2], and Lei Wang[1,3]

[1] Nanyang Technological University, Singapore

[2] NTT Secure Platform Laboratories, Tokyo, Japan

[3] Shanghai Jiao Tong University, China

JJean@ntu.edu.sg, sasaki.yu@lab.ntt.co.jp, wanglei@cs.sjtu.edu.cn

Abstract. In this paper, we present the first third-party cryptanalysis against the authenticated encryption scheme Silver. In high-level, Silver builds a tweakable block cipher by tweaking `AES-128` with a dedicated method and performs a similar computation as `OCB3` to achieve 128-bit security for both of integrity and confidentiality in nonce-respecting model. Besides, by modifying the tag generation of `OCB3`, some robustness against nonce-repeating adversaries is claimed. We first present a forgery attack against 8 (out of 10) rounds with 2^{111} blocks of queries in the nonce-respecting model. The attack exploits a weakness of the dedicated `AES` tweaking method of Silver. Then, we present several attacks in the nonce-repeating model. Those include 1) a forgery against full Silver with $2^{49.46}$ blocks of queries which matches a conservative security claim by the designers, 2) a plaintext recovery against full Silver with a single query and 3) a key recovery against 8 rounds with 2^{111} blocks of queries. In particular, the plaintext recovery breaks the security claim by the designers. Considering that the current best key recovery for plain `AES-128` is up to seven rounds, Silver lowers the security margin of `AES` due to its tweaking method. The attacks have been partially implemented and experimentally verified.

Keywords: Silver, CAESAR, Authenticated Encryption, Forgery, Plaintext Recovery, Key Recovery.

1 Introduction

An authenticated encryption is a symmetric-key cryptographic scheme that provides both of integrity and confidentiality at one time. Currently the CAESAR competition [1] is being conducted to determine a portfolio of authenticated encryptions, and research development for authenticated encryption deserves careful attention.

Security and efficiency are obviously important factors of authenticated encryption designs. Thanks to the `AES-NI`, which is a set of instructions in Intel's CPU, basing the scheme on block cipher `AES` [3] is a promising way of designing authenticated encryptions. On the other hand, the block

size of AES (128 bits) is often too small as a next-generation cryptographic scheme because security bound in many cases can only be proven up to a half of the block size. As a consequence, designing authenticated encryption schemes with large enough security while maintaining the efficiency advantages of AES-NI is a challenging topic.

One possible direction is designing some structure on top of AES to construct a tweakable block cipher [12]. The OCB mode designed by Rogaway [15] shows that a tweakable block cipher with n-bit block size enables to design an authenticated encryption with n-bit security. Several CAESAR candidates were designed in this line [5, 6, 7, 13], including our target Silver [13].

As mentioned above, Silver [13] designed by Penazzi and Montes proposes a dedicated way of tweaking AES-128 to provide 128-bit security for both integrity and confidentiality in the nonce-respecting model. The way of tweaking AES-128 is a little bit complicated and thus a more careful security analysis is necessary. So far, no third-party analysis has been provided.

Silver provides some robustness against nonce repetition. The following argument is claimed for integrity and confidentiality against nonce-repeating adversaries: "*It is safe to assume that a forgery cannot be made with probability greater than, say,* 2^{-50} *for tags of length* 128", and "*any attack against Silver should be readily converted into an attack on AES-ECB. Thus, although there is loss of indistinguishability, the loss of confidentiality is not catastrophic, and it simply reduces to the loss one would be prepared to accept when using ECB.*" Thus, cryptanalysis in the nonce-repeating model is also important to examine the designers' claims.

Apparently a unique design feature of Silver is the way of tweaking AES-128. In each block, the tweak value only impacts to subkeys. Namely, when all the eleven subkeys are specified, computation between plaintext and ciphertext is exactly the same as plain AES. Subkeys in each block are generated with a key K, a nonce N and a tweak tw that depends on the block position. The algorithm is briefly explained as follows. First, $\alpha \leftarrow \mathtt{AES}_K(N)$ is computed and then $(k_0, \ldots, k_{10}) \leftarrow \mathtt{AES}^{KS}(K)$ and $(\alpha_0, \ldots, \alpha_{10}) \leftarrow \mathtt{AES}^{KS}(\alpha)$ are computed, where $\mathtt{AES}^{KS}$ is the key schedule of AES-128. Additionally, an intermediate 128-bit value γ is computed from α. Intuitively, subkeys for the i-th message or associated data block are generated as follows (a complete specification appears in Section 2).

$$\begin{cases} sk_j \leftarrow k_j \oplus \alpha_j & \text{for } j = 0, 2, 3, 4, 6, 7, 8, 10, \\ sk_j \leftarrow k_j \oplus \alpha_j \oplus (\alpha + i \cdot \gamma) & \text{for } j = 1, 5, 9, \end{cases} \tag{1}$$

where '+' is the modular addition operated in 64 bits each. The multiplication by block index i introduces different impact among different block positions. The value $a + i \cdot \gamma$ is injected every four rounds to exploit the good diffusion property of 4-round `AES`.

Another unique design feature of Silver is the tag generation. Intuitively, the tag is the encryption of $\Sigma_A \oplus \Sigma_P \oplus \Sigma_C$. Here, Σ_A is the result of processing associate data which is similar to `OCB3` [10] and Σ_P is the message checksum which is the same as `OCB3`. Then, Σ_C is computed as $\bigoplus_i (C_i + \alpha + i \cdot \gamma)$ with C_i as the i-th ciphertext block. This additional checksum makes the essential difference from `OCB3`, and thus analyzing its security impact is important.

Our Contributions. We first present reduced-round analysis in the nonce-respecting model. We show a forgery against eight rounds with 2^{111} blocks of queries. Then, we explain several attacks in the nonce-repeating model. The first result is a forgery against full Silver with a complexity of $2^{49.46}$ blocks of queries. The second result is a plaintext recovery, i.e. breaking confidentiality of full Silver with a single query. The last result is a key recovery against eight rounds, which makes use of the above 8-round forgery as a tool and the bottleneck of the complexity is in that part, i.e. 2^{111} blocks of queries.

In general, differential cryptanalysis is hard to apply in the nonce-respecting model. To avoid this problem, our 8-round forgery in the nonce-respecting model exploits the internal (higher-order) difference [14], i.e. difference between the blocks of a single plaintext. As shown in Eq. (1), computations in different blocks only differ in the tweak value $i \cdot \gamma$, which is only injected in three subkeys. Thus, many subkeys have no difference ($\Delta sk_0 = 0, \Delta sk_2 = 0$ etc). Such sparse tweak injections allow us to mount an internal differential cryptanalysis even in the nonce-respecting model. By analyzing 256 consecutive blocks, the tweak value assumes the values $\gamma, 2\gamma, \ldots, 255\gamma$. Then, by forcing γ to have a single bit to one (and 127 zeros), an integral (8th-order differential) cryptanalysis [2,9] is applied.

Our forgery attack in the nonce-repeating model exploits the tag generation structure, which is the encryption of $\Sigma_P \oplus \Sigma_C$ (for empty associated data). Let n be the block size, i.e. $n = 128$ for Silver. We observe that the modular addition to compute Σ_C allows us to compute the sum of the x least significant bits (LSBs), where $x \in \{1, 2, \ldots, n\}$, independently of the $n - x$ most significant bits (MSBs). We first make $2^{n/3}$ queries under the same nonce to generate the first message block pair (P_1, P_1') such that the $2n/3$ LSBs of P_1 and P_1' are colliding and the

$2n/3$ LSBs of C_1 and C'_1 are also colliding. The same is iterated until the $n/3$-th block. Now, we have $2^{n/3}$ possible combinations of plaintext blocks, in which any of them produces the same $2n/3$ LSBs of $\Sigma_P \oplus \Sigma_C$ and one pair will produce the same $n/3$ MSBs. With this pair of messages producing the same $\Sigma_P \oplus \Sigma_C$, a forgery can be mounted by the length extension attack, i.e. appending the same message block p to each of two messages. The number of queries is about $2^{n/3}$ and each query consists of $n/3$ blocks. Thus, the data complexity is about $n/3 \cdot 2^{n/3}$, which is $2^{48.08}$ for $n = 128$. By increasing the success probability and optimizing attack procedure, the complexity becomes $2^{49.46}$ message blocks with success probability 0.36, which is almost the same as the one for the ordinary birthday paradox. The designers claim a conservative 50-bit security against forgery in the nonce-repeating model. Here, the choice of 50 bits is very unclear without any reasoning. Our attack shows that their claim is no longer conservative, i.e. forgery in the nonce-repeating model can be mounted with a complexity at most 2^{50}.

Our plaintext-recovery attack exploits the weakness of the domain separation when the last message block is not full. For any ciphertext $C_1 \| \cdots \| C_{N-1} \| C_N$ for some N where $|C_N| < 128$, the corresponding last block of the plaintext P_N is recovered with a single encryption query under the same nonce. Considering that the `ECB` mode never allows such a plaintext-recovery attack, it breaks the security claim by the designers, who state that "*any attack against Silver should be readily converted into an attack on AES-ECB*".

Our key-recovery attack makes use of the fact that Σ_C in the tag generation is computed by modular addition. We first generate a tag collision with 1-block messages (and empty associated data) by making 2^{64} queries under the same nonce. For the colliding pair, we can directly observe the ciphertext difference ΔC. The tag collision indicates $\Delta\Sigma_P \oplus \Delta\Sigma_C = 0$, where $\Sigma_P = P$ for 1 block message, and thus we can recover $\Delta\Sigma_C$. For a 1-block message, Σ_C is computed as $\Sigma_C = C + (\alpha + \gamma)$. From ΔC and $\Delta\Sigma_C$, we recover the secret value $(\alpha + \gamma)$ similarly with the analysis by Lipmaa and Moriai [11] about XOR difference propagation through the modular addition. We notice that the current best key recovery attack on plain `AES-128` [4] is up to seven rounds. In this sense, Silver lowers the security margin of `AES-128` due to its tweaking method.

Our attacks exploit several design aspects of Silver such as the dedicated `AES-128` tweaking method, tag generation structure, combination of XOR and modular addition, domain separation for special treatment, and so on. We believe that the presented analysis in this paper will bring

Table 1: Summary of the cryptanalysis of Silver presented in this paper.

Model	Rounds	Type	#Queries	Reference
Nonce respecting	4	Forgery	2^{79}	Section 3.1
	8	Forgery	2^{111}	Section 3.2
Nonce repeating	Full	Forgery	$2^{49.46}$	Section 4.1
	Full	Plaintext Recovery	1	Section 4.2
	8	Key Recovery	2^{111}	Section 4.3

useful feedback for the design of authenticated encryption schemes based on tweaked `AES`. Our results are summarized in Table 1.

2 Description of Silver

Silver is an authenticated encryption scheme relying on a tweakable block cipher derived from the `AES-128` block cipher E. We note that it can be regarded as an instance of the TWEAKEY framework introduced in [8] to construct tweakable block ciphers, where the tweaked values are injected by XORing together with the subkeys.

We describe here the design of Silver with the set of parameters recommended by the designers [13]. The original description being a bit hard to follow, we introduce new notations and figures to describe the cipher. Four inputs are processed by the encryption algorithm: a 128-bit public message number N, a 128-bit secret key k, possibly empty associated data A and a plaintext P. Those values are encrypted and authenticated in the form of a ciphertext C and a 128-bit tag T. From (N, A, C, T), the decryption algorithm produces the plaintext P if the tag T is verified, and $\perp$ otherwise.

Notations. We denote the byte length of P by b_P, b_A refers to the byte length of A, and $g = b_A \,||\, b_P$ encodes the two byte lengths of A and P in a 128-bit value. The XOR operation is denoted by $\oplus$. We represent the 64-bit most significant bits of a 128-bit value x by x^L, and its 64-bit less significant bits by x^R. We consider that $x = x^L \,||\, x^R$, where $||$ denotes the concatenation. More generally, we refer to the n least significant bits of x by $\lfloor x \rfloor_n$, and similarly to its n most significant by $\lceil x \rceil_n$. We denote the empty string by ϵ. Modular addition represented by $+$ is performed on 128-bit values in $(\mathbb{Z}/2^{64}\mathbb{Z}) \times (\mathbb{Z}/2^{64}\mathbb{Z})$, where the two 64-bit halves of the

values are added independently. Finally, $\overline{x}$ denotes the value x with the least significant bit forced to 1, i.e. $\overline{x} = x \vee 1$.

Initialization. Before processing the inputs, Silver requires an initialization step with one block cipher call, and two calls to the `AES-128` key scheduling algorithm. The block cipher call is parameterized by the secret key k and transforms N into an internal secret value that we denote α, i.e. $\alpha = E_k(N)$. Then, the `AES-128` key schedule produces two sequences of eleven subkeys from both the secret k, and the value α: $(k_0, \ldots, k_{10})$ and $(\alpha_0, \ldots, \alpha_{10})$, respectively. The two sets of subkeys are used in the function denoted KS, which combines the subkeys with a 128-bit input value S: $KS(S) = (u_0, \ldots, u_{10})$, with:

$$
\begin{aligned}
u_0 &= k_0 \oplus \alpha_1, & u_4 &= k_4 \oplus \alpha_4, & u_8 &= k_8 \oplus \alpha_8,\\
u_1 &= k_1 \oplus (\alpha + S), & u_5 &= k_5 \oplus \alpha_5 \oplus (\alpha + S), & u_9 &= k_9 \oplus (\alpha + S),\\
u_2 &= k_2 \oplus \alpha_2, & u_6 &= k_6 \oplus \alpha_6, & u_{10} &= k_{10} \oplus \alpha_{10}.\\
u_3 &= k_3 \oplus \alpha_3, & u_7 &= k_7 \oplus \alpha_7,
\end{aligned}
$$

Finally, two secret counters are also initialized: γ_A later used to process the associated data, and γ used during the plaintext encryption:

$$\gamma = \overline{\alpha_9^L} \,||\, \overline{\alpha_9^R}, \qquad \gamma_A = \overline{\alpha_9^L} \,||\, 0^{64}.$$

The tweakable block cipher used in Silver replaces the original `AES-128` subkeys $(k_0, \ldots, k_{10})$ by the subkeys $KS(S)$ for a given tweak value S.

Associated Data. Similarly to several other authenticated encryption schemes [5, 6, 7, 10], Silver first computes a MAC Σ_A on the associated data A, and then uses this value internally to authenticate A. The process is described on Figure 1. First, A is divided into t blocks of 128 bits, possibly using the 10^* padding. Then, it applies the tweaked `AES` with subkeys $KS(i \cdot \gamma_A)$ independently on each block i, and XORs the t results to produce the checksum Σ_A. Note that if the last block has been padded, the tweak input is 0 to produce subkeys $KS(0)$ for the last padded block.[4]

Encryption. To encrypt the plaintext P, we first divide it into blocks of 128 bits, say s blocks, the last one possibly non-full. The tweaked `AES-128`

[4] This special treatment of the last partial block, in particular the special tweak input value 0, will be later exploited in our forgery attacks in Section 3.

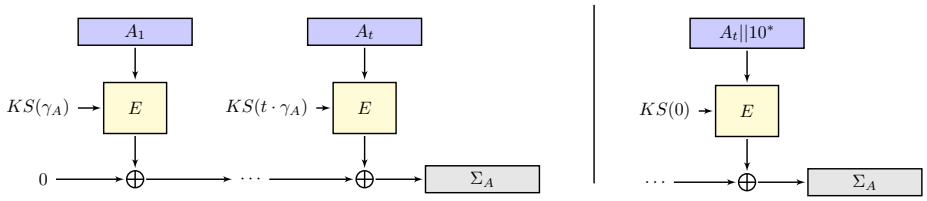

Figure 1: Associated data. There are t blocks in A. If the last block is not full, pad it using the 10^* padding. The checksum produced is Σ_A.

denoted E is used to transform each block P_i to ciphertext block C_i using the tweaked subkeys produced by $KS(i \cdot \gamma)$. Then, a checksum Σ_P is computed from the XOR of all the plaintext blocks, and a checksum Σ_C is computed from modularly masked value of the ciphertext blocks (see Figure 2, left), namely:

$$\Sigma_P = \bigoplus_{i=1}^{s} P_i, \quad \text{and} \quad \Sigma_C = \bigoplus_{i=1}^{s} (C_i + \alpha + i \cdot \gamma).$$

In the event that the last block is not full and contains $0 < l < 16$ bytes (see Figure 2, right), one first generates a mask μ from the encryption of $b_P \,||\, b_P$ under subkeys $KS(s \cdot \gamma)$ and XORs it to the actual partial block P_s to produce the partial ciphertext block C_s. The unused bits from the mask μ are appended to P_s to produce a new 128-bit block, whose last byte is replaced by l, which is encrypted to produce another checksum $\Sigma_{P'}$.

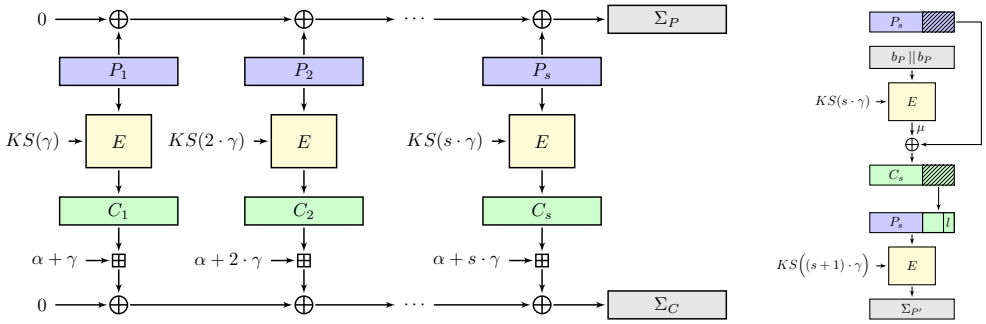

Figure 2: Encryption. Σ_P is the checksum over the full input blocks, Σ_C is a checksum over the ciphertext blocks shifted by modular additions. The last partial block (if any) is processed with two calls to E.

Tag Generation. To produce the tag T, one first XORs all the four (possibly null) checksums Σ_A, Σ_P, Σ_C and $\Sigma_{P'}$ to get Σ. Then, the tag T results from the encryption of Σ using E parameterized by the subkeys $\pi(KS(g))$, where the permutation π changes the order of the subkeys: $\pi = (2, 9, 3, 4, 6, 1, 7, 8, 10, 5, 0)$.

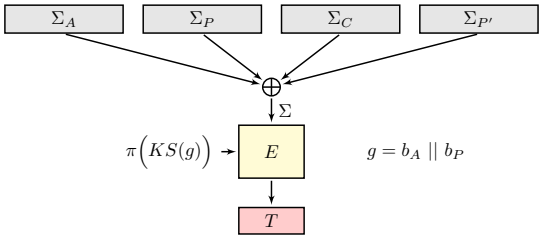

Figure 3: Tag generation. The tag consists of the encryption of the XOR of the four checksums Σ_A, Σ_P, Σ_C and $\Sigma_{P'}$.

3 Nonce-Respecting Analysis

In this section, we propose an attack on reduced-round Silver against a nonce-respecting adversary. Our goal is to exploit the well-known integral property of the `AES` [2,3] by using the counter computation in the associated data. Ultimately, we show how to select associated data A comprised of 256 blocks $A = (A_1, \ldots, A_{255}, A_0)$ such that the checksum Σ_A equals zero. The following demonstrates that this behavior occurs with probability higher than 2^{-128} and can be used to replace A by A' producing the same $\Sigma_A = 0$, hence leading to a forgery attack.

We start by describing a simpler 4-round attack and later extend it to eight rounds.

3.1 Forgery on 4-Round Silver

In Section 2, we have introduced a counter value γ_A used to compute Σ_A from the associated data. We recall that γ_A is defined from a 128-bit value α_9 by $\gamma_A = \overline{\alpha_9^L} \,||\, 0^{64}$, in which $\overline{\alpha_9^L}$ overwrites the LSB of α_9^L with a single bit 1. Therefore, with probability 2^{-63}, $\gamma_A = 0^{63}\,1 \,||\, 0^{64}$. If this holds, then the sequence of tweaks in the associated data computation

with the non-full last block depicted on Figure 4 would be:

$$\begin{aligned}
\gamma_A &= 0^{56}\,00000001 \,||\, 0^{64},\\
2\cdot\gamma_A &= 0^{56}\,00000010 \,||\, 0^{64},\\
3\cdot\gamma_A &= 0^{56}\,00000011 \,||\, 0^{64},\\
&\vdots\\
255\cdot\gamma_A &= 0^{56}\,11111111 \,||\, 0^{64},\\
0 &= 0^{56}\,00000000 \,||\, 0^{64}.
\end{aligned}$$

Additionally, we require that eight bits from α are fixed to zero, so that the modular additions with the counter values $i \cdot \gamma_A$ performed in the KS function do not propagate any carry. Indeed, $KS(i \cdot \gamma_A)$ computes $i \cdot \gamma_A + \alpha$, and if the eight least significant bits of α^L are zeros, then no carry is propagated in the addition. This restriction on α occurs with probability 2^{-8}.

To summarize, with probability 2^{-71}, the 256 values $c_i = i \cdot \gamma_A + \alpha$ for $i \in [0, 255]$ take all the 256 possible values on the eight least significant bits of c_i^L. To use this property, we consider a 256-block associated data $A = A_1||\cdots||A_{255}||A_0 = f(B)$, where:

$$\begin{aligned}
A_0 &= B,\\
\forall\, i \in \{1,\ldots,255\}, \quad A_i &= B \,||\, 10^7,
\end{aligned}$$

with B a random 120-bit value. This definition of A ensures that all full blocks are equal after the 10^* padding (see Figure 4).

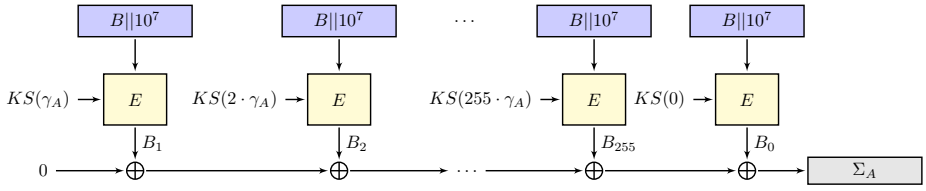

Figure 4: We consider $t = 256$ blocks in A such that the 256 inputs to the block cipher calls are all equal.

In this context, the 256 equal blocks are processed with counter values carrying an integral-like property, which can be observed in the checksum value Σ_A. We recall that we consider a reduced variant of E where we apply only four rounds. We explain now why the resulting Σ_A equals

zero by analyzing the propagation of the integral property in the different blocks. The following Figure 5 shows the status of each byte in the state during the encryption across the 256 blocks. Namely, a byte marked C means that the same value appears for this byte in each of the 256 states, an A stands for a byte having the 256 different values in the 256 states, and a 0 means that the XOR of the bytes from the 256 states produces zero.

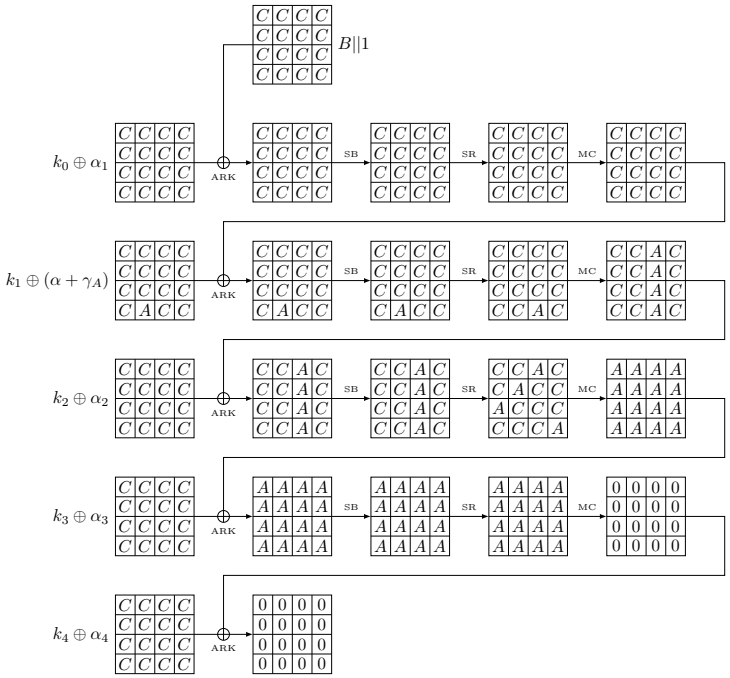

Figure 5: Integral property in the associated data of Silver processed by the 4-round block cipher E.

From the well-known integral property of the `AES` structure, the XOR of the 256 outputs produces zero in the 16 bytes after four rounds, with a difference introduced in the second subkey. As a result, the associated data of this particular A produces $\Sigma_A = 0$ with probability 2^{-71}.

Therefore, after querying the encryption oracle 2^{71} times with distinct nonce N, we expect that one query $(N, A, M = \epsilon)$ yields the ciphertext and tag pair (C, T) with the internal $\Sigma_A = 0$. Once this event happens, any subsequent decryption (N, A', C, T) query under the same nonce but with different associated data $A' = f(B')$ for a random 120-bit $B' \neq B$

would be a forgery. Indeed, the integral property still holds for the different input B', and the intermediate checksum Σ_A for the modified associated data would also be zero. Hence, the same tag has to be produced.

The procedure of the attack is described in Algorithm 1. We have experimentally verified that it works as expected assuming the event with probability 2^{-71} holds. The attack requires a total of queries of $2^{71+8} = 2^{79}$ blocks.

Algorithm 1 – Forgery for 4-round Silver.

$A \xleftarrow{\$} \{0,1\}^{120}$
$B \xleftarrow{\$} \{0,1\}^{120}$ ▷ Need to ensure $A \neq B$
$AD \leftarrow (A||10^7)^{255} \,||\, A$ ▷ $|AD| = b_A = 2^{12} - 1$
$AD' \leftarrow (B||10^7)^{255} \,||\, B$
for $i = 1 \rightarrow 2^{71}$ **do**
  $N_i \leftarrow \{0,1\}^{128}$ ▷ Pick a random nonce
  $(C = \epsilon, T_i) \leftarrow \text{Silver}(N_i, AD, M = \epsilon)$ ▷ 4-round encryption query
  $P \leftarrow \text{Silver}^{-1}(N_i, AD', C, T_i)$ ▷ 4-round decryption query
  if $P \neq \perp$ **then return** (N_i, AD', C, T_i) ▷ Success with probability 2^{-71}

3.2 Forgery on 8-Round Silver

The idea behind the 8-round forgery attack on Silver is essentially the same as the one for four rounds, except that we make sure that the pre-added four rounds do not alter the integral property. We still want to find a 256-block associated data value such that $\Sigma_A = 0$. The main difference here is the choice of these blocks, as the integral property does not propagate over eight rounds.

However, on the nine subkeys added in the eight rounds, only two carry the tweak input, which is different for each block of associated data, namely, the second and the fifth subkeys. Hence, if we choose 256 blocks A_i such that the internal states after the second subkey addition are the same for all the 256 blocks, then all the states would also be equal before the fifth subkey addition. Then, we know that the XOR of all the outputs after the last subkey addition yields zero, as in the previous case.

We are then left with the problem of choosing 256 blocks A_i such that the same state value X is reached after one-round encryption under

subkeys $u_0 = k_0 \oplus \alpha_1$ and $u_1 = k_1 \oplus (\alpha + i \cdot \gamma_A)$. To find them (see Figure 6), we pick a random value for this constant state X, apply all the 256 possible values to byte corresponding to the active byte in u_1. Then, we guess 32 bits of the subkeys u_0 (marked in gray) to partially decrypt the states. This results in 256 input states A_i, and each is partially encrypted to the same state value X by the first round. For each guess of 32 bits, the resulting sequence of 256 blocks A_i can be precomputed and stored along with the guessed value.

Similarly as the 4-round attack, we note that the last block A_{256} needs to be a partial block (e.g., it contains only 15 bytes) as we need the zero tweak value in the integral property. As before, this restricts the 255 other blocks on their last byte, but we emphasize that this can be handled easily as the last byte remains constant across all the blocks. The attack requires a total of queries of $2^{71+32+8} = 2^{111}$ blocks.

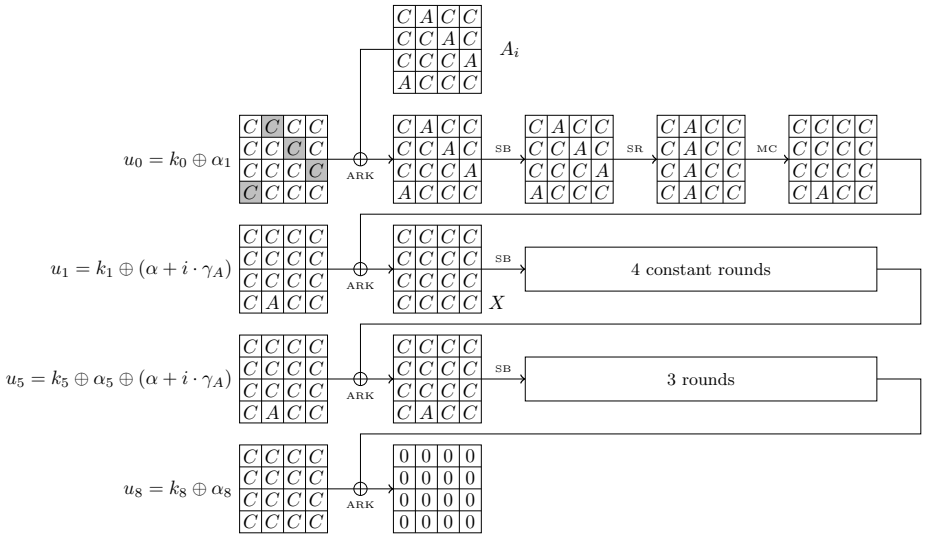

Figure 6: Integral property in the associated data of Silver processed by the 8-round block cipher E. The four bytes marked in gray correspond to guessed values.

4 Nonce-Repeating Analysis

In this section, we propose to analyze the security of Silver with respect to the nonce-reuse scenario. In the original document, the designer make

unclear statements about the expected security of Silver when the adversary reuses the nonce. Namely, they require the design to be used with non-repeating nonce, but still argue that the loss of security is not complete.

On the one hand, dealing with forgery, the designers claim that "*Silver has high forgery resistance even under nonce repetition.*" They conjecture that a forgery cannot be made with probability greater than 2^{-50} for 128-bit tags. However, we show in Section 4.1 that 50-bit security is no longer conservative and it can actually match the upperbound of the security by showing how an adversary can forge a message in $2^{49.46}$ blocks of nonce-repeating queries.

On the other hand, about privacy, they write that "*any attack against Silver should be readily converted into an attack on* `AES-ECB`*. Thus, although there is loss of indistinguishability, the loss of confidentiality is not catastrophic, and it simply reduces to the loss one would be prepared to accept when using* `ECB`". In the following Section 4.2, we show how a single nonce-repeating query can partially break the confidentiality of Silver by recovering part of the plaintext.

Finally, in Section 4.3, we show how to launch a key-recovery attack on 8-round Silver in 2^{111} blocks of queries, by extending the nonce-respecting forgery attack presented in the previous section.

4.1 Forgery Attack

Our forgery attack is mainly based on a *divide-and-conquer* approach to efficiently find two plaintext-ciphertexts (N, A, P, C, T) and (N, A, P', C', T') that have the same nonce and associated data, have the same plaintext length, and collide on the tag $T = T'$. Once such a pair is obtained, we choose a random one-block p, query $(N, A, P\|p)$ to receive $(C\|c, T'')$, and forge $(N, A, C'\|c, T'')$. From $T = T'$, it gives $\Sigma_A \oplus \Sigma_P \oplus \Sigma_C = \Sigma_A \oplus \Sigma'_P \oplus \Sigma'_C$, which is preserved after adding one more block p. Therefore the tag T'' is valid for $(N, A, C'\|c)$ and moreover the corresponding plaintext of forged decryption query is $P'||p$.

Simple Attack. We now focus on how to find such a pair of plaintext-ciphertexts of Silver. The attack targets the mode used in Silver, and can be applied to any underlying block cipher. Therefore, we describe it in a general form with n being the block size in bits (e.g., $n = 128$ in the case of Silver). To start with, we select a nonce N randomly, and use it as the nonce parameter for all the subsequent queries to Silver. Next, we

construct a set of plaintexts which all collide on $\lfloor \Sigma_P \oplus \Sigma_C \rfloor_{2n/3}$. In detail, we find a pair of i-th plaintext-ciphertext blocks (P_i^*, C_i^*) and $(P_i^\$, C_i^\$)$, for $1 \leq i \leq n/3$, such that $\lfloor P_i^* \rfloor_{2n/3} = \lfloor P_i^\$ \rfloor_{2n/3}$ and $\lfloor C_i^* \rfloor_{2n/3} = \lfloor C_i^\$ \rfloor_{2n/3}$. This can be done by selecting $2^{n/3}$ plaintexts, whose i-th blocks P_i^j with $1 \leq j \leq 2^{n/3}$ have the same $2n/3$ LSBs and differ in $n/3$ MSBs; namely, $\lfloor P_i^j \rfloor_{2n/3} = \lfloor P_i^{j'} \rfloor_{2n/3}$ and $\lceil P_i^j \rceil_{n/3} \neq \lceil P_i^{j'} \rceil_{n/3}$ for any distinct $1 \leq j, j' \leq 2^{n/3}$, querying them under the nonce N to receive ciphertexts, and finding a pair of i-th ciphertext blocks C_i^j and $C_i^{j'}$ colliding on the $2n/3$ LSBs, i.e., $\lfloor C_i^j \rfloor_{2n/3} = \lfloor C_i^{j'} \rfloor_{2n/3}$. Their corresponding i-th plaintext blocks are selected as $(P_i^*, P_i^\$)$. After that, a set of plaintexts is constructed:

$$\left\{ P_1^{j_1} \| P_2^{j_2} \| \cdots \| P_{n/3}^{j_{n/3}} \quad \middle| \quad j_1, j_2, \ldots j_{n/3} \in \{*, \$\} \right\}.$$

It is trivial to get that all plaintexts in this set have the same $2n/3$ LSBs for both Σ_P and Σ_C. Finally, we query those $n/3$-block plaintexts with N as nonce and $A = \epsilon$ as associated data to Silver, and find a pair of plaintext-ciphertext that collide on the received tag. Note that if a pair of plaintexts also have the same $\lceil \Sigma_P \oplus \Sigma_C \rceil_{n/3}$, then the tags collide. Since there are in total $2^{n/3}$ plaintexts in the set, we can expect to obtain a pair colliding on tag with a good probability.

Here, one may think that trying $2^{n/6}$ plaintexts instead of trying $2^{n/3}$ plaintexts is sufficient to find a colliding pair on the $n/3$ MSBs. However this does not work. The important thing is that the $2^{n/3}$ plaintexts are not chosen randomly. They are generated by linearly combining $n/3$ paired message blocks, which prevents us from using classical birthday arguments.

Overall, the forgery attack on Silver in the nonce-repeating model is completed. The complexity for finding a collision for the first $n/3$ blocks requires $2^{n/3+1}$ queries and each query has $n/3$ blocks. The same complexity is required for finding the tag collision. In the end, the complexity amounts to $n/3 \cdot 2^{n/3+2}$ queried blocks, which equals $2^{50.08}$, and this slightly exceeds the claimed 50-bit security. In addition, considering the success probability of finding a collision in each block, more queries are required. In the following explanation, we show how to improve the data complexity.

Advanced Attack. The overall idea consists in generating two colliding message pairs in each of the first $n/6$ blocks, while the simple attack generated one pair in each of the first $n/3$ blocks. In the tag collision phase, we first have 2 choices in each block regarding which message pairs we pick. Namely, we have $2^{n/6}$ choices of $n/6$-block message-pairs chain.

Then, for each of them, we can further consider $2^{n/6}$ combinations of messages. In total, $2^{n/6} \cdot 2^{n/6} = 2^{n/3}$ messages are examined for finding a tag collision, which is the same as the simple attack. With this effort, the complexity for generating collisions in each block is roughly $n/6 \cdot 2^{n/3+2}$, which stays unchanged from the simple attack, and the complexity for finding tag collision is reduced to $n/6 \cdot 2^{n/3+1}$, which is a half of the simple attack.

Hereafter, we evaluate the details with optimization particular for Silver, i.e. $n = 128$. First, to generate two colliding pairs in each block, we fix the 84 LSBs of the plaintext and try all the 2^{44} values in the 44 MSBs. Then, we examine the collision on the 84 LSBs of the ciphertext. According to the birthday paradox, we find a collision with probability 0.36 with 2^{42} messages. According to [16, Theorem 3.2], the probability of obtaining a collision for $(\log N)$-bit output function with trying $\theta \cdot N^{1/2}$ inputs is given by $1 - e^{-\frac{\theta^2}{2}}$. With 2^{44} messages, θ equals to 4, thus the success probability of finding a collision is $1 - e^{-8} \approx 0.999$. The probability of finding the second collision is almost the same, thus with probability $(1 - e^{-8})^2$, two colliding pairs can be obtained. We simultaneously apply this analysis for the first 22 blocks. Namely, each query consists of 22 blocks and the 84 LSBs are always fixed. This is iterated 2^{44} times by changing the value of 44 MSBs in all the blocks. In the end, the number of queries is $22 \cdot 2^{44} = 2^{48.46}$ message blocks and the probability of obtaining two colliding pairs in all the 22 blocks is $(1 - e^{-8})^{44} \approx 0.985$.

Generating a tag collision is rather simple, which is done as mentioned in the overview. We first choose which of colliding message pairs is used in each of the first 22 blocks, which yields 2^{22} choices of message-pair chains. For each of them, we further choose the message for each block, in which the 84 LSBs of Σ_P and Σ_C always take the same value. With 2^{22} messages, the probability for colliding 44 MSBs is 2^{-22}. Thus by examining all 2^{22} choices of 22-block message-pair chains, a tag collision can be generated. The number of queries for generating a tag collision is $22 \cdot 2^{44} \approx 2^{48.46}$ message blocks and the success probability is e^{-1}.

By combining the above two phases, the number of queries equals $2^{48.46} + 2^{48.46} = 2^{49.46}$ message blocks, and the success probability is $(1 - e^{-8})^{44} \cdot e^{-1} \approx 0.36$, which is almost the same as one for the ordinary birthday paradox. With this attack, we claim that 50-bit security against forgery in the nonce-repeating model is no longer conservative.

4.2 Breaking Confidentiality

First of all, we point out that Silver encrypts the last partial message block in the similar manner as the `CTR` mode, i.e. it generates the key stream and takes the XOR with the plaintext. Thus, the designers' claim only comparing the security of the `ECB` mode is strange. Comparison with the combination of the `ECB` and `CTR` modes seems more natural. Due to the property of `CTR`-mode like structure, recovering the plaintext of the last partial block with nonce-repeating queries is easy. Yet, we show another way of recovering plaintext which is particular to the computation structure in Silver.

Our observation to break confidentiality in Silver under nonce repetition essentially relies on a missing domain separation. In most ciphers, the cases of full block and partial block treatments are made distinct by using independent permutations by, for instance, injecting different tweak values in the corresponding block cipher calls. In Silver, this is not the case, and the adversary can use this at his advantage to learn internal values during an encryption.

Consider a short plaintext P of 15 bytes, a given nonce N and a secret key k. There is no associated data. To encrypt P, we observe that its length $b_P = 15$ does not fill one block, so we use the partial-block treatment (see Figure 2, right). Namely, we first generate the mask $\mu = E_{KS(s\cdot\gamma)}(b_p \,||\, b_p)$, where $s = 1$, and simply XOR its 15 most significant bytes to P to produce the corresponding ciphertext C. The subsequent operations related to the tag generation T are irrelevant here.

Once an adversary gets (N, C, T) with $|C| = 15$, he can recover the original plaintext P as long as he can recompute the internal mask value μ. We note that this is indeed possible using the nonce-repeating encryption query $(N, A = \epsilon, P = b_p \,||\, b_p)$ since the same subkeys $KS(s \cdot \gamma)$ yield the same permutation to be used since $s = 1$.

Therefore, with a single encryption query under the same nonce, the adversary can break the confidentiality of C. We note that the same observation for longer P with b_p not a multiple of 16 can be conducted, but would only recover the last $b_p \pmod{16}$ bytes of P. The second query would simply consist of $s - 1$ random blocks, $s = \left\lfloor \frac{b_p}{16} \right\rfloor$ and the same last block $b_p \,||\, b_p$ to use the same permutation to reveal μ.

4.3 Key-Recovery Attack on 8-Round Silver

The forgery attack on 8-round Silver in the nonce-respecting model discussed in Section 3.2, can be further extended into a key-recovery attack

working on the same number of rounds in the nonce-repeating model. Recall that this attack recovers four bytes (marked in gray in Figure 6) of $u_0 = k_0 \oplus \alpha_1$ for a nonce N. Then, if the value of α_1 for that nonce N can be recovered, which in turn determines four bytes of k_0, we can do a brute-force search to recover the other bytes of k_0, and therefore the key k. Therefore, we are left to find an algorithm to recover this α_1.

First, note that with respect to the target α_1, it is known that the 8 least significant bits of α and the 63 most significant bits of α_9^L are zeros. Hence, the number of candidate values for α_1 is essentially reduced to $2^{57=(128-8-63)}$. Below, we explain how to further reduce the number of its candidate values, and eventually to recover its value in the nonce-reuse model.

For a nonce N, let (N, A, P, C, T) and (N, A, P', C', T') be two plaintext-ciphertext pairs of Silver with N as the nonce and with the same associated data A. Moreover, both P and P' are one full-block long. We observe that if $T = T'$ holds, it implies $\Sigma_A \oplus \Sigma_P \oplus \Sigma_C = \Sigma_A \oplus \Sigma'_P \oplus \Sigma'_C$, which is equivalent to:

$$\Sigma_C \oplus \Sigma'_C = \Big(C + (\alpha + \gamma)\Big) \oplus \Big(C' + (\alpha + \gamma)\Big) = P \oplus P'.$$

From the known values of C, C' and $P \oplus P'$, we can recover partially the value of $\alpha + \gamma$. For the sake of simplicity, we denote $\alpha + \gamma$ as X and $P \oplus P'$ as Y, the i-th bit of C as $C[i]$, and the i-th carry bit for $C + X$ as $\mathtt{CR}[i]$. Similarly, we define $C'[i]$, $X[i]$, $Y[i]$ and $\mathtt{CR}'[i]$. At the bit level, the computation of $\Sigma_C \oplus \Sigma'_C$ gives:

$$\Big(C[i] \oplus X[i] \oplus \mathtt{CR}[i]\Big) \oplus \Big(C'[i] \oplus X[i] \oplus \mathtt{CR}'[i]\Big) = Y[i],$$

which is equivalent to:

$$\mathtt{CR}[i] \oplus \mathtt{CR}'[i] = C[i] \oplus C'[i] \oplus Y[i].$$

Hence, we can compute the difference for each carry bit between $C + X$ and $C' + X$, that is $\mathtt{CR}[i] \oplus \mathtt{CR}'[i]$ for each i. Moreover, at the bit level, the computations of $C + X$ and $C' + X$ give:

$$\mathtt{CR}[i+1] = \Big(C[i] + X[i] + \mathtt{CR}[i]\Big) \gg 1,$$
$$\mathtt{CR}'[i+1] = \Big(C'[i] + X[i] + \mathtt{CR}'[i]\Big) \gg 1,$$

where $\gg$ denotes the right shift by one bit.

Observation 1 *If* $C[i] = C'[i]$ *and* $P[i] \oplus P'[i] = 1$*, then* $X[i]$ *can be computed as:* $X[i] = C[i] \oplus \mathtt{CR}[i+1] \oplus \mathtt{CR}'[i+1]$.

Proof. From $Y[i] = P[i] \oplus P'[i] = 1$ and $C[i] = C'[i]$, we have $\mathtt{CR}[i] \oplus \mathtt{CR}'[i] = 1$. Without loss of generality, we assume $\mathtt{CR}[i] = 0$ and $\mathtt{CR}'[i] = 1$ and distinguish two cases with $C[i] = 0$ and $C[i] = 1$ separately.

- **Case** $C[i] = C'[i] = 0$**.** We have that:

$$\begin{aligned}\mathtt{CR}[i+1] \oplus \mathtt{CR}'[i+1] &= ((0 + X[i] + 0) \gg 1) \oplus ((0 + X[i] + 1) \gg 1),\\ &= (X[i] + 1) \gg 1.\end{aligned}$$

  It is trivial to get that $\mathtt{CR}[i+1] \oplus \mathtt{CR}'[i+1] = 0$ implies $X[i] = 0$, and $\mathtt{CR}[i+1] \oplus \mathtt{CR}'[i+1] = 1$ implies $X[i] = 1$. Hence $X[i] = C[i] \oplus \mathtt{CR}[i+1] \oplus \mathtt{CR}'[i+1]$ holds.

- **Case** $C[i] = C'[i] = 1$**.** We have that:

$$\begin{aligned}\mathtt{CR}[i+1] \oplus \mathtt{CR}'[i+1] &= ((1 + X[i] + 0) \gg 1) \oplus ((1 + X[i] + 1) \gg 1),\\ &= ((X[i] + 1) \oplus (X[i] + 2)) \gg 1.\end{aligned}$$

  It is trivial to get that $\mathtt{CR}[i+1] \oplus \mathtt{CR}'[i+1] = 0$ implies $X[i] = 1$, and $\mathtt{CR}[i+1] \oplus \mathtt{CR}'[i+1] = 1$ implies $X[i] = 0$. Hence $X[i] = C[i] \oplus \mathtt{CR}[i+1] \oplus \mathtt{CR}'[i+1]$ holds. □

Based on the above observation, for a nonce N, an algorithm of recovering partially its corresponding $\alpha + \gamma$ is as follows. The notations follow the above definitions.

1. Select two sets of 2^{64} distinct one full-block plaintext $\{P\}$ and $\{P'\}$ such that all plaintexts of $\{P\}$ (resp. $\{P'\}$) have the same value of P^R (resp. P'^R) and moreover $P^R \oplus P'^R = 1^{64}$ holds. Query all $(N, A = \epsilon, P)$s and $(N, A = \epsilon, P')$s to Silver and receive (C, T)s and (C', T')s, respectively. Find a pair (N, A, P, C, T) and (N, A, P', C', T') with $T = T'$.
2. Compute the difference of carry bits $\mathtt{CR} \oplus \mathtt{CR}'$ between $C + X$ and $C' + X$, where X refers to $\alpha + \gamma$.
3. Recover the bits $X[i]$s that satisfy $C[i] = C'[i]$ and $P[i] \oplus P'[i] = 1$.

We now evaluate the number of bits of X that can be recovered. Particularly, for each i with $1 \le i \le 64$, the condition $P[i] \oplus P'[i] = 1$ always holds, and $X[i]$ can be recovered as long as $C[i] = C'[i]$ holds.

Therefore on average 32 bits of the right half X^R of X can be recovered. Then, by doing a similar attack again which forces $P^L \oplus P'^L = 1^{64}$ instead, we can recover 32 bits of the left half X^L of X. Combining together, we get to know (at least) 64 bits of X with a complexity of 2^{66}.

Putting everything together, the key-recovery algorithm for 8-round Silver in the nonce-repeating model is detailed as follows.

1. Launch the forgery attack in Section 3. Let N be the nonce used in the forged decryption query.
2. For this nonce N, launch the above attack algorithm to recover (at least) 64 bits of its corresponding $\alpha + \gamma$.
3. Guess the unknown 65 of α_9 (corresponding to the nonce N) exhaustively, then compute key schedule function of AES to get α, α_1 and $\alpha + \gamma$, and examine if all the conditions on them are satisfied or not. This recovers the correct value of α_9 and in turn α_1.
4. Compute 32 bits of k_0 (marked in grey in Fig. 6). Finally recover the other bits of k_0 by a brute-force search, which in turn recovers the key k.

The overall complexity is obviously dominated by Step 1, which is 2^{111} blocks for all queries.

5 Conclusion

In this paper, we have presented the first third-party security analysis of the CAESAR candidate Silver. For nonce-respecting adversaries, we show an 8-round forgery attack with 2^{111} blocks of queries. The attack exploits the sparse injection of the block tweak, which allows to exploit the internal difference between blocks. For nonce-repeating adversaries, we show practical forgery and plaintext-recovery attacks against full Silver and a key-recovery attack against eight rounds of Silver. Our plaintext-recovery attack shows that the security of Silver in the nonce-repeating model is much less than that of `AES-ECB`, which breaks the security claim made by the designers. Our key-recovery attack shows that the adversaries can attack more rounds against Silver than `AES-128`.

We believe that achieving 128-bit security based on `AES-128` is very challenging but definitely worth trying. Proposing a new construction for tweaking `AES` seems a good approach. Our results show that careful security analysis, or possibly security proofs, is strongly required when a new tweaking method is proposed. More security analysis on the other tweaked `AES` based designs are open.

Acknowledgement. Jérémy Jean is supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06). Lei Wang is supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06), Major State Basic Research Development Program (973 Plan) (2013CB338004), National Natural Science Foundation of China (61472250) and Innovation Plan of science and technology of Shanghai (14511100300).

References

1. Bernstein, D.: CAESAR Competition. http://competitions.cr.yp.to/caesar.html (2013)
2. Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher Square. In Biham, E., ed.: FSE 1997. Volume 1267 of LNCS., Springer (1997) 149–165
3. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer (2002)
4. Derbez, P., Fouque, P., Jean, J.: Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In Johansson, T., Nguyen, P.Q., eds.: EUROCRYPT 2013. Volume 7881 of LNCS., Springer (2013) 371–387
5. Jean, J., Nikolić, I., Peyrin, T.: `Deoxys` v1.2 (2014) Submission to the CAESAR competition.
6. Jean, J., Nikolić, I., Peyrin, T.: `Joltik` v1.2 (2014) Submission to the CAESAR competition.
7. Jean, J., Nikolić, I., Peyrin, T.: `Kiasu` v1.2 (2014) Submission to the CAESAR competition.
8. Jean, J., Nikolic, I., Peyrin, T.: Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In Sarkar, P., Iwata, T., eds.: ASIACRYPT 2014, Part II. Volume 8874 of LNCS., Springer (2014) 274–288
9. Knudsen, L.R., Wagner, D.: Integral Cryptanalysis. In Daemen, J., Rijmen, V., eds.: FSE 2002. Volume 2365 of LNCS., Springer-Verlag (2002) 112–127
10. Krovetz, T., Rogaway, P.: The Software Performance of Authenticated-Encryption Modes. In Joux, A., ed.: FSE 2011. Volume 6733 of LNCS., Springer (2011) 306–327
11. Lipmaa, H., Moriai, S.: Efficient Algorithms for Computing Differential Properties of Addition. In Matsui, M., ed.: FSE 2001. Volume 2355 of LNCS., Springer (2002) 336–350
12. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable Block Ciphers. In Yung, M., ed.: CRYPTO 2002. Volume 2442 of LNCS., Springer (2002) 31–46
13. Penazzi, D., Montes, M.: Silver v1. Submitted to the CAESAR competition (March 2014)
14. Peyrin, T.: Improved Differential Attacks for ECHO and Grøstl. In Rabin, T., ed.: CRYPTO 2010. Volume 6223 of LNCS., Springer (2010) 370–392
15. Rogaway, P.: Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In Lee, P.J., ed.: ASIACRYPT 2004. Volume 3329 of LNCS., Springer (2004) 16–31
16. Vaudenay, S.: A Classical Introduction to Cryptography: Applications for Communications Security. Springer (2006)