# Silver v.1

Designers: Daniel Penazzi and Miguel Montes
Submitters: Daniel Penazzi and Miguel Montes
danielpenazzi@gmail.com

March 15, 2014

**Abstract**

We present here Silver , an authenticated encryption with associated data. Silver uses only AES and modular arithmetic operations (with $2^{64}$ as module) as primitives, is parallelizable, online, fast and has proofs of security for both privacy and authenticity. It requires the use of a nonce, but if a nonce is repeated and a forgery is created, this does not affect the ability to forge under other nonces. The nonce does not need to be random, just not repeating. Additional overhead over ECB is small and the length of the ciphertext (excluding the authentication tag) is the same as the length of the plaintext.

# Contents

# Chapter 1

# Specification

## 1.1 Parameters

Silver has three parameters: key length, nonce length, and tag length. Parameter space: Each parameter is an integer number of bytes. The key length and nonce length are each 16 bytes (128 bits). The tag length is 16 bytes, but shorter tag lengths can be used by truncating to the desired number of bytes. We discourage any tag length below 8 bytes, although we understand that for certain applications 4 bytes may be enough.

## 1.2 Recommended Parameters Sets

Primary recommended parameter sets: 16 byte (128 bits) key, 16 byte (128 bit) nonce, 16 byte (128 bit) tag.

## 1.3 Authenticated Encryption

The inputs to authenticated encryption are a plaintext $P$, associated data $A$, a public message number $N$, and a key $K$. The number of bytes in $P$ must be at most $2^{64} - 1$, and the number of bytes of $A$ must be at most $2^{64} - 1$ too but for security purposes we recommend a limit of $2^{50} - 1$ each.

As stated previously in 1.1, the number of bytes of the nonce and key is fixed to be 16. There is no secret message number. The output of authenticated encryption is a ciphertext $(C; T)$ obtained by concatenating an

unauthenticated ciphertext $C$ and a tag $T$ (of length at most 16 bytes, as specified above in 1.1). The length of $C$ is the same as the number of bytes of $P$.

As stated in the requirements to the CAESAR competition, it is assumed that the length of $P, A$ in bits are multiples of 8, i.e., $P$ and $A$ consist of a string of bytes.

We now provide the details.

## 1.4 Notation

$\oplus$ denotes the bitwise xor.

All data is assumed to be little endian.

$+$ will denote the sum of the group $(\mathbb{Z}/2^{64}\mathbb{Z}) \times (\mathbb{Z}/2^{64}\mathbb{Z})$.

$i * M$ denotes $M + M + \ldots + M$, where there are $i$ terms.

$trunc_b$ is truncate a 16 byte array to the leftmost $b$ bytes.

i.e. $trunc_b(B_1, B_2, ..., B_{16}) = (B_1, B_2, ..., B_b)$.

$||$ denotes concatenation.

## 1.5 Treatment of the Public Message Number

The public message number $N$ will be used together with the key $K$ to create a secondary key $\kappa = \kappa_{N,K}$, which is simply the encryption of the nonce $N$ under the key $K$ using AES (of 128 bits). $N$ is no longer used as such after the creation of $\kappa$. Note that $\kappa$ itself is a nonce, i.e., if $N$ does not repeat within the lifetime of a key, neither does $\kappa$. The only difference is that some adversary could have control over $N$, but not over $\kappa$.

## 1.6 MAES (modified AES)

A key part in Silver is a modification of AES which we call MAES. MAES is exactly the same as AES except for the key schedule. Given a key schedule $ks$ that on input a 128 bit key $K$ produces the 11 round keys needed for AES, we will denote by $\text{MAES}(B, ks(K))$ the encryption of a 128 bit block $B$ under the 11 round keys produced by $ks(K)$.

We will use the following specific family of key schedules:

### 1.6.1 Specification of pks

Assume that the usual key schedule of AES on input $K$ produces round keys $R_0(K), R_1(K), ..., R_{10}(K)$. Let $S$ be a 128 bit word. Then $\text{pks}_S^N(K)$ produces round keys $R_0^S(K,N)$, $R_1^S(K,N), ..., R_{10}^S(K,N)$ given by:

$$\begin{aligned} R_j^S(K,N) &:= R_j(K) \oplus R_j(\kappa) \quad (j \neq 0,1,5,9) \\ R_0^S(K,N) &:= R_0(K) \oplus R_1(\kappa) \\ R_j^S(K,N) &:= R_j(K) \oplus (\kappa + S) \quad (j = 1,9) \\ R_j^S(K,N) &:= R_j(K) \oplus R_j(\kappa) \oplus (\kappa + S) \quad (j = 5) \end{aligned}$$

where $\kappa$ is as in 1.5

## 1.7 Encryption of the plaintext

If the byte length $b_P$ of $P$ is a multiple of 16 $P$ is partitioned into blocks of 16 bytes: $P_1, P_2, ..., P_s$. If the byte length $b_P$ of $P$ is not a multiple of 16, we partition $P$ in the same way except that $P_s$ is made up of the final $b_P - \lfloor \frac{b_P}{16} \rfloor$ bytes of $P$. (thus, it is a partial block).

After obtaining $\kappa$ as in 1.5, we create $IC = (R_9(\kappa) \vee (1_{64}||1_{64}))$ where $1_{64}$ is the 64 bits that make the little endian representation of the number 1, $\vee$ is bitwise OR and $R_9(\kappa)$ is as in 1.6.1 (i.e. the one round key from the key round expansion we didn't use). Thus $IC$ is a pair of two odd numbers.

If $b_P$ is a multiple of 16, the ciphertexts blocks are simply:

$$C_i = \text{MAES}(P_i, \text{pks}_{i*IC}^N(K)) \qquad i = 1, ..., s$$

and decryption is simply

$$P_i = \text{MAES}^{-1}(C_i, \text{pks}_{i*IC}^N(K)) \qquad i = 1, ..., s$$

If $b_P$ is not a multiple of 16, then the above is true only for $i \leq s-1$, and for the final block we switch to counter mode:

Let $\ell$ be the length in bytes of $P_s$ (i.e., $\ell = b_P - \lfloor \frac{b_P}{16} \rfloor$). Then we do:

$$\begin{aligned} f &:= b_P || b_P \qquad (b_P \text{ represented as a } 64-\text{bit little endian}) \\ C_s &:= P_s \oplus trunc_\ell(\text{MAES}(f, \text{pks}^N_{s*IC}(\kappa))) \qquad (**) \end{aligned}$$

# 1.8 Authenticity

## 1.8.1 Treatment of the Plaintext/Ciphertext

We construct a checksum $XT$. If there is no plaintext, $XT = 0$.

In the case in which $b_P$ is a multiple of 16 $XT$ is the xor of all plaintexts blocks together with the xor of the ciphertext blocks (masked by multiples of IC) . That is:

$$XT = P_1 \oplus ... \oplus P_s \oplus (C_1 + \kappa + IC) \oplus (C_2 + \kappa + 2 * IC) ... \oplus (C_s + \kappa + s * IC)$$

where $IC$ is the one on 1.7.

When $b_P$ is not a multiple of 16, we construct $XT$ as above only with xors up to the $s-1$ blocks and then we xor to XT a special block $BE$, constructed as follows:

If $\ell$ is the length in bytes of $P_s$, then we create a block $B$ of 16 bytes that consists of the $\ell$ bytes of $P$ concatenated with the $16 - \ell$ bytes of $\text{MAES}(f, \text{pks}^N_{s*IC}(\kappa))$ that were not used in the computation of $C_s$ (see $(**)$ in 1.7), except that the rightmost byte is set to $\ell (= b_P - \lfloor \frac{b_P}{16} \rfloor)$. $B$ is encrypted using MAES with modifier $(s+1) * IC$ to obtain $BE$: $BE = \text{MAES}(B, \text{pks}^N_{(s+1)*IC}(\kappa))$

## 1.8.2 Treatment of the Associated Data

We compute a checksum $AT$ of the associated data in the following way: If there is no associated data, we define $AT = 0$. If the associated data $A$ is not empty, then $A$ is partitioned into blocks of 16 bytes: $A_1, A_2, ..., A_t$. If the byte length $b_A$ of $A$ is not a multiple of 16, $A_t$ is made up of the final $b_A - \lfloor \frac{b_A}{16} \rfloor$ bytes of $A$, padded with one byte equal to 1 and then bytes zero (to the right) until one gets 16 bytes. Construct $IC^{ad}$ as: $IC^{ad} = IC \wedge (\text{0xffffffff} || 0_{64}) =$

$(R_9(\kappa) \wedge (\text{0xffffffff}||0_{64})) \vee (1_{64}||0_{64}))$ i.e., it consists of the left part of IC with 0 in the right 64 bits.

$AT$ is computed thus:

$$\begin{array}{rll} & AT := & 0 \\ \text{For i} = 1, ..., \text{t} & AT := & AT \oplus \text{MAES}(A_i, \text{pks}^N_{i*IC^{ad}}(K)) \end{array}$$

except that if the last block was padded, then it is encrypted using $\text{MAES}(A_i, \text{pks}^N_0(K))$ instead of $\text{MAES}(A_i, \text{pks}^N_{s*IC^{ad}}(K))$

### 1.8.3 Computation of the tag

Let $P_{s+1} = AT \oplus XT$ and compute the tag as

$$T_{base} := \text{MAES}(P_{s+1}, t\text{pks}^N_g(K))$$

where $g = b_A||b_P$ ($b_A$ and $b_P$ in little endian) and $t\text{pks}^N_g$ is a special key expansion used only on the computation of the tag, and it is defined as: Compute the round keys given by $\text{pks}^N_g(K)$, say $r_0, ..., r_{10}$. Then the key schedule of $t\text{pks}^N_g(K)$ is $r_2, r_9, r_3, r_4, r_6, r_1, r_7, r_8, r_{10}, r_5, r_0$

If less than 128 bits is desired, $T_{base}$ can be truncated to the desired number of bytes and concatenated to the end of $C$. The only loss of security is the one suffered from going from a 128 bit tag to a $\tau$-bit tag.

# Chapter 2

# Security Goals

| goal | bits of security |
|---|---|
| confidentiality for the plaintext | 128 |
| integrity for the plaintext | 128 |
| integrity for the associated data | 128 |
| integrity for the public message number | 128 |

If the tag is truncated to $\tau$ bits, the numbers 128 above in the last three entries change to $\tau$.

That is, we expect that any attack on the confidentiality of the plaintext will need $2^{128}$ effort and if the length of the tag $T$ is $\tau$, a forgery of the plaintext, associated data or public message number cannot be made with probability greater than $2^{-\tau}$ (i.e., an expected $2^{\tau}$ attempts need to be made before a forgery is accepted as valid). However, in accordance to the CAESAR call, we do not distinguish between messages one of which is a truncation of the other by a number of bits less than 8.

This assumes that $P$ is at most $2^{50}$ bytes long and $A$ is at most $2^{50}$ bytes long and that the public message number is not repeated within the lifetime of a key, i.e., it must be a nonce.

There is no secret message number.

If the public message number is repeated there is loss of privacy up to indistinguishability from "RandomByBlocksCipher" i.e, Silver under nonce repetition is indistinguishable from a cipher that on input a plaintext $P$ outputs ciphertext $C$ (or vice versa) randomly except for the condition that if two different plaintexts $P^1, P^2$ have two equal blocks in the same position (ie. $P^1_i = P^2_i$) then the corresponding ciphertext blocks must be the same

(and vice versa under ciphertext query).

We do not promise $2^{-\tau}$-integrity if the nonce is repeated, but see 3.4.1 for a discussion of the level of integrity loss under nonce repetition.

However, even if the public message number is repeated and a forgery can be made with that public message number, this has no effect on the ability to make forgeries (or in the privacy loss) with other public message numbers that do not repeat.

Although the number of bytes per message encrypted under a nonce is bounded, it can encrypt a number of messages greater than the birthday bound $2^{64}$, but we recommend not to approach $2^{128}$ messages encrypted under a single key.

# Chapter 3

# Security Analysis

## 3.1 General Considerations

MAES is basically a tweaked version of AES, the tweak being applied to the round keys.

The security of Silver is based on an assumed property of AES. It is widely assumed that AES outputs for distinct inputs are indistinguishable from random (if the number of inputs does not approach $2^{64}$). We are assuming a little more: that the outputs of AES with the round keys changes as specified above are indistinguishable from random if the change is not repeated.

Specifically, we assume the following:

**Property 3.1.1** *Given 128 bit block $B$, an integer $i$ between 0 and $2^{50}-1$, a bit $b$ and a 128 bit block $N$ define $\mu AES(N,B,i,b)$ as $\mu AES(N,B,i,b) = \mathrm{MAES}(B, \mathrm{pks}^N_{i*IC^b}(K))$ where $IC^1 = IC$ and $IC^0 = IC^{ad}$ where these are obtained from $N$ and the secret key $K$ as explained previously. Let $\{(N_j, i_j, b_j)\}_{j\in J}$ be a set of triplets that are different from each other. Then, for any sequence $(B_j)_{j\in J}$ of 128 bits blocks (some of which can be repeated) the sequence $\mu AES(N_j, B_j, i_j, b_j)$ cannot be distinguished from a random sequence.*

If AES does not have this property it would be very surprising, given the 16 years of analysis it has sustained. In fact, if AES cannot be distinguished from random, then $\mu AES(N_j, B_j, i_j, b_j)$ will be a random sequence *if all the $N_j$'s are distinct*, since a change in $N_j$ involves a radical change of the rounds keys, so this would be equivalently to say that a sequence of values encrypted under AES with different keys is a random sequence. (this also holds true

for differentiating $\text{pks}_g^N$ from $t\text{pks}_g^N$, which we didn't write in 3.1.1 so as to not clutter the writing).

So the only possibility that property 3.1.1 is not true involves distinguishing from random a subsequence of the form $\mu AES(N, B_j, i_j, b_j)$ (i.e., with $N$ fixed).

In this case the changes in the round keys involve only three round keys.

True, there has been same attacks against AES that uses precisely related keys (eg. [1]) , so not all key schedule changes to AES would be secure.

However, these attacks use differentials in round keys that are "close", while the designers of AES proved that 4 rounds of AES provide full protection against differential or linear cryptanalysis, so the fact that are 4 rounds between the rounds 1 and 5, and between the rounds 5 and 9 (which are the rounds where the differences in the keys are) should provide excellent protection. The change is made on rounds 1 and 9 instead of 0 and 10 to complicate things a little bit more for an attacker who tries to inject differences in the plaintexts/ciphertext that would cancel the differences in the round keys.

Moreover, the change is a random change unknown to any adversary (since it involves $\kappa$ and $IC$). Actually, since a change with index $i$ involves xoring to the round keys 1,5,9 the block $\kappa + i * IC$ the adversary cannot know what the change is, but the adversary could try to estimate the difference between the changes using index $i$ and index $j$. We limit $i$ to be less than $2^{50}$ so the adversary will have less information over these differences.

Thus, $\mu AES$ can be considered as a tweaked AES, and Silver as a tweaked authenticated encryption scheme, and in that sense, the security proof of [4] provides a proof of the security.

However, since there are some differences with our design, for completeness, we write the adapted proof for our case. Also, we want to discuss what happens if instead of assuming 3.1.1 we just assume that AES is indistinguishable from random, which is a weaker hypothesis but one with which (almost) everyone agrees.

## 3.2 Indistinguishability from random

**Theorem 3.2.1** *If Silver can be distinguished from random with probability better than* $1/2 + \epsilon$ *then* $\mu AES$ *can be distinguished from a random block oracle (see definition below) with probability better than* $1/2 + \epsilon$

*Proof:*

Let $Adv$ be a nonce-respecting adversary that wants to distinguish between the case Silver and a "randomcipher". That is $Adv$ will be given an oracle cipher $CphO$ that takes as input a nonce $N$ and either a plaintext $P$ and outputs a ciphertext $C$, or a ciphertext $C$ and outputs a plaintext $P$. The oracle can be one of two: it will either be Silver or a "randomcipher" (ie. and oracle that on each query $(N, P)$ or $(N, C)$ outputs a uniformly random value of the appropriate length).

$Adv$ is allowed to do $q$ (nonce respecting) queries, after which $Adv$ has to guess whether the provided oracle is Silver or randomcipher.

Assume that $Adv$ can guess correctly with probability $1/2 + \epsilon$.

Let $Adv^*$ be an adversary against $\mu AES$ who wants to distinguish it from random.

That is, $Adv^*$ will be given a family of oracles $\mathbb{O}$ which will contain a family of pairs of oracles $O(N, B, i)$ and $O^{-1}(N, B, i)$ which will, each one, given a 128 bit block $N$, an integer $i$ and a 128 bit block $B$, output another 128-bit block, with the properties:

$O^{-1}(N, O(N, B, i), i) = B$ and $O(N, O^{-1}(N, B, i), i) = B$

In addition $\mathbb{O}$ contains two other families of oracles, but in this case only the encryption ones, $tO$ and $adO$. $adO$ also receives inputs like $O$, and $tO$ receives as input three 128 bit blocks $N, B, S$.

$Adv^*$ has to decide whether these oracles correspond to $\mu AES$ under a fixed secret key $K$ or to random. (which we will call "randomblock" to distinguish it from "randomcipher").

We model "randomblock" in the following way:

For each $(N, i)$ start with $\text{Domain}_{(N,i)}$, $\text{Image}_{(N,i)}$ and $\text{Pairs}_{(N,i)}$ empty. On input $(N, B, i)$ for the "direct" oracle, the oracle checks to see whether $B$ is in $\text{Domain}_{(N,i)}$. If not, it outputs a (uniformly) random 128-bit value $C$ adds $B$ to $\text{Domain}_{(N,i)}$, $C$ to $\text{Image}_{(N,i)}$ and $(B, C)$ to $\text{Pairs}_{(N,i)}$. In the other hand if $B \in \text{Domain}_{(N,i)}$ then it searches for $(B, C)$ in $\text{Pairs}_{(N,i)}$ and outputs $C$.

In the case the call is to the inverse the procedure is similar, starting with checking whether $B$ is in $\text{Image}_{(N,i)}$.

In the case of $adO$ and $tO$ the procedure is similar, but no calls to the inverse are accepted.

Let $Adv^*$ do the following: given $\mathbb{O}$, $Adv^*$ creates a cipher "cipher($\mathbb{O}$)" which will follow the procedure given in 1.7 except that instead of doing $C_i = \text{MAES}(P_i, \text{pks}^N_{i*IC}(K))$ it does $C_i = O(N, P_i, i)$. (and similarly, with the inverse, for the decryption), and instead of using $t\text{pks}^N_S(K)$ for the com-

putation of the tag, uses $tO$ and for the computations related to the tag, $adO$.

Now, $Adv^*$ takes the queries that $Adv$ would do to distinguish Silver from "randomcipher", runs cipher($\mathbb{O}$) with them and returns them to $A$.

If $\mathbb{O} = \mu AES$, then since cipher($\mu AES$)=Silver , $Adv^*$ would be providing $Adv$ with an oracle that is Silver .

On the other hand, let's see what happens when $\mathbb{O}$ =randomblock. Because $Adv$ is nonce respecting, all $N$'s from all queries are distinct. Since for each fixed $N$ the $i$ 's and $b$'s involved are all distinct, then when running randomblock applied to the queries of $Adv$, then actually all $\text{Domain}_{(N,i)}$, $\text{Image}_{(N,i)}$ and $\text{Pairs}_{(N,i)}$ are going to be empty or of cardinality one, and in this last case $(N, i)$ will not be requested again. Hence we can change randomblock to an oracle $O^*$ that on any call it simply returns a (uniformly) random 128 bit string.

That is, as far as $Adv$ is concerned, cipher(randomblock)=cipher($O^*$). But cipher($O^*$) will simply be a string of independent and random values, that is, "randomcipher".

Thus $Adv^*$ would be providing $Adv$ with an oracle cipher which is one of two possibilities:Silver or randomcipher. When $Adv$ guesses which oracle was provided, $Adv^*$ guesses that $O = \mu AES$ if $Adv$ guesses Silver , and $Adv^*$ guesses $O$ =randomblock if $Adv$ guesses randomcipher.

Since $Adv$ guesses correctly with with probability $1/2+\epsilon$, $Adv^*$ also guesses correctly with the same probability.

==========QED.

Of course, the above proof assumes 3.1.1. What if we just use the weaker hypothesis that AES in indistinguishable from random?

Since a change of nonce implies a radical change of round keys, then the AES encryptions or decryptions will be totally uncorrelated between different nonces. Thus, suppose an adversary makes $q$ queries $Q_1, ..., Q_q$ to the oracle and has a test $Test(Q_1, .., Q_q)$ which outputs 1 or 0, (1 meaning the oracle is Silver , 0 that it is random), which has a probability $p^*$ of being right. Since AES is indistinguishable from random, the answer to each of the queries $Q_j$ are uncorrelated to each other, so in fact $Test(Q_1, .., Q_q)$ will have to be of the form DoSomething($test(Q_1), test(Q_2), ..., test(Q_q)$) where DoSomething:$(\mathbb{Z}/2\mathbb{Z})^q \mapsto \mathbb{Z}/2\mathbb{Z}$ is some function devised by the adversary. A minimal condition for DoSomething to be coherent would have to be that DoSomething$(1, 1, ..., 1) = 1$ and DoSomething$(0, 0, ..., 0) = 0$. In fact, it

seems that the only reasonable choice for DoSomething will have to be of the form DoSomething$(b_1, ..., b_q) = 1$ if there are at least $t$ $i$'s with $b_i = 1$ and zero otherwise, where $t$ is some threshold.

Therefore if we denote by $p$ the probability that *test* is right, then if $p - \frac{1}{2}$ is small, $p^* - \frac{1}{2}$ will be small too. Thus we can assume that *Adv* will make just one query.

*Adv* will then have to find some correlation between the encryptions (under one nonce) of different blocks. The encryptions are done using different but related keys, but the analysis done on the part of resistance against forgery (see below) shows that the probability of finding any such correlation is negligible.

## 3.3 Unforgeability

Again, the proof of the following theorem is very similar to the one on [4], but we try to do it by reducing unforgeability as much as possible to AES being indistinguishable from random as opposed to $\mu AES$ being indistinguishable from random. Besides, the way we construct the tag is different than the one in [4].

**Theorem 3.3.1** *The probability of a successful forgery of a tag of $\tau$ bits of Silver by a nonce respecting adversary is at most $2^{-\tau}$.*

*Proof:* Let *Adv* be a nonce respecting adversary that can request encryption/authentications queries and one decryption/verification query.

*Adv* wins when a quad $(N, A, C, T)$ is produced that was never an answer to an encryption query and is accepted as valid.

The queries that *Adv* does will be of the form $(N^j, A^j, P^j)$ with answers $(N^j, A^j, C^j, T^j)$ and all $N^j$ are different between them.

We have several cases:

1. $N \neq N_j$ for all $j$.

   The tag is obtained by truncating $T_{base}$, where $T_{base}$ is constructed as in 1.8.3. This computation involves the calculation of MAES$(P_{s+1}, t\text{pks}_g^N(K))$

   Since $N \neq N_j$ for all $j$, then the key schedule $t\text{pks}_g^N$ is wildly different from the key schedule $t\text{pks}_g^{N^j}$ hence the AES encryption under these

different keys would collide only with probability $2^{-128}$ and the probability that the tag that $Adv$ creates is the correct one is $2^{-\tau}$ in this case.

2. $N = N_j$ for some $j$.

   Moreover, since we saw in the last item that encryption under different nonces are totally uncorrelated, those $r$ for which $N \neq N^r$ are irrelevant to the possibility of constructing a forgery, hence for practical purposes we can assume that the attacker made just ONE query. (i.e., $j = 1$).

   (a) The length of $C$ is different from the length of corresponding ciphertext $C^1$.

   In this case there are several independent reasons why the probability of forgery will be bound by $2^{-\tau}$. We will start with the weakest one.

   Since $g = b_A||b_P$. codifies the length of $P$(=length of $C$), then the $g$ that corresponds to the case $(N, A, C, T)$ will be different from the $g^1$ that correspond to $(N^1, A^1, C^1, T^1)$. Thus $t\text{pks}_g^N$ would not have been used anywhere else. Under the hypothesis 3.1.1 the output will be random. But let us analyze a little bit more closely the situation to see how exactly 3.1.1 will have to fail in order to produce a forgery. In this part of the analysis we do not have a hard reduction to the indistinguishability from random of AES, since we have two key schedules that are equal on all keys except the keys 1,5,9 on which they differ by some quantity. For $i = 1, 5, 9$, one of the keys is of the form $r_i = f_i \oplus (\kappa + g)$, the other of the form $r_i^j = f_i \oplus (\kappa + g^1)$. The difference $g - g^1$ is known, and in fact can be manipulated, since the attacker can choose both of them. However, the difference $r_i \oplus r_i^1 = (\kappa + g) \oplus (\kappa + g^1)$ cannot be so easily manipulated.The probability that that difference involves a particular difference $\Delta$ will depend on the Hamming weight of $\kappa$ ([3]), which is random and not under the control of the adversary. In addition to this, the input $(N, A, C, T)$ will produce a checksum $AT \oplus XT$ and $(N^1, A^1, C^1, T^1)$ will produce a checksum $AT^1 \oplus XT^1$. If $AT \oplus XT = AT^1 \oplus XT^1$ then there will be a non-zero difference between the encryption of $AT \oplus XT$ under $t\text{pks}_g^N$ and the encryption under $t\text{pks}_{g^1}^N$. By the time we reach round 5,

the probability of any particular differential trail would be $2^{-150}$ and the probability that the final encryption of $AT \oplus XT$ exactly matches $T_{base}$ would be $2^{-128}$ (or, the probability that a $\tau$-bit truncation of the encryption of $AT \oplus XT$ exactly matches $T$ is $2^{-\tau}$).

So it must be $AT \oplus XT \neq AT^1 \oplus XT^1$ and the difference must be such that the difference between the encryption of $AT \oplus XT$ under $t\text{pks}_g^N$ after the xor with the round 1 key and the encryption of $AT^j \oplus XT^j$ under $t\text{pks}_{g^j}^N$ after the xor with the round 1 key is zero. We saw above a bound on calculating a concrete difference at the level of the round key, below we will discuss what is the probability that a useful difference on $AT \oplus XT$ found. But at any rate, then, after the round 5, a new difference will appear, and the probability of any particular differential trail up to the 9th round would be $2^{-150}$, and the probability that a $\tau$-bit truncation of the encryption of $AT \oplus XT$ exactly matches $T$ is $2^{-\tau}$.

All of that would be even if the attacker could somehow construct with high probability a "good" $AT \oplus XT$. However, the probability of getting the "right" $XT$ is also $2^{-128}$ because of the following analysis:

If the number of blocks of $C$ and $C^1$ are different, then in one of the cases there is at least one more AES encryption or decryption (depending whether the extra block was in the encrypt query or on the decrypt query) that is completely missing in the other computation, so the difference between $XT$ and $XT^1$ will be random since AES behaves like a random cipher and $XT$ is affected by both the plaintext block and the ciphertext block.

If the number of blocks is the same and the final block of both is a partial block, then they differ in the number of bytes. But since $EB$ contributes to $XT$ and $EB$ is the encryption of something that codifies the number of bytes in the last partial blocks, (and the adversary never sees nor has control over this particular encryption) then again by appealing to AES being indistinguishable from random, $XT$ would be random.

Note that these cases reduces to AES being indistinguishable from random.

The case that remains is that they have the same number of blocks,

say $s$, but one of them has a length that is not a multiple of 16 bytes and the other has a length that is a multiple of 16 bytes. Then the one with a length multiple of 16 will contribute to $XT$ with an encryption of a block using modifier $s*IC$ while the other will encrypt some block with the modifier $(s+1)*IC$ (because if the last block is a partial block, the encryption with $s*IC$ is used to form the ciphertext, but the encryption with $(s+1)*IC$ is the one that contributes to $XT$), and further more, this encryption is never revealed.

Thus as before, the probability of obtaining the correct tag would be $2^{-\tau}$.

(b) The lengths of $C$ and $C^1$ are equal, but $C \neq C^1$.

Let $i$ be the first index such that $C_i \neq C_i^1$. First assume that either $i$ is not the last index ($s$) or that the last block of plaintext is of 16 bytes. When verifying the tag, the verifier must decrypt $C$ in order to obtain the plaintext blocks. $C_i$ will be decrypted using key schedule modifier given by $(N, i)$, which is used only with the pair plaintext-ciphertext given by $(P_i^1, C_i^1)$. In particular $C_i$ will be decrypted using the same key schedule than $C_i^1$ and since they are different the decryption $P_i$ will be randomly different from $P_i^1$, under the assumption that AES is indistinguishable from random. The only way that $P_i$ can be distinguished from random is if there is a correlation to some other(s) $P_k$ or $P_k^1$. Similarly to the sub analysis we did when we were comparing encryption under $g$ and $g^1$, we will have here that the different plaintexts will be uncorrelated. There is a couple of differences in the analysis however: In this case the differences in the round keys are of the form $(\kappa + i * IC) \oplus (\kappa + j * IC)$ and $IC$ and $\kappa$ are random and unknown. (they are related somehow by the Rijndael key schedule, $\kappa$ being the seed and $IC$ being the 9th round key masked. Since they are separated by nine iterations of the expansion, we expect no useful relations can be found).

In favor of the attacker however, there are many blocks to work with now, instead of difference with exactly one as before. However if a specific difference in the round keys is needed, then this limits the choice to one block to work with. (the one that produces that difference). Moreover, since the attacker is limited to

this one query, any attack must be pre-planned so to speak. That is, since there is only a single query, there cannot be any adaptive attack.

Therefore, $P_i$ will be (almost) indistinguishable from random (the only restriction is the condition $P_i \neq P_i^1$). Moreover, the contribution of this block to $XT$ is $P_i \oplus (C_i + \kappa + i * IC)$, and since $C_i \neq C_i^1$, that xor difference can be anything.

Thus $AT \oplus XT$ will be equal to $AT^1 \oplus XT^1$ with probability $2^{-128}$, independently of what happens with the other blocks.

There remains to analyze the case in which $i = s$ and the last block is incomplete. Then any change in $C_s$ will produce the same changes in $P_s$. However, in this case the contribution to $XT$ comes not from $P_s, C_s$ but from $BE$, which is the encryption of $B$, where $B =$the concatenation of $P_s$ with some other bytes that come from the encryption of the length. The modifiers used in this are never used elsewhere. Thus $C_s \neq C_s^j \Rightarrow P_s \neq P_s^j \Rightarrow B \neq B^j$ which implies that the output of the two encryptions (the first one and the one done during the forgery) will be different but random apart from that.

Thus $XT \oplus XT^1$ will be random.

If $AT \oplus XT = AT^1 \oplus XT^1$ and $Adv$ provides $T = T^1$, then the forgery will succeed. If not, the input to AES under the modifier used to calculate the tag is different from the only one used with that same modifier, so, again, the output $T_{base}$ will be uniformly random (except that it MUST be different from $T_{base}^1$), and the truncation will equal the tag $T$ with probability almost $2^{-\tau}$.

Thus if trunc denotes the truncation to $\tau$ bits, and we denote $T_{base}$ as $T_b$, the probability $p$ of successful forgery in this case is:

$$\begin{aligned} p = \quad & P(T_b = T_b^j).P\left(trunc(T_b) = trunc(T_b^j) | T_b = T_b^j\right) + \\ & + P(T_b \neq T_b^j).P\left(trunc(T_b) = trunc(T_b^j) | T_b \neq T_b^j\right) \\ = \quad & 2^{-128}.1 + (1 - 2^{-128}).\frac{2^{128-\tau} - 1}{2^{128} - 1} \\ = \quad & 2^{-128} + 2^{-128}.(2^{128-\tau} - 1) \\ = \quad & 2^{-\tau} \end{aligned}$$

(c) $C = C^1$. We must have then $A \neq A^1$. (because the case $(N, A, C) = (N^1, A^1, C^1)$ with $T \neq T^1$ is impossible due to the deterministic nature of the computation of the tag). We have two sub cases:

i. $length(A) \neq length(A^j)$ Since $g.$ codifies the length of $A$, then the considerations in the first part of the analysis of the case of different ciphertexts length apply.
In addition, the second part of that analysis also apply here: if the number of blocks are different, then one block is missing in one of the computations. If one is a partial block and the other is not, there will be a computation that involves modifier 0 in one of them and not in the other, so again $AT \oplus AT^1$ will be random. Thus they have the same number of blocks and the last block is partial. That block is padded with a pad that codifies the number of bytes of the block, so there will be an AES encryption with the same key of two different blocks, hence the difference on the outputs is random (except that it must be non zero).

ii. The lengths are equal but there is an $i$ with $A_i \neq A_i^1$. The analysis is the same as the case in which the ciphertexts differ in at least one block, except that here the attacker does not even see the encryptions. The other distinct thing in the analysis is that if there is only one block of difference then $AT = AT^1$ with probability 0. If there are two or more blocks of difference then the probability of forgery will be $2^{-\tau}$.
If $i$ is the only block in which there is a difference, then $AT \neq AT^j$, hence $T_{base} \neq T_{base}^j$ and the probability of forgery is 0 if $\tau = 128$.
If $\tau < 128$, then the probability of forgery is $P(trunc_\tau(T_{base} \oplus T_{base}^j) = 0) = \frac{|\{w \neq 0 : trunc_\tau(w) = 0\}|}{|\{w \neq 0\}|} = \frac{2^{128-\tau}-1}{2^{128}-1} < 2^{-\tau}$.

==========QED.

## 3.4 Repetition of nonce

If the nonce is repeated, the previous proofs are invalid, since we used the fact that we do not repeat the nonce in several places.

For example, suppose that $XT$ had been specified as only the xor of the plaintexts. Then an appropriate proof of theorem 3.3.1 would still be valid, but that scheme would totally fail authenticity if the nonce is repeated three times: simply fix the nonce $N$ and the associated data $A$, take three non null blocks $P_1$, $P_2$, $P_3$ and request encryption of $P^1 = P_1||P_2$, $P^2 = (P_1 \oplus P_3)||P_2$ and $P^3 = P_1||(P_2 \oplus P_3)$. If the ciphertexts are $C_1^j||C_2^j$ and the tags are $T^j$, then $(N, A, C, T^1)$ is a forgery, where $C = C_1^2||C_2^3$.

However, this attack does not work against Silver due to the more involved creation of the tag.

In fact, there is no obvious loss of authenticity, due to several countermeasures taken.

We will show that for an adversary that can repeat nonces, obtaining a forgery with probability better than $2^{-\tau}$ is impossible in many cases, and in the case left, the probability of forgery is small, though we cannot provide a specific bound, thus this section is a discussion rather than a theorem.

### 3.4.1 Resistance of authenticity against nonce repetition

Suppose an adversary as before, but now not nonce-respecting.

Then as in the the proof of Theorem 3.3.1, if the final nonce $N$ was never used before, the probability of a successful forgery is $2^{-\tau}$, so suppose that $N$ equals some of the $N_j$'s. Since the nonces that are different than $N$ are irrelevant, we may suppose that all nonces are equal.

Those $j$'s for which $(b_A, b_P) \neq (b_{A^j}, b_{P^j})$ will have the modifier $g^j$ different from the modifier $g$ and as in the proof of Theorem 3.3.1 there will be some extra computations in the calculation of $AT, XT$ or both, so they will irrelevant in bounding the probability of the output of the encryption of $T_{base}$. Thus, we can discard them from further analysis, i.e., we keep $j$'s for which $(b_A, b_P) = (b_{A^j}, b_{P^j})$.

Now assume that the $A^j$'s are not all equal.

If there is an index $i$ such that $A_i$ is different from all other $A_i^j$, then the analysis of Theorem 3.3.1 applies. So, we can assume that for every $i$ there exists at least one $j_i$ with $A_i = A_i^{j_i}$ (though, of course, the $j_i$'s may be different between them).

Now, the attacker knows all the associated data but doesn't know the encryptions of them, nor the tags $AT$.

If we form a matrix such that in each row there are the encryptions and the $AT$'s corresponding to each query, the attacker is reduced to the following problem: given a matrix of random strings of 128 bits, which the attacker cannot see, but such that the attacker can know which entries are equal between them, choose an element from each column in such a way that the row formed in that way is not one of the rows of the matrix and the xor of the elements of this row equals zero. Clearly, this can be done only with probability $2^{-128}$.

So we may assume that all $A_j$'s are equal and $A$ is equal to them.

Hence we must have all ciphertexts different.

If $T \neq T^j$ for all $j$, then either $XT = XT^j$ for some $j$ (and then the verification will fail) or $XT \neq XT^j$ for all $j$, and then the tag will be (almost) random ($T_{base}$ will have to be different from all others $T^j_{base}$). If $\tau = 128$ the verification will fail, and if $\tau < 128$ the probability that the tag computed is $T$ has probability near $2^{-\tau}$.

So the attacker will have to pick a $k$ and choose $T = T^k$.

Again, if there is an index $i$ such that $C_i$ is different from all $C^j_i$, then $XT$ will be random, so we can assume that for every $i$ there exists at least one $j_i$ with $C_i = C^{j_i}_i$.

The difference between the case of the associated data and this, is that here the attacker knows all $C^j_i$ and $P^j_i$ (and hence, by the previous hypothesis, $C_i$ and $P_i$). However, $XT$ is influenced by $P_i \oplus (C_i + \kappa + i * IC)$.

Disregard the sum with $\kappa + i * IC$ for the moment.

In that case the attacker knows all differences in the $q$ matrices that have $q-1$ rows and $b = b_P$ columns and such that the $i$th matrix has entries $(j,k)$ equal to $P^i_k \oplus C^i_k \oplus P^j_k \oplus C^j_k$.

The attacker needs to find a nonempty subset of columns and for each column in the set, an element of the matrix in that column, (one element per column) such that the xor of all these elements is zero. If such set exists, then the blocks of $C^k$ that corresponds to those columns can be replaced with the corresponding blocks of the $C^j$'s that were chosen.

There are $\sum_{m=1}^{b} \binom{b}{m}(q-1)^m = q^b - 1$ such sums, and $q$ matrices, so there are a total of $q(q^b-1)$ sums, one of which must be zero. The probability of this event is $1 - (1-2^{-128})^{q(q^b-1)} \simeq 1 - e^{-q(q^b-1)2^{-128}}$.

For example, if $q = 2$ (nonce repeated once) and $b = 127$, then the probability will be approximately $1 - e^{-1}$ which is quite high. However finding such sum by brute force requires computing $q(q^b - 1) \simeq 2^{128}$ xor sums.

In fact, the problem is a special case of the subset problem (special, because of the requirement that there must be at most one element in each column), and that problem is NP complete, so there is no known algorithm that could solve it in general in a reasonable time. Of course, as with all NP problems, this is not much help in cryptography, since we require a problem that is hard to solve in all cases, not just the most difficult ones. Still, in our case all entries are random, so one should expect a difficult to solve problem.

Moreover, things are more complicated in the real Silver , because of the addition of $\kappa + i * IC$

The attacker will not know the actual differences that affect the tag, thus the attacker will have to do some differential analysis on the sum with an unknown mask. The attacker knows the plains and the ciphertext, hence the differences between them and the corresponding ones with query $j$, but in each query the ciphertext has been randomly produced, so the attacker cannot *control* the differences in the ciphertext, though the ones in the plaintexts are controllable (but *before* getting the ciphertexts).

These differential probabilities will depend on the hamming weight of $\kappa + i * IC$, which since it is random, we should expect to be quite high.

However, this is not a formal proof in the theoretical sense, thus is why we have set it as a discussion and not a theorem, but it illustrates that Silver has high forgery resistance even under nonce repetition. It is safe to assume that a forgery cannot be made with probability greater than, say, $2^{-50}$ for tags of length 128, which contrasts highly with the case of GCM in which repetition of a nonce allows trivial forgeries.

Also, the above discussion and the proof of theorem 3.3.1 show that even if repetition of a nonce allows a forgery for that nonce, other nonces past or present are unaffected (this is not true, for example, with AES-GCM).

### 3.4.2 Resistance against privacy under nonce repetition

Suppose the adversary repeats the nonce. We can assume that a single nonce is used several times.

Still, the different block positions will receive different modifiers. Hence randomblock will behave by outputting random blocks in different positions, and simply checking whether, when calls are made on the same position, the call was made previously or not.

Thus we can model this as a series of different block ciphers, one for each position. Within each position, this would be equivalent to ECB mode, so clearly repetition of a nonce involves a loss of indistinguishability.

But since the positions are independent, 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. (actually, something like blockECB). Of course, ECB is not a safe mode, so one may not want to accept such loss.

The proof of Theorem 3.2.1, suitable changed to contemplate this case, proves that if Silver with nonce repetition and "RandomByBlocksCipher" can be distinguished with probability $1/2+\epsilon$, then $\mu AES$ and randomoracle can be distinguished with the same probability. Here "RandomByBlocksCipher" is a cipher that on input $P$ outputs $C$ (or vice versa) randomly except for the condition that if two different plaintexts $P^1, P^2$ have two equal blocks in the same position (ie. $P_i^1 = P_i^2$) then the corresponding ciphertext blocks must be the same.

This contrasts with the total loss of confidentiality that happens when repeating the nonce in counter mode.

Of course, the above is under the hypothesis that $\mu AES$ is indistinguishable from random. Under the weaker (but more accepted) hypothesis that AES is indistinguishable from random, then the proof does not apply. An attacker could try to mount some sort of adaptive attack looking at the answers to the queries in order to search for differences that could be used to mount, say, a boomerang attack. However, the structure of the changes of the round keys and the Rijmen-Daemen theorem conspire against that.

## 3.5 Resistance against key collision attack

Since the key used from message to message changes, there could be a key collision attack. For example, suppose that we had used in the designs just $R_j(\kappa)$ as round keys, and the changes in rounds 1,5,9 had been simply xor with $i$. Then a possible attack would involve capturing $2^{64}$ different messages (hence, with different nonces) such that the first block of all these messages is fixed, say $B$, and the attacker knows $B$ (but not the other blocks). Let $C_1^j$ be the first blocks of these ciphertexts.

Then the attacker chooses $2^{64}$ different keys $k$, and computes $b_k = \tilde{E}_k(B)$

where $\tilde{E}$ is AES, but with round keys 1,5,9 xored with 1. Then the attacker searches for a collision between some $b_k$ and some $C_1^j$.

If there is one, then with high probability $k = \kappa^j$, and the attacker could decrypt the rest of the corresponding ciphertext.

However, this attack does not work against Silver for several reasons, including the fact that $\kappa$ is created from $K$ with a high nonlinearity (the AES encryption) and then both sets of round keys are used (i.e., the round keys of $K$ are not discarded) and that rounds 1,5,9 are modified using a secret counter.

Of course the attacker may try to create different $b_k$'s, say using as index a full set of round keys, but that would involve guessing a "key" of 1408 bits, so this approach would not work.

## 3.6 Resistance against related keys

Recent attacks like the one in [1] use keys that are related to improve the differential probabilities of an attack, taking advantage of some weakness in the key expansion of AES. (but for the 192 and 256 versions, which we do not use). However, the key expansion of Silver does not have those weakness, since $AES_K(N)$, which is highly nonlinear, is expanded and some of the round keys xored to the original round keys. the round keys, so related keys $K$ and $K'$ will produce highly different round keys for each nonce.

## 3.7 Resistance against forgery under loss of privacy

Let $Adv$ be an adversary that has access to a triplet $(N, A, P)$, but not to the tag produced. For example, a system administrator encrypts some $(N, A, P)$'s to send them somewhere, but then keeps the tags in a secure location thinking, mistakenly, that the tags will provide authenticity of $(N, A, P)$. (the correct method is keep $(N, A, C)$ and erase $(N, A, P)$ if there is possibility of an intrusion). The intruder may gain access to the stored texts, and wishes to alter them, but without access to the tags. In some AEADs (for example, OCB) it is easy to change the plaintext in this scenario without changing the tag. Not so in Silver , since the tag depends on both the ciphertext and the plaintext. Concretely:

**Theorem 3.7.1** *Let Adv be an adversary who has write access to a message* $(N, A, P)$ *and read-access to its ciphertext/tag pair* $(C, T)$ *but does not have write access to* $T$*, and does not have access to the AEAD system, but wishes to produce a forgery* $(N^*, A^*, P^*)$ *that when is feed into the AEAD, produces the same tag* $T$*.*

*If AEAD is instantiated with Silver , the probability of forgery is* $2^{-\tau}$*.*

*Proof:* As in the proof of 3.3.1 if the nonce, associated data or length of $P$ are changed, the probability of forgery is $2^{-\tau}$.

Thus, *Adv* has to change some block(s) of $P$. When fed into Silver this will produce ciphertext(s) block(s) that will be randomly generated (except for having to be distinct from the previous ones). Thus, as in the proof of 3.3.1, but with the roles of $P$ and $C$ reversed, the probability of forgery is $2^{-\tau}$.

==========QED.

# Chapter 4

# Features

The cipher has many advantages. It is basically AES in ECB mode, with a tweak that covers the deficiencies of ECB, so it is conceptually very simple. It builds on the work of [4] but has a novel feature in that the key is changed from session to session and the tweak is applied to the round keys.

The extra cost against, say ECB, is one extra AES encryption plus AES key expansion per message, plus 3 xors, and an update of a counter per block encrypted. In addition, for authenticity 2 64-bit sums and 2 xors are needed per block to update the checksum, plus a final AES encryption, and an extra AES encryption in the case of an incomplete block. So it is very fast on software and should also be fast on hardware (though we have not implemented it on hardware).

Since the encryption itself is AES with other round keys, it benefits from all the known speed ups of AES, including the Intel instructions and the hardware implementations.

It is NOT a mode of operation of AES, so it cannot use a black box implementation of AES. However, it can take advantage of any implementation of AES that is divided into two black boxes: one that computes the round keys, and the other that uses the round keys to implement a black box implementation of AES.

Since it is basically AES-ECB, it is highly parallelizable and it is also online for both encryption and decryption, meaning that it can produce ciphertext blocks before subsequent plaintexts blocks are known (except for the last block, which if incomplete need the length of the message), and a similar statement for decrypting. (however, it is to be noted than since this is an AEAD, in the case of decrypting the plaintext blocks should not be

released until the authenticator has been verified).

Moreover, not only it is online, but the order in which the plaintext can be processed is completely arbitrary (as long as the position of the block in the message is known), i.e., the cipher has a random access property. (although in order to use this property modular multiplication is needed).

The cipher is also incremental: if one block of plaintext or associated data is modified, then only two AES extra calls plus a few xors and arithmetic operations are needed to update the ciphertext and tag. (incrementally does not extend to changing the nonce: in that case all the computations must be redone).

The cipher can process $P$ or parts of $P$ without seeing $A$, $A$ or parts of $A$ without seeing $P$, or any combination.

This cipher is faster than AES-GCM. In Haswell it can encrypt at about 0,73 cycles per byte for long messages ($\simeq$ 200KB) and at about 1 cpb for shorter messages (1500 bytes). Decryption is a little bit slower, because of the need to multiply the keys by the inverse Rijndael matrix, but still it runs at about 0,89 cpb for long messages.

One big advantage over AES-GCM is that only AES and arithmetic operations are used: no Galois field operations are needed, taking advantage of the adaptability of AES to different environments (from latest generation chips to embedded 32 bit systems and byte oriented environments).

Another advantage over AES-GCM is that it can process longer messages.

Another advantage over AES-GCM is that a tag of length $\tau$ provides $\tau$ bits of security, unlike AES-GCM. (for example a 32-bit tag of AES-GCM only provides 16 bits of security).

Another advantage over AES-GCM is the following: in the latest Intel chips, there is a special instruction that speeds up the Galois field multiplication, but on environments that lack this instruction, AES-GCM needs to be implemented in general with the use of some large tables to be able to encrypt efficiently. Whenever a key is changed, these tables must be recomputed. On the other hand, re-keying in Silver is simply another AES key expansion.

Another advantage over AES-GCM is the resistance against public message number reuse. In AES-GCM repetition of the nonce is lethal for integrity and confidentiality since the encryption part of AES-GCM is counter mode and as for authenticity, if the nonce is repeated, the secret key $H$ used in the GHASH function can be obtained, allowing subsequent forgeries.

We have seen in 3.4 that forgery even under nonce repetition appears to

be intractable, (though we cannot give a hard proof) and even if a forgery can somehow be made, still repetition of the nonce does not reveal any information that would allow subsequent forgeries for other nonces, unlike the case in AES-GCM.

We have also seen that under nonce repetition there is indistinguishability with "RandomByBlocksCipher", which may not be a good enough thing, but it is better than repetition of nonce in counter mode, which produces total loss of privacy.

# Chapter 5

# Design Rationale

The designers have not hidden any weakness in this cipher.

There were several goals in the design of Silver . One was to have a high throughput cipher, faster than AES-GCM, and capable of competing even with OCB ([5]).

The other was to use as much components as possible of AES, since it is probably the most thoroughly analyzed cipher in existence, and no great weakness have been found on it.

For this, we could try to build a block-cipher based construction, or a stream-cipher based construction. However, we felt that the stream-cipher based construction, although potentially faster, would leave more room for attacks than a block-cipher based one, so we decided to go with this construction. Besides, our tests on some of the versions we created were not fast enough.

In order to allow high throughput, the cipher would need to allow for parallelization. This leads to only two classical modes: counter mode or ECB.

However counter mode is the one AES-GCM uses, hence in that case the only thing to do would be to find a better authentication. Also, ECB has the advantage that a change on one bit in the plaintext will change an average of 64 bits on the ciphertext, instead of the one bit change of counter mode, thus if we include in the computation of the tag the xor of the plaintext and ciphertext, one or the other will be uncontrollable by the attacker.

But ECB has many weakness, including distinguishability from random. Hence it would have to be a variation of ECB. OCB([5] and IAPM ([2]) do a variation of it, by masking the input and output of the black box AES

encryption of one block. However, it requires a careful analysis of exactly what changes to make. (for example using simply the block number as mask is not good). Moreover, since they are patented, using a similar approach with different masks may bring in legal troubles. (though OCB has a very nice patent permission scheme).

We decided to use the nonce to obtain a radical change of keys from one message to another, allowing better protection. The problem remained what changes to make from block to block in a single encryption. For reasons of efficiency it had to be something simpler than the radical change made from message to message.

Since AES is a very strong cipher, we had the idea to change the round keys by xoring a secret to some or all of them. The diffusion properties of AES ensure that even small changes will propagate to all bits of the block after 2 to 4 rounds.

One possibility was to change all round keys. But this, in addition to a loss of efficiency, may allow an adversary to try to nullify a change on one round with a change on the next round. By separating the changes by 4 full AES rounds, we allow the Rijmen-Daemen theorem to hold, obtaining a bound of $2^{-150}$ on the probability on any differential trail. Thus, we chose the round keys of rounds 1,5,9, given the 4 round property of AES and that we didn't want to let the adversary to be able to easily eliminate one of the changes by changing the plaintexts (if we chose the initial whitening key) or the ciphertext (if we had chosen the last round key).

We toyed with the idea of doing different changes on different round keys, but we then opted for simplicity. One possibility was to change some keys and also the order of the internal keys. Another was to for example, generate further round keys, at least after some encryptions, or apply some AES round to one or more round keys. But these approach would make it hard to be able to process a particular block without doing the changes to the keys that precede that block, so a change that depended on simply counting the block number was considered better.

The reason the nonce affects the round keys as defined (by xoring the round keys of the expansion of $K$ and $\kappa$) instead of using directly $\kappa$ as a session key has to do with two things: one is the resistance against a possible key collision attack, as explained above, and also because of the possibility that someone may choose to generate the nonces by computing the next nonce as $E_K(N)$. Although this is terrible practice (the secret key should not be used to generate nonces) if someone did it it would be catastrophic

under the other scheme, since the nonce of the next session would reveal the encryption key of the previous one. Also the way the round keys are defined make it impossible to find pairs $(K, N)$ and $(K^*, N^*)$ that would generate the same round keys.

We could have chosen a different order to mix the round keys of $K$ and $\kappa$, but we chose that one to allow on-the -fly implementation.

We decided to use both $P_i$ and $C_i$ to generate the tag as a protection against a trivial attack under nonce repetition and protection against an intruder attack (see 3.7) and we decided to mask $C_i$ as a protection against more elaborate forms of attack with nonce reuse. Both these measures imply a loss of some speed, about 0,1 cpb taken together, but we thought that this measures make Silver more robust.

We limit the message length to $2^{50}$ bytes=$2^{46}$ blocks so that even in the extreme case there would still be 17+17=34 bits of $i * IC$ unknown. (17 because the low bit in each half is known to be 1). Although even one bit of change should provide protection, this offers some extra protection at no real cost for the moment, since a thousand terabytes is more than enough for all practical purposes.

(for the associated data there would only be 17 bits unknown, but since the outputs of the encryptions are never revealed, we consider it safe).

In short, the things that distinguish Silver from other similar designs are: radical change of key from message to message, gentler change from block to block that can be made on any block independently of the others, use of both $P_i$ and $C_i$ (masked) in the production of the tag, and dependence only on AES for security. (well, plus some extra protection due to the additions).

# Chapter 6

# Intellectual Property

There are no known patents, patent applications, planned patent applications, or other intellectual-property constraints relevant to the use of the cipher.

If any of this information changes, the submitter will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

# Chapter 7

# Consent

The submitters hereby consents to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitters understands that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitters understands that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitters acknowledges that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitters understands that if they disagrees with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitters understands that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

# Bibliography


[1] Alex Biryukov and Dmitry Khovratovich, *"Related-key Cryptanalysis of the Full AES-192 and AES-256"*, Advances in Cryptology-ASIACRYPT 2009 Lecture Notes in Computer Science Volume 5912, 2009, pp 1-18.

[2] C. Jutla, "Encryption modes with almost free message integrity, Advanced in Cryptology, EUROCRYPT 01, Springer-Verlag, 2001

[3] Lipmaa, H. and Moriai, S. "Efficient algorithms for computing differential properties of addition". In Fast Software Encryption 2001, number 2355 in Lecture Notes in Computer Science, pages 336-350, Berlin, 2002.

[4] Moses Liskov and Ronald L. Rivest and David Wagner, *"Tweakable Block Ciphers"*, Advance in Cryptology, CRYPTO'02, Lecture Notes in Computer Science Volume vol 2442, 2002, pp 31-46

[5] P. Rogaway, M. Bellare, J. Black, and T. Krovitz, "OCB: a block-cipher mode of operation for efficient authenticated encryption", ACM CCS, 2001