Silver and AESCPFB

Miguel Montes[1]   Daniel Penazzi[2]

[1] Instituto Universitario Aeronáutico, Córdoba, Argentina

[2] Universidad Nacional de Córdoba, Facultad de Matemática, Astronomía y Física, Córdoba, Argentina

23,24-8-14

Table of Contents

Table of Contents

- CPFB is a mode of operation, uses AES as a black box, including the key expansion.

- CPFB is a mode of operation, uses AES as a black box, including the key expansion.
- Silver is a tweak of AES. The tweak can be thought to be wholly contained within the key expansion, thus only the encryption/decryption component of AES can be used as a black box.

- CPFB is a mode of operation, uses AES as a black box, including the key expansion.
- Silver is a tweak of AES. The tweak can be thought to be wholly contained within the key expansion, thus only the encryption/decryption component of AES can be used as a black box.
- Silver is basically ECB with a change in the key expansion on each block, CPFB is a mix of counter mode with Plaintext Feedback mode.

- CPFB is a mode of operation, uses AES as a black box, including the key expansion.
- Silver is a tweak of AES. The tweak can be thought to be wholly contained within the key expansion, thus only the encryption/decryption component of AES can be used as a black box.
- Silver is basically ECB with a change in the key expansion on each block, CPFB is a mix of counter mode with Plaintext Feedback mode.
- Silver can be paralellized on both encryption and decryption, CPFB only on encryption.

- CPFB is a mode of operation, uses AES as a black box, including the key expansion.
- Silver is a tweak of AES. The tweak can be thought to be wholly contained within the key expansion, thus only the encryption/decryption component of AES can be used as a black box.
- Silver is basically ECB with a change in the key expansion on each block, CPFB is a mix of counter mode with Plaintext Feedback mode.
- Silver can be paralellized on both encryption and decryption, CPFB only on encryption.
- CPFB only requires the encryption module of AES, Silver requires both the encryption and decryption modules.

- CPFB is a mode of operation, uses AES as a black box, including the key expansion.
- Silver is a tweak of AES. The tweak can be thought to be wholly contained within the key expansion, thus only the encryption/decryption component of AES can be used as a black box.
- Silver is basically ECB with a change in the key expansion on each block, CPFB is a mix of counter mode with Plaintext Feedback mode.
- Silver can be paralellized on both encryption and decryption, CPFB only on encryption.
- CPFB only requires the encryption module of AES, Silver requires both the encryption and decryption modules.
- They both are based wholly on AES. (no Galois Field operations or calls to other hashes or MACs).

- CPFB is a mode of operation, uses AES as a black box, including the key expansion.
- Silver is a tweak of AES. The tweak can be thought to be wholly contained within the key expansion, thus only the encryption/decryption component of AES can be used as a black box.
- Silver is basically ECB with a change in the key expansion on each block, CPFB is a mix of counter mode with Plaintext Feedback mode.
- Silver can be paralellized on both encryption and decryption, CPFB only on encryption.
- CPFB only requires the encryption module of AES, Silver requires both the encryption and decryption modules.
- They both are based wholly on AES. (no Galois Field operations or calls to other hashes or MACs).
- They both use the nonce and master key to derive session keys.

Table of Contents

- We wanted Silver to be AES based parallelizable in both encryption and decryption.

- We wanted Silver to be AES based parallelizable in both encryption and decryption.
- So we chose a tweaked ECB mode.

- We wanted Silver to be AES based parallelizable in both encryption and decryption.
- So we chose a tweaked ECB mode.
- The tweak consist in changing some round keys.

- We wanted Silver to be AES based parallelizable in both encryption and decryption.
- So we chose a tweaked ECB mode.
- The tweak consist in changing some round keys.
- We chose the 1st,5th and 9th round keys to take advantage of the AES 4 round property.

- We wanted Silver to be AES based parallelizable in both encryption and decryption.
- So we chose a tweaked ECB mode.
- The tweak consist in changing some round keys.
- We chose the 1st,5th and 9th round keys to take advantage of the AES 4 round property.
- The change to the rounds is a simple xor with a counter, but the counter is key and nonce dependent.

- We wanted Silver to be AES based parallelizable in both encryption and decryption.
- So we chose a tweaked ECB mode.
- The tweak consist in changing some round keys.
- We chose the 1st,5th and 9th round keys to take advantage of the AES 4 round property.
- The change to the rounds is a simple xor with a counter, but the counter is key and nonce dependent.
- key and nonce of 128 bits each.

Encrypt(P, $roundkeys$, κ, IC)

- Split P into 128 bit blocks, last block partial if necesary (no pad).

$$\text{Encrypt}(P, roundkeys, \kappa, IC)$$

- Split *P* into 128 bit blocks, last block partial if necesary (no pad).
- For $i \leftarrow$ 1...last complete block

$$\text{Encrypt}(P, \mathit{roundkeys}, \kappa, \mathit{IC})$$

- Split P into 128 bit blocks, last block partial if necesary (no pad).

- For $i \leftarrow$ 1...last complete block
  - $\mathit{temprkeys}_i = \mathit{roundkeys}_i$, $(i \neq 1, 5, 9)$
  - $\mathit{temprkeys}_i = \mathit{roundkeys}_i \oplus (\kappa + \mathit{counter})$, $(i = 1, 5, 9)$

$$\text{Encrypt}(P, \textit{roundkeys}, \kappa, \textit{IC})$$

- Split P into 128 bit blocks, last block partial if necesary (no pad).
- $\kappa = \mathrm{AES}_{key}(npub)$,

- For $i \leftarrow 1$...last complete block
    - $temprkeys_i = roundkeys_i$, $(i \neq 1, 5, 9)$
    - $temprkeys_i = roundkeys_i \oplus (\kappa + counter)$, $(i = 1, 5, 9)$

$$\text{Encrypt}(P, \mathit{roundkeys}, \kappa, \mathit{IC})$$

- + is the sum of $(\mathbb{Z}/2^{64}\mathbb{Z}) \times (\mathbb{Z}/2^{64}\mathbb{Z})$
- Split *P* into 128 bit blocks, last block partial if necesary (no pad).
- $\kappa = \mathrm{AES}_{key}(npub)$,

- For $i \leftarrow$ 1...last complete block
  - $\mathit{temprkeys}_i = \mathit{roundkeys}_i, \ (i \neq 1, 5, 9)$
  - $\mathit{temprkeys}_i = \mathit{roundkeys}_i \oplus (\kappa + \mathit{counter}), \ (i = 1, 5, 9)$

Encrypt(P, $roundkeys$, κ, IC)

- + is the sum of $(\mathbb{Z}/2^{64}\mathbb{Z}) \times (\mathbb{Z}/2^{64}\mathbb{Z})$
- Split P into 128 bit blocks, last block partial if necesary (no pad).
- $\kappa = \mathrm{AES}_{key}(npub)$, $counter \leftarrow \{0\}^{128}$
- For $i \leftarrow 1$...last complete block
    - $counter \leftarrow counter + 1$
    - $temprkeys_i = roundkeys_i$, $(i \neq 1, 5, 9)$
    - $temprkeys_i = roundkeys_i \oplus (\kappa + counter)$, $(i = 1, 5, 9)$

$$\text{Encrypt}(P, \mathit{roundkeys}, \kappa, \mathit{IC})$$

- + is the sum of $(\mathbb{Z}/2^{64}\mathbb{Z}) \times (\mathbb{Z}/2^{64}\mathbb{Z})$
- Split P into 128 bit blocks, last block partial if necesary (no pad).
- $\kappa = \mathrm{AES}_{key}(npub)$, $counter \leftarrow \{0\}^{128}$

- For $i \leftarrow 1$...last complete block
    - $counter \leftarrow counter + IC$
    - $temprkeys_i = roundkeys_i$, $(i \neq 1, 5, 9)$
    - $temprkeys_i = roundkeys_i \oplus (\kappa + counter)$, $(i = 1, 5, 9)$

$$\text{Encrypt}(P, \mathit{roundkeys}, \kappa, \mathit{IC})$$

- + is the sum of $(\mathbb{Z}/2^{64}\mathbb{Z}) \times (\mathbb{Z}/2^{64}\mathbb{Z})$
- Split P into 128 bit blocks, last block partial if necesary (no pad).
- $\kappa = \mathrm{AES}_{key}(npub)$, $counter \leftarrow \{0\}^{128}$
- $IC \leftarrow AESroundkey_9(\kappa)\mathrm{OR}([1]_{64} \,||\, [1]_{64})$
- For $i \leftarrow 1$...last complete block
  - $counter \leftarrow counter + IC$
  - $temprkeys_i = roundkeys_i$, $(i \neq 1, 5, 9)$
  - $temprkeys_i = roundkeys_i \oplus (\kappa + counter)$, $(i = 1, 5, 9)$

$$\text{Encrypt}(P, \textit{roundkeys}, \kappa, \textit{IC})$$

- + is the sum of $(\mathbb{Z}/2^{64}\mathbb{Z}) \times (\mathbb{Z}/2^{64}\mathbb{Z})$
- Split P into 128 bit blocks, last block partial if necesary (no pad).
- $\kappa = \mathrm{AES}_{key}(npub)$, $counter \leftarrow \{0\}^{128}$
- $IC \leftarrow AESroundkey_9(\kappa)\mathrm{OR}([1]_{64} \,||\, [1]_{64})$
- For $i \leftarrow 1$...last complete block
  - $counter \leftarrow counter + IC$
  - $temprkeys_i = roundkeys_i$, $(i \neq 1, 5, 9)$
  - $temprkeys_i = roundkeys_i \oplus (\kappa + counter)$, $(i = 1, 5, 9)$
  - encrypt P_i using AES with *temprkeys* to obtain C_i

$$\text{Encrypt}(P, roundkeys, \kappa, IC)$$

- + is the sum of $(\mathbb{Z}/2^{64}\mathbb{Z}) \times (\mathbb{Z}/2^{64}\mathbb{Z})$
- Split P into 128 bit blocks, last block partial if necesary (no pad).
- $\kappa = \mathrm{AES}_{key}(npub)$, $counter \leftarrow \{0\}^{128}$, $XT \leftarrow \{0\}^{128}$
- $IC \leftarrow AESroundkey_9(\kappa)\mathrm{OR}([1]_{64} \,||\, [1]_{64})$
- For $i \leftarrow 1$...last complete block
  - $counter \leftarrow counter + IC$
  - $temprkeys_i = roundkeys_i$, $(i \neq 1, 5, 9)$
  - $temprkeys_i = roundkeys_i \oplus (\kappa + counter)$, $(i = 1, 5, 9)$
  - encrypt P_i using AES with $temprkeys$ to obtain C_i
  - $XT \leftarrow XT \oplus P_i$

$$\text{Encrypt}(P, \mathit{roundkeys}, \kappa, \mathit{IC})$$

- + is the sum of $(\mathbb{Z}/2^{64}\mathbb{Z}) \times (\mathbb{Z}/2^{64}\mathbb{Z})$
- Split P into 128 bit blocks, last block partial if necesary (no pad).
- $\kappa = \mathrm{AES}_{key}(npub)$, $counter \leftarrow \{0\}^{128}$, $XT \leftarrow \{0\}^{128}$
- $IC \leftarrow AESroundkey_9(\kappa)\mathrm{OR}([1]_{64} \,||\, [1]_{64})$
- For $i \leftarrow 1$...last complete block
  - $counter \leftarrow counter + IC$
  - $temprkeys_i = roundkeys_i$, $(i \neq 1, 5, 9)$
  - $temprkeys_i = roundkeys_i \oplus (\kappa + counter)$, $(i = 1, 5, 9)$
  - encrypt P_i using AES with *temprkeys* to obtain C_i
  - $XT \leftarrow XT \oplus P_i \oplus C_i$

$$\text{Encrypt}(P, roundkeys, \kappa, IC)$$

- + is the sum of $(\mathbb{Z}/2^{64}\mathbb{Z}) \times (\mathbb{Z}/2^{64}\mathbb{Z})$
- Split P into 128 bit blocks, last block partial if necesary (no pad).
- $\kappa = \mathrm{AES}_{key}(npub)$, $counter \leftarrow \{0\}^{128}$, $XT \leftarrow \{0\}^{128}$
- $IC \leftarrow AESroundkey_9(\kappa)\mathrm{OR}([1]_{64} \,||\, [1]_{64})$
- For $i \leftarrow 1$...last complete block
  - $counter \leftarrow counter + IC$
  - $temprkeys_i = roundkeys_i$, $(i \neq 1, 5, 9)$
  - $temprkeys_i = roundkeys_i \oplus (\kappa + counter)$, $(i = 1, 5, 9)$
  - encrypt P_i using AES with *temprkeys* to obtain C_i
  - $XT \leftarrow XT \oplus P_i \oplus (C_i + \kappa + counter)$

- Return (C, XT)

If there is a last incomplete block of ℓ bytes:

Encrypt with, basically, counter mode:

- Return (C, XT)

If there is a last incomplete block of ℓ bytes:

Encrypt with, basically, counter mode:

- $bP = \left[\frac{|P|}{8}\right]_{64}$
- $counter \leftarrow counter + IC$
- $tmp =$ encrypt $(bP||bP)$ with roundkeys associated to the counter.
- Split tmp in bytes $tmp_1||tmp_2||...||tmp_{16}$
- $C_s = P_s \oplus (tmp_1||...||tmp_\ell)$

- Return (C, XT)

If there is a last incomplete block of ℓ bytes:

Encrypt with, basically, counter mode:

- $bP = \left[\frac{|P|}{8}\right]_{64}$
- $counter \leftarrow counter + IC$
- $tmp =$ encrypt $(bP||bP)$ with roundkeys associated to the counter.
- Split tmp in bytes $tmp_1||tmp_2||...||tmp_{16}$
- $C_s = P_s \oplus (tmp_1||...||tmp_\ell)$
- to authenticate:
- Return (C, XT)

If there is a last incomplete block of ℓ bytes:

Encrypt with, basically, counter mode:

- $bP = \left[\frac{|P|}{8}\right]_{64}$
- $counter \leftarrow counter + IC$
- $tmp =$ encrypt $(bP||bP)$ with roundkeys associated to the counter.
- Split tmp in bytes $tmp_1||tmp_2||...||tmp_{16}$
- $C_s = P_s \oplus (tmp_1||...||tmp_\ell)$
- to authenticate:
- $B = P_s||tmp_{\ell+1}||...||tmp_{15}||\,[\ell]_8$
- $counter \leftarrow counter + IC$
- $XT \leftarrow XT \oplus$ (encryption of B with AES using roundkeys associated to the new counter)
- Return (C, XT)

ProcessAD(A, $roundkeys$, κ, IC)

- Split A in 128 bits blocks, padding with bytes 1,0,...,0 if necessary (but only if necesary).

ProcessAD(A, $roundkeys$, κ, IC)

- Split A in 128 bits blocks, padding with bytes 1,0,...,0 if necessary (but only if necesary).
- Encrypt the blocks with roundkeys associated to counters, but this time the counter increases by $AIC = IC\&(\{1\}^{64}||\{0\}^{64})$.

ProcessAD(A, $roundkeys$, κ, IC)

- Split A in 128 bits blocks, padding with bytes 1,0,...,0 if necessary (but only if necesary).
- Encrypt the blocks with roundkeys associated to counters, but this time the counter increases by $AIC = IC\&(\{1\}^{64}||\{0\}^{64})$.
- If the last block is complete, use the counter that would go there, else, use counter 0.

ProcessAD($A, roundkeys, \kappa, IC$)

- Split A in 128 bits blocks, padding with bytes 1,0,...,0 if necessary (but only if necesary).
- Encrypt the blocks with roundkeys associated to counters, but this time the counter increases by $AIC = IC\&(\{1\}^{64}||\{0\}^{64})$.
- If the last block is complete, use the counter that would go there, else, use counter 0.
- Xor all the ciphertexts to form an AD tag AT.

Tag

- Obtain AT, XT as above.

Tag

- Obtain AT, XT as above.
- Final tag T is the encryption of $AT \oplus XT$ with AES and roundkeys given by:

Tag

- Obtain AT, XT as above.
- Final tag T is the encryption of $AT \oplus XT$ with AES and roundkeys given by:
  - roundkeys changed by using counter $g \leftarrow \left(\left[\frac{|A|}{8} \right]_{64} || \left[\frac{|P|}{8} \right]_{64} \right)$

Tag

- Obtain AT, XT as above.
- Final tag T is the encryption of $AT \oplus XT$ with AES and roundkeys given by:
  - roundkeys changed by using counter $g \leftarrow \left(\left[\frac{|A|}{8} \right]_{64} || \left[\frac{|P|}{8} \right]_{64} \right)$
  - and changing the order of the roundkeys using the permutation $(2, 3, 4, 6, 7, 8, 10, 0)(9, 1, 5)$

Tag

- Obtain AT, XT as above.
- Final tag T is the encryption of $AT \oplus XT$ with AES and roundkeys given by:
  - roundkeys changed by using counter $g \leftarrow \left(\left[\frac{|A|}{8} \right]_{64} || \left[\frac{|P|}{8} \right]_{64} \right)$
  - and changing the order of the roundkeys using the permutation $(2, 3, 4, 6, 7, 8, 10, 0)(9, 1, 5)$

Decryption and Verification are the obvious ones.

- In addition to the tweak on each block, Silver changes the key expansion of AES so that the nonce also influences the round keys:

- In addition to the tweak on each block, Silver changes the key expansion of AES so that the nonce also influences the round keys:
- $\kappa = \mathrm{AES}_{\mathit{key}}(\mathit{npub})$

- In addition to the tweak on each block, Silver changes the key expansion of AES so that the nonce also influences the round keys:
- $\kappa = \mathrm{AES}_{key}(npub)$
- $roundkey_i = AESroundkey_i(key) \oplus AESroundkey_i(\kappa), \quad i \neq 0, 1, 9$

- In addition to the tweak on each block, Silver changes the key expansion of AES so that the nonce also influences the round keys:
- $\kappa = \mathrm{AES}_{key}(npub)$
- $roundkey_i = AESroundkey_i(key) \oplus AESroundkey_i(\kappa), \quad i \neq 0, 1, 9$
- $roundkey_i = AESroundkey_i(key)$, $i \leftarrow 1, 9$

- In addition to the tweak on each block, Silver changes the key expansion of AES so that the nonce also influences the round keys:
- $\kappa = \mathrm{AES}_{key}(npub)$
- $roundkey_0 = AESroundkey_0(key) \oplus AESroundkey_1(\kappa)$
- $roundkey_i = AESroundkey_i(key) \oplus AESroundkey_i(\kappa), \quad i \neq 0, 1, 9$
- $roundkey_i = AESroundkey_i(key)$, $i \leftarrow 1, 9$

Some of these details have as objective blocking some attacks. For example:

- We use a mix of the expanded keys of *key* and κ instead of only the expanded keys of κ to prevent a key collision attack.

Some of these details have as objective blocking some attacks. For example:

- We use a mix of the expanded keys of *key* and κ instead of only the expanded keys of κ to prevent a key collision attack.
- We use the plaintext and the ciphertext for the plaintext tag but only the ciphertext (which is never seen by the adversary) for the associated data tag, thus these two parts are treated differently.

Some of these details have as objective blocking some attacks. For example:

- We use a mix of the expanded keys of *key* and κ instead of only the expanded keys of κ to prevent a key collision attack.
- We use the plaintext and the ciphertext for the plaintext tag but only the ciphertext (which is never seen by the adversary) for the associated data tag, thus these two parts are treated differently.
- To further differentiate, the *IC* used is different.

Some of these details have as objective blocking some attacks. For example:

- We use a mix of the expanded keys of *key* and κ instead of only the expanded keys of κ to prevent a key collision attack.
- We use the plaintext and the ciphertext for the plaintext tag but only the ciphertext (which is never seen by the adversary) for the associated data tag, thus these two parts are treated differently.
- To further differentiate, the *IC* used is different.
- The order of the round keys for the tag is different to ensure that that call to the encryption function is not used elsewhere.

Some of these details have as objective blocking some attacks. For example:

- We use a mix of the expanded keys of *key* and κ instead of only the expanded keys of κ to prevent a key collision attack.
- We use the plaintext and the ciphertext for the plaintext tag but only the ciphertext (which is never seen by the adversary) for the associated data tag, thus these two parts are treated differently.
- To further differentiate, the *IC* used is different.
- The order of the round keys for the tag is different to ensure that that call to the encryption function is not used elsewhere.
- Several measures ensure that an attempted forgery must be done with equal lengths texts.

Some of these details have as objective blocking some attacks. For example:

- We use a mix of the expanded keys of *key* and κ instead of only the expanded keys of κ to prevent a key collision attack.
- We use the plaintext and the ciphertext for the plaintext tag but only the ciphertext (which is never seen by the adversary) for the associated data tag, thus these two parts are treated differently.
- To further differentiate, the *IC* used is different.
- The order of the round keys for the tag is different to ensure that that call to the encryption function is not used elsewhere.
- Several measures ensure that an attempted forgery must be done with equal lengths texts.
- The masking of the ciphertext in the construction of *XT* is there to give some protection in the case that the nonce is repeated by mistake.

In cycles per byte (cpb) on Haswell Silver runs at:

- With AESNI instructions
  - encrypts at:
    - 0,73 cpb for long messages
    - 1 cpb for 1536 bytes
    - 10,8 cpb for 44 bytes.
  - decrypts at:
    - 0,81 cpb for long messages
    - 1,2cpb for 1536 bytes
    - 9,6 cpb for 44 bytes.
- Without AESNI the numbers are:
  - 11,45/12,9 cpb for long messages,
  - 11,85/13,59 for 1536 bytes
  - 30,4/28,2 cpb for 44 bytes.

Table of Contents

- CPFB (Counter/Plaintext Feedback) combines CTR y PFB.

- CPFB (Counter/Plaintext Feedback) combines CTR y PFB.
- CTR provides security.

- CPFB (Counter/Plaintext Feedback) combines CTR y PFB.
- CTR provides security.
- PFB gives an authenticator.

- CPFB (Counter/Plaintext Feedback) combines CTR y PFB.
- CTR provides security.
- PFB gives an authenticator.
- PFB is little used partly because it can be vulnerable to a chosen plaintext attack. Its combination with CTR prevents this.

- CPFB (Counter/Plaintext Feedback) combines CTR y PFB.
- CTR provides security.
- PFB gives an authenticator.
- PFB is little used partly because it can be vulnerable to a chosen plaintext attack. Its combination with CTR prevents this.
- CTR and PFB allows paralellization on the encryption, but PFB prevents paralellization on decryption.

- CPFB (Counter/Plaintext Feedback) combines CTR y PFB.
- CTR provides security.
- PFB gives an authenticator.
- PFB is little used partly because it can be vulnerable to a chosen plaintext attack. Its combination with CTR prevents this.
- CTR and PFB allows paralellization on the encryption, but PFB prevents paralellization on decryption.
- Public message number must be a nonce between 8 and 15 bytes.

- CPFB (Counter/Plaintext Feedback) combines CTR y PFB.
- CTR provides security.
- PFB gives an authenticator.
- PFB is little used partly because it can be vulnerable to a chosen plaintext attack. Its combination with CTR prevents this.
- CTR and PFB allows paralellization on the encryption, but PFB prevents paralellization on decryption.
- Public message number must be a nonce between 8 and 15 bytes.
- Key can be 128 or 256 bits.

- CPFB (Counter/Plaintext Feedback) combines CTR y PFB.
- CTR provides security.
- PFB gives an authenticator.
- PFB is little used partly because it can be vulnerable to a chosen plaintext attack. Its combination with CTR prevents this.
- CTR and PFB allows paralellization on the encryption, but PFB prevents paralellization on decryption.
- Public message number must be a nonce between 8 and 15 bytes.
- Key can be 128 or 256 bits.
- Message is split into 96-bit blocks, each one concatenated with a 32 bit counter.

- Initially two keys κ_0, κ_1 are generated from the nonce and key, in maner similar to Silver, but with a counter added.

- Initially two keys κ_0, κ_1 are generated from the nonce and key, in maner similar to Silver, but with a counter added.
- κ_0 is used as encryption key to process the AD, κ_1 to process the message

- Initially two keys κ_0, κ_1 are generated from the nonce and key, in maner similar to Silver, but with a counter added.
- κ_0 is used as encryption key to process the AD, κ_1 to process the message
- If the message is long, it may be necessary to generate more.

- Initially two keys κ_0, κ_1 are generated from the nonce and key, in maner similar to Silver, but with a counter added.
- κ_0 is used as encryption key to process the AD, κ_1 to process the message
- If the message is long, it may be necessary to generate more.
- κ_0 is also used as a mask in the message processing, to prevent a key collision attack, and in the process of the tag.

Encrypt(M, κ_1, κ_0)

- Split message into 96-bit blocks, with last block incomplete if necessary. (no pad)

Encrypt(M, κ_1, κ_0)

- Split message into 96-bit blocks, with last block incomplete if necessary. (no pad)

- $\mathit{stream} \leftarrow \mathrm{AES}_{\kappa_1}(\{0\}^{128})$, $\quad \mathit{counter} \leftarrow 0$
- For $i \leftarrow 1 \ldots n$
    - $C_i \leftarrow M_i \oplus \mathrm{MSB}_{96}(\mathit{stream})$
    - $\mathit{counter} \leftarrow \mathit{counter} + 1$
    - $\mathit{stream} \leftarrow \mathrm{AES}_{\kappa_1}([\mathit{counter}]_{32})$

Encrypt(M, κ_1, κ_0)

- Split message into 96-bit blocks, with last block incomplete if necessary. (no pad)

- $\mathit{stream} \leftarrow \mathrm{AES}_{\kappa_1}(\{0\}^{128})$, $\quad \mathit{counter} \leftarrow 0$
- For $i \leftarrow 1 \dots n$
  - $C_i \leftarrow M_i \oplus \mathrm{MSB}_{96}(\mathit{stream})$
  - $\mathit{counter} \leftarrow \mathit{counter} + 1$
  - $\mathit{stream} \leftarrow \mathrm{AES}_{\kappa_1}\ (M_i || [\mathit{counter}]_{32})$

Encrypt(M, κ_1, κ_0)

- Split message into 96-bit blocks, with last block incomplete if necessary. (no pad)
- $X \leftarrow \{0\}^{128}$
- $stream \leftarrow \mathrm{AES}_{\kappa_1}(\{0\}^{128}), \quad counter \leftarrow 0$
- For $i \leftarrow 1 \ldots n$
  - $C_i \leftarrow M_i \oplus \mathrm{MSB}_{96}(stream)$
  - $counter \leftarrow counter + 1$
  - $stream \leftarrow \mathrm{AES}_{\kappa_1}((M_i || [counter]_{32})$
  - $X \leftarrow X \oplus stream$

Encrypt(M, κ_1, κ_0)

- Split message into 96-bit blocks, with last block incomplete if necessary. (no pad)
- $X \leftarrow \{0\}^{128}$
- $stream \leftarrow \mathrm{AES}_{\kappa_1}(\{0\}^{128}), \quad counter \leftarrow 0$
- For $i \leftarrow 1 \ldots n$
    - $C_i \leftarrow M_i \oplus \mathrm{MSB}_{96}(stream)$
    - $counter \leftarrow counter + 1$
    - $stream \leftarrow \mathrm{AES}_{\kappa_1}((M_i || [counter]_{32})$
    - $X \leftarrow X \oplus stream$
- Return (C, X)

Encrypt(M, κ_1, κ_0)

- Split message into 96-bit blocks, with last block incomplete if necessary. (no pad)
- $X \leftarrow \{0\}^{128}$
- $stream \leftarrow \mathrm{AES}_{\kappa_1}(\kappa_0), \quad counter \leftarrow 0$
- For $i \leftarrow 1...n$
  - $C_i \leftarrow M_i \oplus \mathrm{MSB}_{96}(stream)$
  - $counter \leftarrow counter + 1$
  - $stream \leftarrow \mathrm{AES}_{\kappa_1}((M_i || [counter]_{32}) \oplus \kappa_0)$
  - $X \leftarrow X \oplus stream$
- Return (C, X)

Encrypt(M, κ_1, κ_0)

- Split message into 96-bit blocks, with last block incomplete if necessary. (no pad)
- $X \leftarrow \{0\}^{128}$
- $stream \leftarrow \mathrm{AES}_{\kappa_1}(\kappa_0)$, $\quad counter \leftarrow 0$
- For $i \leftarrow 1 \ldots n$
    - $C_i \leftarrow M_i \oplus \mathrm{MSB}_{96}(stream)$
    - $counter \leftarrow counter + 1$
    - $stream \leftarrow \mathrm{AES}_{\kappa_1}((M_i || [counter]_{32}) \oplus \kappa_0)$
    - $X \leftarrow X \oplus stream$
- If there is a final partial block M^*_{n+1} of length r:
- Return (C, X)

Encrypt(M, κ_1, κ_0)

- Split message into 96-bit blocks, with last block incomplete if necessary. (no pad)
- $X \leftarrow \{0\}^{128}$
- $stream \leftarrow \mathrm{AES}_{\kappa_1}(\kappa_0), \quad counter \leftarrow 0$
- For $i \leftarrow 1...n$
  - $C_i \leftarrow M_i \oplus \mathrm{MSB}_{96}(stream)$
  - $counter \leftarrow counter + 1$
  - $stream \leftarrow \mathrm{AES}_{\kappa_1}((M_i || [counter]_{32}) \oplus \kappa_0)$
  - $X \leftarrow X \oplus stream$
- If there is a final partial block M^*_{n+1} of length r:
  - $C^*_{n+1} \leftarrow M^*_{n+1} \oplus \mathrm{MSB}_r(stream)$
- Return (C, X)

Encrypt(M, κ_1, κ_0)

- Split message into 96-bit blocks, with last block incomplete if necessary. (no pad)
- $X \leftarrow \{0\}^{128}$
- $\mathit{stream} \leftarrow \mathrm{AES}_{\kappa_1}(\kappa_0), \quad \mathit{counter} \leftarrow 0$
- For $i \leftarrow 1 \ldots n$
    - $C_i \leftarrow M_i \oplus \mathrm{MSB}_{96}(\mathit{stream})$
    - $\mathit{counter} \leftarrow \mathit{counter} + 1$
    - $\mathit{stream} \leftarrow \mathrm{AES}_{\kappa_1}((M_i || [\mathit{counter}]_{32}) \oplus \kappa_0)$
    - $X \leftarrow X \oplus \mathit{stream}$
- If there is a final partial block M^*_{n+1} of length r:
    - $C^*_{n+1} \leftarrow M^*_{n+1} \oplus \mathrm{MSB}_r(\mathit{stream})$
    - $\mathit{counter} \leftarrow \mathit{counter} + 1$
    - $\mathit{stream} \leftarrow \mathrm{AES}_{\kappa_1}((M^*_{n+1} || \{0\}^{96-r} || [\mathit{counter}]_{32}) \oplus \kappa_0)$
    - $X \leftarrow X \oplus \mathit{stream}$
- Return (C, X)

$$\mathrm{ProcessAD}(AD, \kappa_0)$$

- Pad AD with zeroes and split into 96 bit blocks.
- $X \leftarrow \{0\}^{128}, \quad counter \leftarrow 0$
- For $i \leftarrow 1 \ldots n$
  - $counter \leftarrow counter + 1$
  - $X \leftarrow X \oplus \mathrm{AES}_{\kappa_0}(AD_i || [counter]_{32})$
- Return X

$$\text{EncryptAndAuthenticate}(AD, M, npub, key)$$

- $(\kappa_0, \kappa_1) \leftarrow \text{GenerateKeys}(npub, key)$

- $X_{AD} \leftarrow \text{ProcessAD}(AD, \kappa_0)$
- $(C, X_M) \leftarrow \text{Encrypt}(M, \kappa_1, \kappa_0)$

EncryptAndAuthenticate($AD, M, npub, key$)

- $(\kappa_0, \kappa_1) \leftarrow$GenerateKeys($npub, key$)
- $X_{AD} \leftarrow$ ProcessAD(AD, κ_0)
- $(C, X_M) \leftarrow$ Encrypt(M, κ_1, κ_0)
- $T \leftarrow \mathrm{AES}_{\kappa_0}(X_{AD} \oplus X_M)$
- Return (C, T)

EncryptAndAuthenticate(AD, M, $npub$, key)

- $(\kappa_0, \kappa_1) \leftarrow$GenerateKeys($npub$, key)
- $mlen \leftarrow |M|/8$, $adlen \leftarrow |AD|/8$
- $X_{AD} \leftarrow$ ProcessAD(AD, κ_0)
- $(C, X_M) \leftarrow$ Encrypt(M, κ_1, κ_0)

- $T \leftarrow \mathrm{AES}_{\kappa_0}(X_{AD} \oplus X_M)$
- Return (C, T)

EncryptAndAuthenticate($AD, M, npub, key$)

- $(\kappa_0, \kappa_1) \leftarrow$GenerateKeys($npub, key$)
- $mlen \leftarrow |M|/8$, $adlen \leftarrow |AD|/8$
- $X_{AD} \leftarrow$ ProcessAD(AD, κ_0)
- $(C, X_M) \leftarrow$ Encrypt(M, κ_1, κ_0)
- $L \leftarrow \mathrm{AES}_{\kappa_0}([mlen]_{64} \,||\, [adlen]_{32} \,||\{0\}^{32})$
- $T \leftarrow \mathrm{AES}_{\kappa_0}(X_{AD} \oplus X_M)$
- Return (C, T)

EncryptAndAuthenticate($AD, M, npub, key$)

- $(\kappa_0, \kappa_1) \leftarrow$GenerateKeys($npub, key$)
- $mlen \leftarrow |M|/8$, $adlen \leftarrow |AD|/8$
- $X_{AD} \leftarrow$ ProcessAD(AD, κ_0)
- $(C, X_M) \leftarrow$ Encrypt(M, κ_1, κ_0)
- $L \leftarrow \mathrm{AES}_{\kappa_0}([mlen]_{64} \,||\, [adlen]_{32} \,||\{0\}^{32})$
- $T \leftarrow \mathrm{AES}_{\kappa_0}(X_{AD} \oplus X_M \oplus L)$
- Return (C, T)

EncryptAndAuthenticate($AD, M, npub, key$)

- $(\kappa_0, \kappa_1) \leftarrow$GenerateKeys($npub, key$)
- $mlen \leftarrow |M|/8$, $adlen \leftarrow |AD|/8$
- $X_{AD} \leftarrow$ ProcessAD(AD, κ_0)
- $(C, X_M) \leftarrow$ Encrypt(M, κ_1, κ_0)
- $L \leftarrow \mathrm{AES}_{\kappa_0}([mlen]_{64} \,||\, [adlen]_{32} \,||\{0\}^{32})$
- $T \leftarrow \mathrm{AES}_{\kappa_0}(X_{AD} \oplus X_M \oplus L)$
- Return (C, T)

Decryption and verification are the obvious ones.

Table of Contents

- Both algorithms came with proofs of security, although the reduction to AES security is tighter for AESCPFB.

- Both algorithms came with proofs of security, although the reduction to AES security is tighter for AESCPFB.
- Both are reasonably fast.

- Both algorithms came with proofs of security, although the reduction to AES security is tighter for AESCPFB.
- Both are reasonably fast.
- Silver is not only faster than AESGCM, it is in fact competitive even with OCB and it appears to be among the group of the fastest CAESAR candidates.

- Both algorithms came with proofs of security, although the reduction to AES security is tighter for AESCPFB.
- Both are reasonably fast.
- Silver is not only faster than AESGCM, it is in fact competitive even with OCB and it appears to be among the group of the fastest CAESAR candidates.
- They both benefit from whatever improvement in speed, area, energy consumption, etc, to AES.

- Both algorithms came with proofs of security, although the reduction to AES security is tighter for AESCPFB.
- Both are reasonably fast.
- Silver is not only faster than AESGCM, it is in fact competitive even with OCB and it appears to be among the group of the fastest CAESAR candidates.
- They both benefit from whatever improvement in speed, area, energy consumption, etc, to AES.
- The basic idea is simple in both: combine CTR with PFB in one, change three round keys in the other.

- Both algorithms came with proofs of security, although the reduction to AES security is tighter for AESCPFB.
- Both are reasonably fast.
- Silver is not only faster than AESGCM, it is in fact competitive even with OCB and it appears to be among the group of the fastest CAESAR candidates.
- They both benefit from whatever improvement in speed, area, energy consumption, etc, to AES.
- The basic idea is simple in both: combine CTR with PFB in one, change three round keys in the other.
- In both cases whatever damage is caused by repetition of a nonce is limited to that nonce, i.e., repetition of a nonce X does not affect confidentiality or authentication of messages used with nonce Y.

- Both algorithms came with proofs of security, although the reduction to AES security is tighter for AESCPFB.
- Both are reasonably fast.
- Silver is not only faster than AESGCM, it is in fact competitive even with OCB and it appears to be among the group of the fastest CAESAR candidates.
- They both benefit from whatever improvement in speed, area, energy consumption, etc, to AES.
- The basic idea is simple in both: combine CTR with PFB in one, change three round keys in the other.
- In both cases whatever damage is caused by repetition of a nonce is limited to that nonce, i.e., repetition of a nonce X does not affect confidentiality or authentication of messages used with nonce Y.
- Silver has some resistance against nonce misuse but we have not been able to precisely measure this resistance.

- Both algorithms came with proofs of security, although the reduction to AES security is tighter for AESCPFB.
- Both are reasonably fast.
- Silver is not only faster than AESGCM, it is in fact competitive even with OCB and it appears to be among the group of the fastest CAESAR candidates.
- They both benefit from whatever improvement in speed, area, energy consumption, etc, to AES.
- The basic idea is simple in both: combine CTR with PFB in one, change three round keys in the other.
- In both cases whatever damage is caused by repetition of a nonce is limited to that nonce, i.e., repetition of a nonce X does not affect confidentiality or authentication of messages used with nonce Y.
- Silver has some resistance against nonce misuse but we have not been able to precisely measure this resistance.
- As of the moment of this writing there are no attacks against either.

Thanks! Gracias! Merci!

Kiitos! Danke!