# AES-CPFB v1

Designers: Miguel Montes and Daniel Penazzi

Submitters: Miguel Montes and Daniel Penazzi
mmontes@iua.edu.ar

March 15, 2014

# Contents

**Abstract**

We describe here a cipher mode of operation that provides both privacy and authenticity. AES-CPFB uses a combination of Plaintext Feedback and Counter modes. It uses AES as a black box, and depends on the assumption that the output of AES is indistinguishable from random. In particular, its security regarding privacy is equivalent to that of Counter mode. It requires the use of a nonce, that is, a public message number that is not repeated through the lifetime of the key. There is no need for the nonce to be random. The length of the ciphertext, excluding the length of the tag, is the same as the length of the plaintext.

# Chapter 1

# Specification

## 1.1 Parameters

AES-CPFB has three parameters: key length, nonce length, and tag length. Parameter space: Each parameter is an integer number of bytes[1]. The key length is 16 bytes (128 bits) or 32 bytes (256 bits). The nonce length is between 8 bytes and 15 bytes. The tag length is between 1 byte and 16 bytes, but 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 bytes (128 bits) key, 12 bytes (96 bits) nonce, 16 bytes (128 bits) tag.

## 1.3 Authenticated Encryption

The inputs to authenticated encryption are a plaintext $P$, associated data $A$, a public message number $N$, which must be a nonce, and a key $K$.

The maximum length of A is $2^{32} - 1$ bytes. The maximum length of the plaintext depends on the nonce length, with a hard limit of $2^{64} - 1$ bytes. For nonces of 96 bits or less the maximum length of the plaintext is $2^{64} - 1$ bytes. For 104, 112 and 120 bits nonces, the maximum length is $3 \times 2^{34} \times 2^{125-|N|} - 12$, where $|N|$ is the length in bits of the public message number.

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 $\tau$ (at most 16 bytes, as specified above in 1.1). The length of $C$ is equal to the length of

[1] Although we sometimes use lengths in bits, it must be understood that those values are multiples of 8.

$P$. The total length of the ciphertext is thus the number of bytes in $P$ plus the tag length $\tau$.

We now provide the details.

## 1.4 Notation

$\oplus$ denotes the bitwise exclusive or (XOR).

$MSB_m(X)$ is the bit string consisting of the $m$ most significant bits of the bit string $X$.

$||$ denotes concatenation.

$E_K(X)$ denotes the forward cipher function of a 128 bit block cipher algorithm under the key $K$ applied to the data block $X$. In particular, it is assumed that the block cipher algorithm is AES.

$|X|$ denotes the length in bits of the bit string $X$.

$\{0\}^n$ denotes a bit string of zeros of length $n$.

All quantities representing counters and lengths are considered as non negative integers with the least significant bit on the right (big-endian).

## 1.5 Treatment of the public message number

The public message number (nonce) will be used together with the key $K$ to create a sequence of keys $\kappa_0, \kappa_1, \ldots, \kappa_k$, where $|\kappa_i| = |K|$. The nonce length must be an integer number of bytes between 8 and 15. That is, the minimum nonce length is 64 bits, and the maximum is 120 bits. The nonce is used to produce a 128 bits nonce, padding to 15 bytes with zeros, and adding a byte with a value unique for each nonce length. There are 8 possible nonce lengths, so only the 3 least significant bits of the last byte are used to this end. The padded bits are used as a counter, so the number of keys that can be generated from a pair $(K, N)$ is $2^{128-3-|N|}$, although the algorithm never uses more than $2^{32}$ keys.

The key $\kappa_0$ is the encryption of a string of zeros under the key $K$ using AES[2] in CBC mode, using the extended nonce as IV. The encryption function is used once, for generating 128 bit keys, or twice, for 256 bit keys. The padded bits are then used as a counter to generate the keys $\kappa_1, \ldots, \kappa_k$. $\kappa_1$ can be obtained incrementing the 128 bit nonce by 8 (so the three bits indicating the nonce length are untouched). Note that each $\kappa_i$ itself is a nonce, i.e., if $N$ does not repeat within the lifetime of a key, neither does any $\kappa_i$. The only difference is that some adversary could have control over $N$, but not over $\kappa_i$.

Let $l$ be the 3 bit tag identifying the nonce length. Then

$$\begin{aligned} C_0 &= N||i||l \\ C_1 &= E_K(\{0\}^{128} \oplus C_0) = E_K(N||i||l) \\ C_2 &= E_K(\{0\}^{128} \oplus C_1) = E_K(E_K(N||i||l)) \end{aligned}$$

[2] AES-128 or AES-256, depending on $|K|$

So, for 128 bit keys

$$\kappa_i = E_K(N||i||l)$$

and for 256 bit keys

$$\kappa_i = E_K(N||i||l)||E_K(E_K((||i||l))$$

Table 1.1 shows the allowed nonce lengths, the tag identifying each length, and the maximum plaintext length associated with that length.

| Nonce length (bits) | Tag | Counter length (bits) | Max P length (bytes) |
|---|---|---|---|
| 64 | 0 | 61 | $2^{64} - 1$ |
| 72 | 1 | 53 | $2^{64} - 1$ |
| 80 | 2 | 45 | $2^{64} - 1$ |
| 88 | 3 | 37 | $2^{64} - 1$ |
| 96 | 4 | 29 | $2^{64} - 1$ |
| 104 | 5 | 21 | $3 \times 2^{55} - 12$ |
| 112 | 6 | 13 | $3 \times 2^{47} - 12$ |
| 120 | 7 | 5 | $3 \times 2^{39} - 12$ |

Table 1.1: Nonce lengths

## 1.6 Treatment of the Associated Data

The associated data is padded with as many zeros as needed to bring its length to the next multiple of 96 bits. If the original length is a multiple of 96 bits, no padding is needed. Then it is partitioned in blocks $A_i, i = 1, \ldots, m$ of 96 bits, and each 96 block is concatenated with a 32 bit counter. A partial tag is calculated using the exclusive-or on the results of encrypting these 128 bit blocks with AES-128 under the key $\kappa_0$.

$$\begin{aligned} X_0 &= \{0\}^{128} \\ X_i &= X_{i-1} \oplus E_{\kappa_0}(A_i||i) \qquad i = 1, \ldots, m \end{aligned}$$

$X_m$ will be used later to obtain the tag.

## 1.7 Encryption of the plaintext

The plaintext is padded with as many zeros as needed to bring its length to the next multiple of 96 bits. Then it is partitioned in blocks $P_i, i = 1, \ldots, n$ of 96 bits. If the original length is a multiple of 96 bits, no padding is needed. Each

block is concatenated with a 32 bit counter, xored with $\kappa_0$, and encrypted with a key $\kappa_j$, generating a key stream.

$$
\begin{aligned}
P_0 &= \{0\}^{96} \\
I_i &= P_{i-1}||((i-1) \mod 2^{32}) && i = 1, \ldots, n+1 \\
O_i &= E_{\kappa_j}(I_i \oplus \kappa_0) && i = 1, \ldots, n+1 \qquad j = \lfloor i/2^{32} \rfloor + 1 \\
C_i &= P_i \oplus MSB_{96}(O_i) && i = 1, \ldots, n-1 \\
C_n^* &= P_n^* \oplus MSB_u(O_n)
\end{aligned}
$$

For the last block, which may be a partial block of $u$ bits, only the $u$ most significant bits of the corresponding output block are used for the exclusive-or. A partial tag is calculated using the exclusive-or on the output blocks.

$$
\begin{aligned}
Y_0 &= \{0\}^{128} \\
Y_i &= Y_{i-1} \oplus O_{i+1} \qquad i = 1 \ldots n \\
&= Y_{i-1} \oplus E_{\kappa_j}((P_i||(i \mod 2^{32})) \oplus \kappa_0)
\end{aligned}
$$

## 1.8 Decryption of the ciphertext

Decryption is as follows:

$$
\begin{aligned}
P_0 &= \{0\}^{96} \\
I_i &= P_{i-1}||((i-1) \mod 2^{32}) && i = 1, \ldots, n+1 \\
O_i &= E_{\kappa_j}(I_i \oplus \kappa_0) && i = 1, \ldots, n+1 \qquad j = \lfloor i/2^{32} \rfloor + 1 \\
P_i &= C_i \oplus MSB_{96}(O_i) && i = 1, \ldots, n-1 \\
P_n^* &= C_n^* \oplus MSB_u(O_n)
\end{aligned}
$$

For the last block, which may be a partial block of $u$ bits, only the $u$ most significant bits of the corresponding output block are used for the exclusive-or. A partial tag is calculated using the exclusive-or on the output blocks.

$$
\begin{aligned}
Y_0 &= \{0\}^{128} \\
Y_i &= Y_{i-1} \oplus O_{i+1} \qquad i = 1 \ldots n \\
&= Y_{i-1} \oplus E_{\kappa_j}((P_i||(i \mod 2^{32})) \oplus \kappa_0)
\end{aligned}
$$

## 1.9 Computation of the tag

The length *mlen* of $P$ is encoded as a 64 bit big-endian unsigned integer, and is concatenated with the length *adlen* of $A$, encoded as a 32 bit big-endian

unsigned integer, and 32 zero bits. This 128 bit block is then encrypted under $\kappa_0$.

$$L = E_{\kappa_0}(mlen||adlen||\{0\}^{32})$$

Then, the 128 bit tag $T$ is calculated as

$$T_{128} = E_{\kappa_0}(X_m \oplus Y_n \oplus L)$$

Smaller tags can be obtained by taking the desired most significant bits of $T$.

$$T_\tau = MSB_\tau(T_{128})$$

## 1.10 Verification of authenticity

On receiving $(N, A, C, T)$, the receiver computes $T_\tau$ as described in 1.9 and checks $T_\tau = T$.

# Chapter 2

# Security Goals

| goal | `aes128cpfbv1` bits of security | `aes256cpfbv1` bits of security |
|---|---:|---:|
| confidentiality for the plaintext | 128 | 256 |
| integrity for the plaintext | 128 | 128 |
| integrity for the associated data | 128 | 128 |
| integrity for the public message number | 128 | 128 |

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

There is no secret message number. The public message number is a nonce. It is safe to use the same key with nonces of different lengths. The cipher does not promise any integrity or confidentiality if the legitimate key holder uses the same nonce to encrypt two different (plaintext, associated data) pairs under the same key.

The reuse of the nonce affects only the messages involved, and does not compromise the security of $K$.

# Chapter 3

# Security Analysis

## 3.1 Privacy

The security of AES-CPFB is based on an assumed property of AES, namely that $B \mapsto MSB_{96}AES_K(B)$ is a 128-bit to 96-bit pseudorandom function.

It is widely assumed that AES outputs for distinct inputs are indistinguishable from random (if the number of inputs does not approach $2^{64}$).

Hence, truncation of the output of AES should also be indistinguishable from random.

Then, confidentiality holds, since it is basically counter mode, except that the input also takes into account the previous plaintext, but the 32 bit counter guarantees that the input behaves as such, and all blocks feeded to the cipher are different. In addition, no more than $2^{32}$ blocks are processed under the same temporal key $\kappa_i$, way below the birthday bound.

More formally, since the various $MSB_{96}(AES_{\kappa_i}(P_i||i))$ are indistinguishable from random, then the ciphertexts blocks $C_i$ are random 96 bit blocks, so AES-CPFB is indistinguishable from random.

## 3.2 Unforgeability

The proof in similar in spirit to the one in [4]. We will prove the following:

**Theorem 3.2.1.** *If a forgery of AES-CPFB by a nonce respecting adversary can be made with probability better than $2^{-\tau} + \epsilon$ then there exists an adversary that with access only to an encryption oracle can distinguish AES from a random block oracle (see definition below) with probability better than $1/2 + \epsilon/2$*

*Proof.* Let *Adv* be an adversary that can create a forgery of AES-CPFB with probability $p$ with $p > 2^{-\tau} + \epsilon$. Here $\tau$ is the length of the tag, and *Adv* can request encryption/authentications by asking queries $(N^j, A^j, P^j)$ with different $N^j$'s after which *Adv* can request a decryption/verification query $(N, A, C, T)$ that was never an answer and wins if is accepted as valid.

Let $Adv^*$ be an adversary that tries to distinguish AES from a (keyed) random block oracle, using only the encryption oracle.

The model of attack of $Adv^*$ is the following: $Adv^*$ will be provided with a family of oracles $O_{N,j}$ where $N$ is any string of bytes of the number of bytes allowed and $j$ an integer. Each $O_{N,j}$ is a permutation on the 128 bit strings (hence $O_{N,j}^{-1}$ exists but it will not be provided to $Adv^*$).

There will be two possible families provided to $Adv^*$ (and which one is given will be determined randomly). One will be the family in which $O_{N,j} = \text{AES}_{\kappa_j}$ and the other will be the "randomblock oracle" family (modeled below).

$Adv^*$ tasks is to be able to distinguish with probability significantly better than $1/2$ between the two families.

We will model "randomblock oracle" in the following way:

For each $(N, j)$ start with $\text{Domain}_{N,j}$, $\text{Image}_{N,j}$ and $\text{Pairs}_{N,j}$ empty. On input $(N, j, B)$ for a call to $O_{N,j}$, the oracle checks to see whether $B$ is in $\text{Domain}_{N,j}$. If not, it outputs a (uniformly) random 128-bit value $C$, with the only restriction that it must not be in $\text{Image}_{N,j}$. Then it adds $B$ to $\text{Domain}_{N,j}$, $C$ to $\text{Image}_{N,j}$ and $(B, C)$ to $\text{Pairs}_{N,j}$. On the other hand if $B \in \text{Domain}_{N,j}$ then it searches for $(B, C)$ in $\text{Pairs}_{N,j}$ and outputs $C$.

$Adv^*$ uses the following strategy: $Adv^*$ takes all queries that $Adv$ does in order to construct a forgery and runs AES-CPFB , except that it replaces all calls to AES by calls to the oracle, giving the result to $Adv$.

When $Adv$ outputs a possible forgery $F$, $Adv^*$ checks to see whether $F$ is effectively a forgery. If it is, $Adv^*$ guesses $O = \text{AES}$. If not, $Adv^*$ guesses $O = \text{randomblock}$

We have 4 possibilities:

1. $O = \text{AES}$ and F is a forgery. By the hypothesis on $Adv$ this happens with probability $p/2$. (the $1/2$ because the oracle is chosen at random between AES and randomblock). In this case $Adv^*$ guesses correctly.

2. $O = \text{AES}$ and F is not a forgery. This happens with probability $(1-p)/2$ and in this case $Adv^*$ guesses incorrectly.

3. $O = \text{randomblock}$ and F is a forgery. Let's call $p^*$ the probability that $F$ is a forgery given that $O = \text{randomblock}$. Then the situation in this item has probability $p^*/2$. In this case $Adv^*$ guesses incorrectly.

4. $O = \text{randomblock}$ and F is not a forgery. This happens with probability $(1-p^*)/2$ and $Adv^*$ guesses correctly.

So, the probability that $Adv^*$ guesses correctly is $(p + 1 - p^*)/2$. We will see below that $p^* \leq 2^{-\tau}$, hence the probability that $Adv^*$ guesses correctly is $(p + 1 - p^*)/2 < 1/2(1 + 2^{-\tau} + \epsilon - 2^{-\tau}) = 1/2 + \epsilon/2$

So, let's compute $p^*$. We are then in the case in which the oracle is randomblock and the forgery succeeds.

We have the following possibilities

1. $N \neq N^r$ for all $r$.

   In this case when computing $L = E_{\kappa_0}(mlen||adlen||\{0\}^{32})$ the call to $E_{\kappa_0}$ will be replaced by a call to $O_{N,0}$.

   Since $N$ was never used before this query, and in this query the calls to $O_{N,0}$ are all of 128 bits blocks whose 32 rightmost bits are not all zero, then $mlen||adlen||\{0\}^{32}$ will not be in $\text{Domain}_{N,0}$. Hence the oracle will return an (almost) uniformly random value (almost because it must be different from the return values of the previous calls). However, since $O_{N,0}$ has been used at most $2^{32}$ times, this is far below the birthday bound, so in fact it will be indistinguishable from a random value.

   Hence the value $X_m \oplus Y_n \oplus L$ will be a uniformly random value, independently of what the values of $X_m$ and $Y_n$ are.

   Since the tag will be computed as $T_{128} = O_{N,0}(X_m \oplus Y_n \oplus L)$ then the correct tag will be the truncation of the encryption of a uniformly random value, so the probability that it is exactly $T$ is $2^{-\tau}$.

   There is one small problem though. This will be encrypted again calling $O_{N,0}$, which has been used before. The 32 rightmost bits of $X_m \oplus Y_n \oplus L$ will be a random string, but once $L$ is returned, they will be some concrete 32 bit string $i$. If the associated data is long enough to have reached $i$, then a call to $O_{N,0}(A_i||i)$ would have already been made. Thus if the 96 leftmost (random) bits of $X_m \oplus Y_n \oplus L$ equal $A_i$, then the return call will not be random, but it will be the return call that was made when $O_{N,0}(A_i||i)$ was called. However $Adv$ never sees that return call, since $Adv^*$ does not reveal the computations involving $A$, so as far as $Adv$ is concerned this is the first time the call was made, so the probability that the output will match the tag that $Adv$ provided is $2^{-128}$.

2. $N$ is some $N^r$ (wlog we may assume $r = 1$).

   Since the output of the last encryption will either be randomly different from $T^1_{128}$ or exactly equal to it, if $Adv$ provides any tag different from (the truncation of) $T^1_{128}$, then the probability of forgery is $2^{-\tau}$ or zero. So we can assume that $T = MSB_\tau(T^1_{128})$.

   We have several sub-cases:

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

   The only calls to the oracle $O_{N,0}$ either in the decryption query or on the encryption query under $N^1$ that have been made before the computation of $L$ involve blocks with their rightmost 32 bits different from the zero string, except for the call made when computing $L^1$ (the $L$ that was calculated during the $N^1$ query), that is $O_{N,0}(mlen^1||adlen^1||\{0\}^{32})$.

However $mlen||adlen||\{0\}^{32} \neq mlen^1||adlen^1||\{0\}^{32}$. This implies that $mlen||adlen||\{0\}^{32} \notin \text{Domain}_{N,0}$ so as in the previous case the return value will be indistinguishable from random and the probability of collision would be $2^{-\tau}$.

(b) $adlen = adlen^1$ and $mlen = mlen^1$ but there is an index $i$ with $A_i \neq A_i^1$.

In that case $(A_i||i) \oplus \kappa_0 \neq (A_i^1||i) \oplus \kappa_0$, so we'll have that $(A_i||i) \oplus \kappa_0 \notin \text{Domain}_{N,0}$ thus the return call would be uniformly random (except that it must be different from the previous call). If this is the only difference of $(A, C)$ with respect to $(A^1, C^1)$ then the input to the computation of $T_{128}$ will be different than the input from the computation of $T_{128}^1$, thus $T_{128} \neq T_{128}^1$ with probability 1.

Thus, if $\tau = 128$ and $Adv$ provides $T = T_{128}^1$ as tag the probability of forgery is 0.

If $\tau < 128$, the probability of forgery is

$P(MSB_\tau(T_{128} \oplus T_{128}^j) = 0) = \frac{|\{w \neq 0 : MSB_\tau(w) = 0\}|}{|\{w \neq 0\}|} = \frac{2^{128-\tau}-1}{2^{128}-1} < 2^{-\tau}$.

If there are changes to other blocks, then the input to the computation of $T_{128}$ will be a uniformly random string, hence since the encryption is a permutation $T_{128}$ will equal $T_{128}^1$ with probability $2^{-128}$. The probability of forgery is:

$$\begin{aligned} p = \quad & P(T_{128} = T_{128}^1) \cdot P\left(MSB_\tau(T_{128}) = MSB_\tau(T_{128}^1) | T_{128} = T_{128}^1\right) + \\ & + P(T_{128} \neq T_{128}^1) \cdot P\left(MSB_\tau(T_{128}) = MSB_\tau(T_{128}^1) | T_{128} \neq T_{128}^1\right) \\ = \quad & 2^{-128} \cdot 1 + (1 - 2^{-128}) \cdot \frac{2^{128-\tau} - 1}{2^{128} - 1} \\ = \quad & 2^{-128} + 2^{-128} \cdot (2^{128-\tau} - 1) \\ = \quad & 2^{-\tau} \end{aligned}$$

(c) The lengths are the same and $A = A^1$ but $C \neq C^1$.

Hence there exists an $i$ such that $C_i \neq C_i^1$ (and let's consider the first one). Then $P_i$ will be different from $P_i^j$, since $P_i \oplus P_i^1 = C_i \oplus C_i^1$. Let $j = \lfloor i/2^{32} \rfloor + 1$. Then $P_i$ will contribute to the checksum used to compute the input to the computation of $T_{128}$ through the term $O_{N,j}((P_i||i) \oplus \kappa_0)$. Since $P_i \neq P_i^j$ then $(P_i||i) \oplus \kappa_0 \neq (P_i^1||i) \oplus \kappa_0$, so we'll have that $(P_i||i) \oplus \kappa_0 \notin \text{Domain}_{N,j}$ thus the return value will be (almost) uniformly random and the analysis proceeds as in the previous case.

□

# Chapter 4

# Features

The cipher has many advantages. It is basically AES in counter mode, with a change to counter mode that allows a fast authentication, so it is conceptually very simple and easy to analyze.

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

It is a mode of operation, so it can use a black box implementation of AES.

It is highly parallelizable for encryption. Moreover, 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. These properties do not hold for decryption.

It can be used online, because there is no need to know the length of the associated data or the length of the plaintext.

The cipher has an incremental property: if one block of plaintext is modified (with the same nonce), then the corresponding ciphertext block needs to be modified in the same bits, the next ciphertext block needs to be recomputed, all other ciphertext blocks are left undisturbed, and the tag needs an extra AES computation and some bit changes.

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 fast enough: although it uses one 128 bit AES operation on each 96 bit block, the average overload over CTR mode is 50%. On an Ivy Bridge machine (Intel Core i7-3770K 3.5 GHz) it encrypts long messages (32 kbytes) at 1.47 cycles per byte (using AESNI), while with 1500 byte messages it runs slightly below 2 cycles per byte. These numbers are for encryption only, the processing of the additional data is better. The decrypting process is slower, due to the inabilty to parallelize, taking about 7.5 cycles per byte for messages of 1500 bytes or longer.

The only arithmetic operation used is the 32 bit modular addition, thus making it easy to implement in both 32 bit and 64 bit architectures.

Any attack to the privacy or authenticity of a message under one nonce does not affect the privacy or authenticity of ciphers encrypted with other nonces.

Only the AES block encryption is needed (not the decryption), this saves both hardware gates and software code.

AES-CPFB shares the above property with AES-GCM, but an advantage over AES-GCM is that no Galois field operations are needed, again saving code in software and gates in hardware, and it can take advantage of the adaptability of AES to different environments, from the latest Intel chips to 32 bit systems and byte oriented environments. It also makes re-keying simpler than in AES-GCM in any environment where the AESNI instructions are not available.

Another advantage over AES-GCM is its message size. Up to $2^{64} - 1$ bytes can be encrypted with a single (key, nonce) pair. Furthermore, there are never more than $2^{32}$ AES operations done with the same key, very far below the birthday bound.

Also, another advantage over AES-GCM is that a tag of length $\tau$ is expected to have full $\tau$ bits of security.

# Chapter 5

# Design Rationale

The designers have not hidden any weaknesses in this cipher.

There were several goals in the design of AES-CPFB . The principal one was to use AES as a black box, that is we wanted AES-CPFB to be a mode of operation. Moreover, we wanted to base the security only on the security of AES, and not in other primitives, such as operations on the Galois Field $GF(2^{128})$.

Secondly, we wanted its speed to be competitive with AES-GCM.

It also should be capable of online processing: it should not need to know the length of $AD$ or $P$ before processing, and also should be able to process both independently.

We decided to combine a counter with an encryption of the plaintext. We found that idea applied in RPC [1], but it came at the cost of increasing the size of the ciphertext. So we thought of using the encryption used by RPC as the key stream of a stream cipher. We found out that a similar design exists in CCFB+H [2], but with severe restrictions on the size of the tag.

In our design, we opted for using the plaintext as feedback. As this mode does not propagate the IV, and we wanted all the encryption to depend on the public message number, we decided to make the key dependent on the nonce. This allow us to use a small counter, but to encrypt large messages by way of re-keying. We wanted to ensure that all the auxiliary keys dependent on one nonce to be different between them, and different for any key used with another nonce and the same key. This is easily achieved using a counter.

This design combines the privacy properties of Counter mode [3] with the dependence of the message of PFB. The concatenation of a block of message with a counter provides protection against chosen plaintext attacks (which is one of the problems of PFB).

AES-CPFB pays the price of using a smaller block size, but the use of 96 bits for the message and 32 for the counter appears to be a good compromise. The security of the cipher is not affected: it is still AES with a 128 bit block. The loss of speed on the cipher is compensated by the simplicity of the tag calculation: only an exclusive-or of the key stream blocks is needed, plus some

constant overload.

The use of temporary keys that are nonce-dependent guarantees that there are no repeated key stream blocks during the lifetime of each key. Furthermore, the procedure ensures that no temporary keys are repeated during the lifetime of the master keys, provided that there is no nonce repetition.

The procedure to generate the temporary keys is secure under the assumption that AES is indistinguishable from random.

An attacker never "sees" any block encrypted with $\kappa_0$, except the final tag.

We decide to apply an exclusive-or of $\kappa_0$ with the plaintext before encrypting with any other $\kappa_i$ to prevent a pre-computation collision attack.

The attacker chooses $2^{64}$ different keys $k$, and computes $b_k = AES_k(0)$. Then the encryption of $2^{64}$ different messages are captured and assume the first plaintext block $P_1^j$ of these are known. Let $C_1^j$ be the first blocks of these ciphertexts. Then the attacker knows $S_1^j = P_i^j \oplus C_1^j$ which is the truncation of $E_{\kappa_1}(0)$.

Then the attacker searches for a collision between some $MSB_{96}(b_k)$ and some $S_1^j$.

If there is one, then $k = \kappa_1$ with probability $2^{-32}$, and the attacker could decrypt the rest of the corresponding ciphertext.

The xor with $\kappa_0$, which cannot be derived from $\kappa_1$, prevents this attack.

We used ambiguous padding, but the length of $AD$ and $P$ is encoded in a special block which is encrypted.

We decided to use as tag the XOR of the keystream but this has some security problems part of which are solved by the xor of the encryption of the lengths and the rest is solved by doing one more encryption of the result.

# 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 understand 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 acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

# Bibliography


[1] Jonathan Katz, Moti Yung *"Unforgeable Encryption and Chosen Ciphertext Secure Modes of Operation"*, Fast Software Encryption Lecture Notes in Computer Science Volume 1978, 2001, pp 284-299

[2] Stefan Lucks, "Two-Pass Authenticated Encryption Faster Than Generic Composition, Fast Software Encryption Lecture Notes in Computer Science Volume 3557, 2005, pp 284-298

[3] Morris Dworking, NIST Special Publication 800-38A Recommendation for Block Cipher Modes of Operation, 2001 Edition

[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