++AE v1.0

Author: Francisco Recacha   Email contact at: frecacha@gmail.com

March, 15th 2014

Abstract: ++AE (read "*plusplusae*") is an original proposal of an authenticated encryption mode designed for submission to CAESAR competition. ++AE is a lightweight AE mode with an almost negligible overhead when compared with only-encryption EBC mode since, basically, only four sums are added per each plaintext block plus one additional block cipher call is needed per each message. Moreover, ++AE supports authentication of optional plain associated data, parallelized computation, allows to avoid padding overheads by means of a bit stealing method and can be used with any block cipher algorithm with arbitrary block and key sizes and with more than generous limits either on both message size and session length.

1. Introduction

++AE is a new authenticated encryption mode design partially based on a couple of previous modes designed by the same author, namely IOBC and IOC. The core principles inherited from its parent modes are the "encrypt with chained redundancy" paradigm and, just from IOC, the combined use of x-or and modular sums in the redundancy chaining path.

The main contributions achieved by ++AE mode, in comparison with IOBC and IOC, are that (a) the cipher computations can be now parallelized, the design includes also (b) optional authentication of plain associated data, (c) optional method for avoiding data padding transmission, (d) optional scalable method for reducing the number of authentication bits accompanying the cryptogram and (e) a deeper and stronger security analysis substantiating the quality of the proposed mode.

Moreover, ++AE computational and operational overhead is kept minimal, being possibly the best one in its class. Firstly, the added computational overhead in comparison with the only-encrypt ECB mode are just four block sums per message block (just two for associated data blocks) and an additional call to the block cipher algorithm to generate a last cryptogram block used as authentication tag (or Modification Detection Code, MDC). Secondly, the keying material required by ++AE is reduced just to the encryption key, the one already required by the underlying cipher algorithm, and a nonce public sequence number associated to each message.

To conclude with this introduction just to mention that, as Conway's *Game of Life* shows, complex universes emerge even (if not always) from minimal sets of very simple rules. In ++AE case, is basically a combination the xor binary sums with the modular additions what raises, supported by a good block cipher, the required complexity to obtain a very sound AE mode.

2. ++AE Specification

2.1 Native ++AE operation

Figure 1 illustrates the encryption and decryption native/ baseline procedures for ++AE mode. This native specification can be upgraded by a number of optional features for authentication of plain associated data, padding elimination and partial reduction of authentication bits. Implementations for these optional features are specified later in this same document.

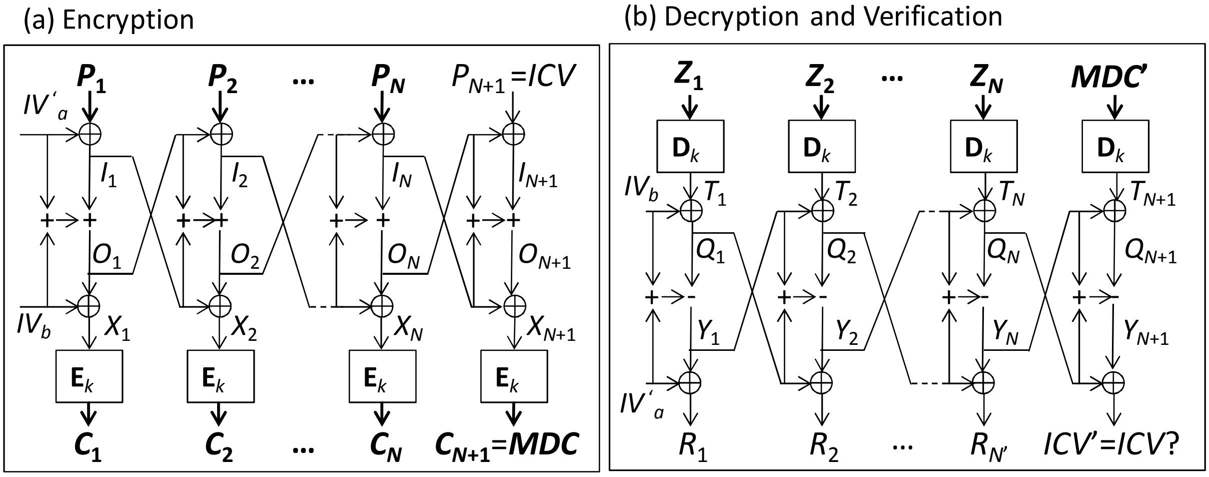

Figure 1: ++AE native encryption and decryption procedures

A formal specification for the baseline encryption procedure:

$$\begin{aligned} I_{\mathrm{i}} &= P_i \oplus O_{i\text{-}1}; \\ O_{\mathrm{i}} &= I_i + I_{i\text{-}1} + O_{i\text{-}1}; \\ X_{\mathrm{i}} &= O_{\mathrm{i}} \oplus I_{i\text{-}1}; \\ C_{\mathrm{i}} &= E_{\mathrm{k}}(\, X_i\,); \end{aligned} \tag{1}$$

for $i = 1, \ldots, (N+1)$ and where

a) P_i is a block of b bits of the plain message and C_i its corresponding cipher-text block;
b) b is the operating block size in bits of the used block cipher algorithm, $E_k(\,)$;
c) $N \le 2^{b/2}$ is the length, in b-bit blocks, of the plain message (see below for padding issue);
d) $O_0 = IV_a$ is a secret and random b-bit value changed for each message;
e) $I_0 = IV_b$ is a secret and random b-bit value changed for each message and different from IV_a;
f) $P_{N+1} = ICV$ (Integrity Check Vector) is a secret random b-bit value changed for each message;
g) $C_{N+1} = MDC$ (Modification Detection Code) is the cryptogram authentication b-bit tag;
h) $E_k(\,X\,)$ is the result of the block encryption of a n bit vector X, using the session key k;
i) $\oplus$ is the x-or binary sum operator applied bit by bit to the two input b-bit vectors;
j) $+$ is the regular arithmetic addition modulo 2^b;

On its turn, the formal specification for the baseline decryption procedure:

$$\begin{aligned} T_i &= D_k(\, Z_i\,); \\ Q_i &= T_i \oplus Y_{i\text{-}1}; \\ Y_i &= Q_j - (Y_{i\text{-}1} + Q_{i\text{-}1}); \\ R_i &= Y_i \oplus Q_{i\text{-}1}; \end{aligned} \tag{2}$$

for $i = 1, \ldots, (N'+1)$ and where

a) - is the regular arithmetic subtraction modulo 2^b.
b) $N' \le 2^{b/2}$ is the number of received cryptogram blocks (i.e. $Z_1, Z_2, \ldots, Z_{N'}$ and $Z_{N'+1} = MDC'$);
c) $R_1, R_2, \ldots, R_{N'}$ and $R_{N'+1}$ are the N' decoded plaintext blocks and the associated ICV' value;
d) $Q_0 = IV_a$ and $Y_0 = IV_b$ as per the encryption procedure;
e) $D_k()$, the inverse operator of $E_k()$ (i.e. $D_k(\, E_k(X)\,) = X$);
f) the decoded plain message is accepted as authentic only if $ICV' = R_{N'+1} = ICV$;

It is immediate that the decoding operation for any authentic cryptogram is just the inverse of the encoding one (i.e. $R_i = P_i$ for i=1 … N and $ICV' = ICV$, with $N' = N$).

Further to the notation already introduced, the following complementary notation is used in this document:

- $X|Y$ is the concatenation of the bit strings X and Y;
- $|X|$ is the length, in bits, of the bit string X;
- X^{*n} is the bit string composed concatenating n instances of the bit string X;
- $X \gg n$ is the right-hand shift of n positions in the bit string X (filling it with '0's on the left);
- $X \ll n$ is the left-hand shift of n positions in the bit string X (filling it with '0's on the right);
- $[X]_n$ is the right-most n bit sub-string (i.e. lest-significant-bits, or simply "lsb") in X;
- ${}_n[X]$ is the left-most n bit sub-string (i.e. Most-Significant-Bits, or simply "MSB") in X;

To conclude ++AE native specification, the mode assumes that the plaintext string has a length multiple of the working block size and no adding is required. Nevertheless, in case the user application delivers a plaintext string that requires its bit tail to be padded, then an optional padding mechanism is available for ++AE according to the specification given in section 2.3.

2.2 Optional authentication of plain associated data

In case that associated data is to be authenticated, then the procedure illustrated in figure 2 is used both by the sender and the receiver to process them.

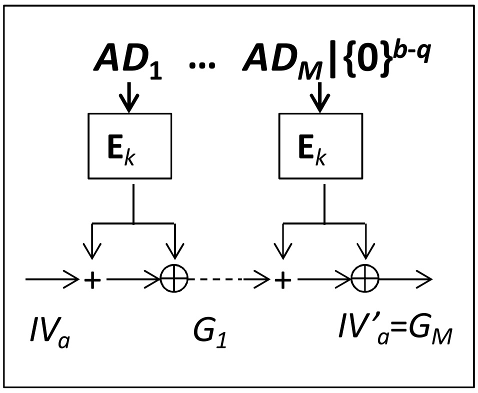

Figure 2: optional authentication of plain associated data

The authentication of the optional associated data consists of computing the following iterative authentication tag over the AD blocks and before the rest of the message is processed:

$$G_i = E_k(AD_i) \oplus (E_k(AD_i) + G_{i-1}); \tag{4}$$

for $i = 1, \ldots, M$, and where

- M, such that $(M+N) \leq 2^{b/2}$, is the block length of the associated data AD;
- $AD_1, AD_2, \ldots, AD_M$ are the b bit blocks in which AD is break down;
- The last block AD_M is filled, if required, with 0s on its lsb positions till completing the block size; i.e. $AD_M = [AD]_q \mid 0^{*(b-q)}$ being q the number of residual bits from AD for the last block;
- $G_0 = IV_a$ (one of the initializing vectors "stolen" from the encryption procedure);
- G_M constitutes an authentication tag that replaces the stolen initializing vector: $IV'_a = G_M$;

2.3 Optional padding with bits stealing from ICV

To maintain the cryptogram bit size as in the original message, the next method is proposed as optional padding mechanism alternative to the native configuration specified above (i.e. alternative to have the plaintext formatted by the user application in block-size multiple length). The idea consists of filling the last plaintext block, P_N, with $(b\text{-}w)$ MSB bits taken from the *ICV* and discarding w bits in the authentication tag *MDC*, where $w<b$ is the number of bits in the plaintext tail to be completed till the block size:

$P_N = [P]_w \mid {}_{(b\text{-}w)}[ICV]$;
$P_{N+1} = ICV$ (as per the native configuration); (5)
$C_{N+1} = [MDC]_{(b\text{-}w)}$; (i.e. $C = C_1 \mid C_2 \mid \ldots \mid C_N \mid (MDC = [MDC]_{(b\text{-}w)})$).

Observe that the last step in the native decryption and verification procedure shall be substituted now with the following two-step one (see Figure 3):

- the blocks $R1, \ldots, R_{N'}$ are decoded normally as per the native decryption procedure;
- $w' = |MDC'|$ is the number of the tail received bits in the last cryptogram block;
- if $[R_{N'}]_{(b\text{-}w')} \neq {}_{(b\text{-}w')}[ICV]$ then the cryptogram is not accepted as authentic, else
- now the encryption procedure is applied to the *ICV* value using the inner vectors obtained at the last decryption step to compute an authentication tag *MDC''*
- finally the decoded message is accepted as authentic only if $[MDC'']_{(b\text{-}w')} = MDC'$.

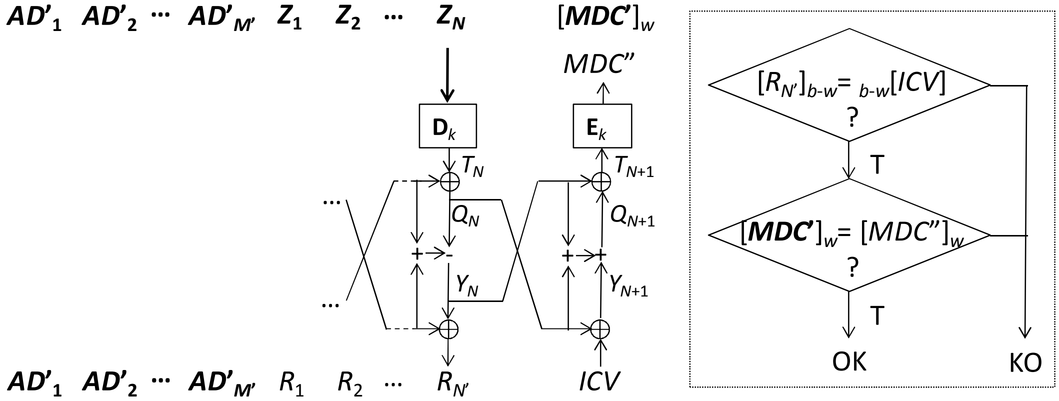

Figure 3: Decryption and verification procedure with padding based on *ICV* bit stealing

2.4 Optional reduction of authentication tag bits

The particular decryption and verification procedure specified above for padding with *ICV* bits stealing can also be directly extended for reducing authentication tags for which only the $h \leq b$ lsb bits are transmitted. Since the authentication strength will be roughly reduced by a 2 factor per each saved bit, this specification recommends not reducing tag size below 64 bits.

Moreover, if this optional authentication tag reduction is used in combination with *ICV* bit stealing padding then, for those messages that $(b\text{-}w) \geq h$ (with w the number of "tail" plaintext bits in the last block P_N), no tag bits will have to be appended since the b bits of the C_N block will contain more authentication bits than required (i.e. $b\text{-}w$). If combination with ICV bits stealing is used, h can be viewed as an "equivalent" tag size since the bits of MDC actually accompanying the cryptogram will get further reduced.

The equivalent tag size can be selected either by the implementer (e.g. hard-wired within the implementation) or parameterized by the user indicating to the encryption and decryption procedures the specific size to be used. For the latter case, the handshake protocol between the sender and receiver applications to set this parameter is kept outside this specification scope.

2.5 Operational requirements for session management

A “security session” is defined as the chain of plain messages that are encoded using a same ciphering session key k and numbered each one of them with a different public message number S, where $0 \leq S \leq 2^{b/2}-1$. This message number S is used for each message as an input nonce to generate its integrity check vector and, in some cases, its initializing vectors. S counter can be coded by the user with any bit word size from $b/2$ to b, although ++AE shall handle it internally as a b bit integer (appending 0s at the leftmost positions if necessary). ++AE specification considers S as a public counter shared by both communication ends. It can be managed as a sequential counter, as a random value, … the only important requirement is that it does not repeat a same value during a given session lifespan.

Although ++AE can be used in a stateless configuration where no internal status information is kept to be reused between calls, the best efficiency and resilience against accidental S repetitions is achieved in its stateful configuration where the last inner vectors O_{N+1} and I_{N+1} are saved to be reused as initializing vectors IV_a and IV_b for the following message (furthermore, in this stateful implementation the implementer has the possibility to save the internal status related to the session key in order to save block cipher rekeying overhead for the subsequent call). Although not required, it is optional to the user to use random and secret S values as a complementary protection in front of accidental repetitions of S.

Note that a signaling protocol will be required between the sender and receiver applications to set the key k as well as to manage the session along its lifecycle, including S counter synchronization when required. This signaling protocol is kept outside the scope of this specification. In any case, if ++AE is used in its stateful configuration the encryption and decryption procedures shall be notified when a new security session is initiated and every time the internal status shall be restarted (see below for fresh generation of the initializing vectors).

A session can encompass up to $2^{b/2}$ messages in order: (a) a same value of the message counter, S, is never repeated during that session and (b) as complementary measure to avoid certain theoretic situations where the inner vectors I_i and O_i could become periodic in the absence of any plaintext input different to ‘0’ (see annexed security analysis for further details). Moreover, the length of each message, N, shall not be greater than $2^{b/2}$ blocks to avoid the mentioned periodicity in the inner vectors. To sum up, one could say that in practical scenarios the message / session limits are more than generous.

In any case, it will be up to the session management protocol to use more restrictive criteria for session limits bellow the $2^{b/2}$ messages threshold. In most practical scenarios this will be the case since a security session will be usually linked to a parallel concept at application or key management level (e.g. file size, duration of a connection-oriented communication protocol, key life timing-out, etc). This means that a session will be terminated usually far more earlier than the $2^{b/2}$ threshold is reached. In any case, it is paramount the session counter is not repeated for two messages in a given session since it could enable an attacker exchanges them (that’s not actually true if the stateful configuration is used since it would be required also that “fresh” IVs are generated for both cryptograms).

2.6 Initializing Vectors Management

2.6.1 Stateless IVs Management

In the ++AE stateless operation, for every message to be processed couple of “fresh” initializing vectors shall be computed as follows (see Figure 4.a):

$$IV_a = E_k(S); \quad IV_b = Ek(IV_a); \tag{6}$$

2.6.2 Stateful IVs Management

The following optional procedure for stateful management of the initializing vectors allows saving for each message the couple of cipher algorithm calls required for fresh *IV*s generation. After generating a couple of fresh IVs for the first session message as per procedure (6) above, for subsequent messages the last inner vectors O_{N+1} and I_{N+1} will be reused (see Figure 4.b):

$IV_a = O_{N+1}$; $IV_b = I_{N+1}$ (and saved to be used for the next session message[1]); (7)

Alternatively to method (7), IV_a and IV_b can be reset with "fresh" values for any particular message using the method specified in (6). This reset shall be done at the beginning of each security session but it can be also forced at any point by the session management protocol. In this latter case, this *IV*s reset may help for instance for resynchronization in case of message losses during a data-loss tolerant communication.

Observe that although stateful operation introduces some burden in the implementation to save the status information between procedure calls for a particular session, there are significant advantages that are worth of this burden:

- In comparison with stateless operation, it saves 2 calls of the cipher algorithm per message;
- In some implementations, it can enable also to save the key preset to process each message;
- It is inherently (partially) resilient to accidental repetitions of the session counter S by poor session management implementations. In the case of stateless operation such repetition, if known by an attacker, would allow to exchange the cryptograms without being detected.

2.7 Integrity Check Vector Management

For each message, the random and secret ICV vector, appended of the plaintext bit string as an additional block, is computed as (see Figure 4.c):

$$ICV = (IV_a \oplus S) + (IV_b \oplus (N + M)); \quad (8)$$

Where N and M are the block lengths of the plaintext and the associated data, respectively.

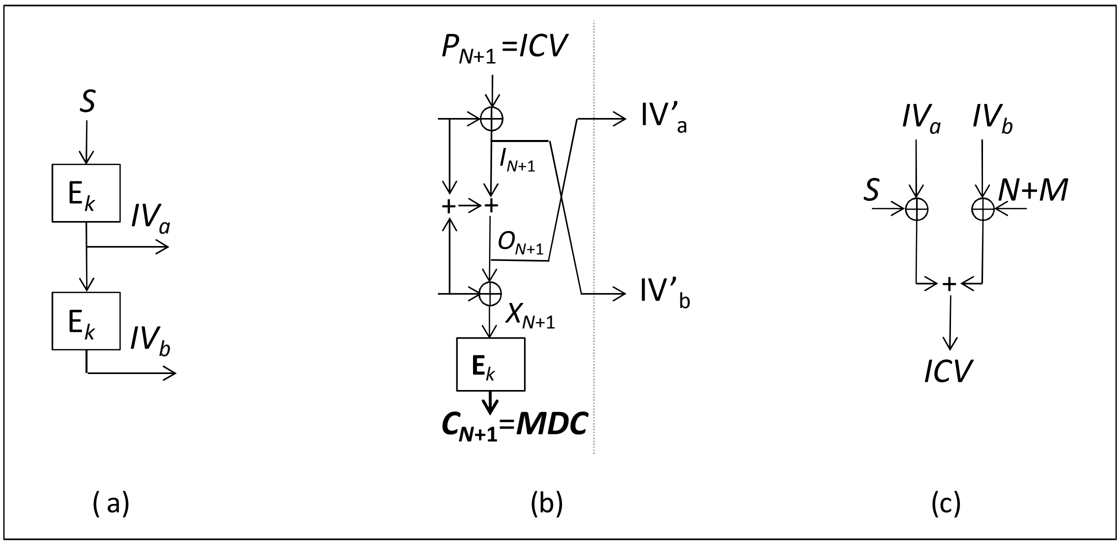

Figure 4: Generation of IOC Initializing Vectors and Integrity Check Vector

[1] Note that depending on particular implementations, the session key configured for the block cipher algorithm may be kept also for the next message to avoid key preset procedure and the associated computing overhead.

2.8 Note on the underlying block cipher algorithm

++AE can be used with any block cipher algorithm and it can operate with any block sizes and key sizes. Moreover, although potential applications and associated requirements have not been assessed, ++AE could be also applicable for some uses with public key algorithms.

3. Security goals and security strength assessment

++AE design pursues, and achieves, the following security goals:

- Confidentiality strength offered to any block of the plaintext data, P_i, is equivalent, or better, to the one offered by the underlying cipher algorithm, $E_k()$ when applied in ECB mode just once to P_i and all the other plaintext blocks processed with the same key being secret and random disregarding the actual values they may have. More precisely:
    - Any plaintext block value being ciphered twice will produce random cryptogram blocks which values would collide just with a (uniform) random probability of 2^{-b};
    - The previous property is achieved even when chosen plaintext is forced by an attacker for the rest of plaintext blocks;
    - ++AE scheme does not leak any information about the value of any of the P_i blocks;
    - ++AE scheme does not leak any information to collect a (X, $E_k(X)$) dictionary.
- Integrity strength both for associated data and plaintext such that whatever resources could be spent to forge ++AE integrity, and assuming an ideal block cipher, the success probability of such attack will not be higher than $2^{-(b-1.25)}$, or $2^{-(h-1.25)}$ if only h bits (with $64 \leq h \leq b$) of the authentication tag are appended to the cryptogram.

The above characterization applies to implementations in ++AE native form or with any of its optional features provided, as usual, that the cipher algorithm performs as a perfect pseudorandom permutation. It also includes chosen plaintext attacks. See annexed document for detailed ++AE security analysis as required per CAESAR call.

++AE only uses a public message number S, as a nonce to generate the initializing vectors, IV_a and IV_b, and the integrity check vector, ICV. Two forms of implementation are supported: stateless and stateful. In the stateless one, 'fresh' vectors IV_a, IV_b and ICV are computed for each message while in the stateful one the initializing vectors IV_a and IV_b are taken from the final computation status of the previous message. Note that while for the stateless operation the repetition of the message counter for two messages in a same session would be fatal if known by an attacker, in the stateful one the chaining of internal variables from one message as the initializing vectors of the following one provides some partial mitigation against this type of accidental nonce repetitions by wrong implementations.

++AE issues regarding possible variations in attack resources, such as software side channels, hardware side channels, large numbers of active keys, relationships among keys, large numbers of legitimate messages encrypted, …, if any, should be the same ones than for the used block cipher. In particular, those issues for ++AE stateful operation are similar to the case of stateful implementation of the cipher algorithm where the session key preset is kept between calls.

4. ++AE Features Summary

It is worth to point out the following ++AE mode differential features:

1. ++AE is one of the most lightweight AE modes ever proposed, especially in its stateful form: in comparison with ECB mode it just adds four b-bit sums per message block, one cipher algorithm call for each message and two more for session start;
2. It allows parallelize all the cipher algorithm calls to achieve high processing speeds.

3. The optional padding mechanism based on ICV bit stealing eludes padding overheads;
4. In comparison with ECB mode, the only additional keying material that is required to be managed by the user is a public message counter S, that is used as a nonce to generate the integrity check vector and, in some cases, the initializing vectors;
5. In its stateful form, ++AE shows partial resilience against accidental repetitions of the message counter S.
6. ++AE allows authentication tags that in native mode have the same size as the working block but it is also supported an optional mode to reduce the number of tag bits.
7. A session can cover up to $2^{b/2}$ messages with a maximum size of $2^{b/2}$ blocks per message.
8. ++AE is applicable with any block cipher algorithm disregarding its block and key sizes;
9. Although neither potential applications nor specific requirements have been assessed, ++AE could be used also with public key cipher algorithms.
10. To the knowledge of the author, the only ++AE performance where it performs below other well established modes is about preprocessing: ++AE does not support neither encryption nor decryption preprocessing.

5. Design Rationale

This section is included just to formally comply with CAESAR requirements since the author has tried to justify ++AE design details in those places of this document where they are specified or, in some cases due to the required extend, in the accompanying security analysis.

The author has not hidden any weaknesses in ++AE design and, to the knowledge of the author, if implemented following the specifications contained in this document, no ++AE design detail would allow hiding any weakness.

6. ++AE Intellectual Property Issues

Intellectual property rights on AE++ scheme, its sub-schemes and some variations are protected by patent application ref. ES P201430169. In any case, the author grants free use rights, at least, for any implementation not being part of (a) a data communications equipment with a market price over 2.017 USD or (b) a commercial Operating System with more than 15.485.863 instances sold. If any of this information changes, the author will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

7. Consent Statement for CAESAR Selection Committee

The submitter 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 submitter 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 submitter 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 submitter acknowledges that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitter understands 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 submitter understands that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.