Annex: ++AE v1.0 - Security Analysis

[Version Draft 0.9]

A.1 Introduction

One of the main security cornerstones of ++AE mode is the behavior exhibited by the combination of binary xor sums with regular modular additions, or equivalently, by the arithmetic operator 'Δ', formerly introduced by the author for IOC mode, and defined as:

$$x \Delta y = (x + y) \oplus (x \oplus y); \qquad \text{(A1)}$$

where $\oplus$ is the regular exclusive-or binary sum, + is the regular modulo 2^b addition, and '–' the regular modulo 2^b subtraction. As it can be easily derived from equation (A1), $x\Delta y$ is just the vector composed by the carries that appear during the modulo 2^b addition of x and y (hence, we will call it as the 'carry-delta' vector).

Sections A.2 and 0 characterize the security of ++AE data confidentiality and integrity based in some cases on the properties of the delta-carry vector that are analyzed in section A.4, where this operator is duly studied.

As general approach, the overall analysis is performed in principle just for native ++AE operation but it is complemented in some cases where the extension of the native case with some optional mechanism is not evident. The general assumptions taken are:

- The cipher algorithm behaves as a pseudorandom permutation randomly selected by the key k from a family of them;
- The session k is a random and secret value and the public message counter S is a nonce;
- IV_a, IV_b and ICV are secret and random values managed according ++AE specifications;
- The maximum length including plaintext and associated data is $2^{b/2}$ blocks.
- No assumptions on plaintext: where it is not a goal per-se, it can be known, or even chosen, by the attacker.

It shall be remarked that the analysis here presented does not follow any of the usual schemes from what is understood as "provable security" by the specialized community. Basic notions and known results on arithmetic and probability that can be followed by almost anyone (with a bit of patience) are used instead. The reason behind is quite simple: ++AE author is professionally making a living in activities quite far from cryptography and he is completely outdated with the academic constructs that have consolidated in the field during the last some decade. To override this limitation, and taking profit from the very simple structure and components of ++AE, it has been tried to follow a well structured treatment based on extremely simple steps.

The followed approach probably misses a part of the conceptual accuracy the current provable security toolbox would have provided but, being pragmatic, this could be considered also an advantage indeed since, if it has been rightly done, it could open a critical review from a broader community than if just based on a highly specialized bunch of sophisticated notations and concepts. In any case, the purpose of this document is neither to introduce any new methodology to build security proofs nor to open any discussion on appropriate methodologies. It is just to build a sound security demonstration specific for ++AE mode and it shouldn't be a showstopper if the job has been rightly done. In any case, the author admits that a complementary security proof using the tools consolidated by the provable security tradition could lead to a richer insight of ++AE security. Thus, any demonstration in that line would be absolutely welcome, as well as any independent demonstration or critical review.

A.2 Plaintext Confidentiality

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 given 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.

A.2.1 Demonstration

The encryption procedure used by ++AE is a composition of the ECB mode (i.e. $C_i = E_k(X_i)$) with a previous chained transformation of the plaintext given by

$$\begin{aligned} I_i &= P_i \oplus O_{i-1}; \\ O_i &= I_i + I_{i-1} + O_{i-1}; \qquad \text{with } O_0 = IV_a \text{ and } I_0 = IV_b \\ X_i &= O_i \oplus I_{i-1}; \end{aligned} \tag{A2}$$

In the worst case, this transformation would not introduce any additional entropy in the X blocks with respect P ones and, from where it is obvious that ++AE confidentiality strength will be at least the one offered by ECB mode. Nonetheless, since the initializing vectors, $IV_a = O_0$ and $IV_b = I_0$, are random and secret it is possible, at least in principle, that some of their entropy is incorporated in the Xs output vectors and, hence, the "base" ECB confidentiality strength would get improved. Let's see that this is the case.

If we look at (A2) as a recursive process, we can view it as separated into three subprocesses:

- An iterative internal background flow composed by the sequencing of the internal vectors, Is and Os. This sequence evolves at each step, by means of the + operator (actually, this evolution is also impaired by the plaintext blocks as shown below);
- An input subprocess that injects at each step the corresponding P_i value to the internal "flow" by means of a $\oplus$ operator that "disturbs" somehow the internal flow;
- Finally, a third subprocess 'extracts' the Xs vectors also as a $\oplus$ sum of the internal vectors. Observe that this last extraction process does not 'disturb' at all such internal process[2].

Observe that to transform a P_i block to the corresponding X_i, it has to "cross the river" through the pseudorandom flow composed by the I_i and O_i sequences and the data gets altered in the process with the status of the random flow and this status by the data crossing: while the background flow is "animated" by the + operator, the plaintext blocks cross it by the xor operator and that will make that mutual alterations are not linear and very dependent on the specific position they take place.

In order to gain further comprehension of this process, let's focus for a moment on the internal vectors flow when they are not disturbed from outside (i.e. with injected plaintext). In such case assuming that all the plaintext blocks are just 0s we have $O_i = I_i + I_{i-1} + O_{i-1}$ and $I_i = O_{i-1}$ or, just to simplify things:

[2] Note that the injection / extraction subprocesses invert their roles in the decryption process.

$I_i = 2 \cdot I_{i-1} + I_{i-2}$; with $I_1 = IV_a$ and $I_0 = IV_b$

for i=2, ..., N (A3)

$O_i = 2 \cdot O_{i-1} + O_{i-2}$; with $O_1 = IV_a + IV_b$ and $O_0 = IV_a$

Equation A3 points out the interesting and paramount fact that the "background" evolution of the inner vectors follows a 2nd order recursive linear evolution defined as a particular form of a generalized Fibonacci / Lucas series but using as initial terms the initializing vectors. In fact, given that $I_i = O_{i-1}$ they are the same sequence shifted 1 step and we will focus just on I_i.

That is, when all the plaintext blocks are just 0s, I_i takes the values:

$I_i = A_i \cdot IV_a + B_i \cdot IV_b$; (A4)

where
- $A_i = 2 \cdot A_{i-1} + A_{i-2}$ and $A_1 = 1$, $A_0 = 0$;
- $B_i = A_{i-1}$;

To sum up, the sequence of the inner vectors I_i is a linear combination of both initializing vectors. Moreover, each one of the coefficients multiplying the initializing vectors IV_a and IV_b follow the same 2nd order linear recurrence but now with the initial values 1 and 0 (B_i is just a one-step delayed copy of A_i). That type on recurrence sequence adjusts to Binet forms where:

$U_i = m \cdot U_{i-1} + U_{i-2}$ with $U_1 = 1$, $U_0 = 0$;

Those sequences can be expressed with their non recursive Binet form:

$U_i = \frac{\alpha^i - \beta^i}{\delta}$;

with
- $\propto = \frac{m+\delta}{2}$;
- $\beta = \frac{m-\delta}{2}$;
- $\delta = \sqrt{m^2+4}$;

And, therefore, the coefficients A_i sequence with m=2 can be written:

$A_i = \frac{(1+\sqrt{2})^i - (1-\sqrt{2})^i}{2\sqrt{2}}$; ; (A5)

with
- $\propto = 1 + \sqrt{2}$;
- $\beta = 1 - \sqrt{2}$;
- $\delta = 2\sqrt{2}$;

If these linear recurrent sequences operate in some modulo r then they become periodic: once a consecutive couple of values is repeated (there are only r^2 possible couples) the sequence becomes periodic (let call (m,r) to this period length). Moreover, since the recurrence is reversible the periodicity starts as soon as the values 0 and 1 appear again in our coefficients sequence making, on its turn that the I_i / O_i sequences repeat periodically with that period (but due to that their initial values are not 0 and 1 but IV_a and IV_b, their period could be even smaller).

Fortunately, our coefficient sequence A_i (and, consequently, B_i), whit m=1 and r =2^b, has a period λ=r=2^b according partial empirical verification done and with theoretical basis from:

- L.A.G Dresel. *On pseudoprimes related to generalized Lucas sequences*. The Fibonacci Quarterly 35, 1; 35-42.1995

Nevertheless, although the coefficients period is significantly large (λ=2^b), it shall be taken into account that depending on actual values of the initializing vectors, the I_i / O_i sequences exhibit smaller periods of value λ_s=2^{b-s} (let's call them "subperiods" of the inner vectors).

This strong dependence with the actual values of the initializing vectors introduces some burden in order to analyze the behavior of such subperiods. Fortunately, since IV_a and IV_b are random by definition, this allows at least an empirical characterization of the subperiods. Preliminary empirical evaluation performed indicates that the probability of having some subperiod of length λ_s=2^{b-s} is $\vartheta(2^{-s})$.

> Conjecture: P(λ_s=2^{b-s}) = $\vartheta(2^{-s})$ if IV_a and IV_b are random.
> Note on Annex Version Draft 0.9: only a preliminary empirical evaluation for values of b from 3 up to 19 performed with random values of IV_a and IV_b is available at the moment of releasing this draft version. This preliminary evaluation indicates that for T random combinations of the initial vectors, the T of them exhibit period 2^b, $T/2$ 2^{b-1}, $T/4$ 2^{b-2}, and so on. A more complete evaluation covering all cases of interest is ongoing and will be completed for a final version 1.0.

That means that if the message length is limited to $2^{b/2}$ and assuming that all the plaintext blocks contain 0s then the probability of having a repetition of the inner vectors during the process of that message would be $2^{-b/2}$, that is, negligible. Moreover, if as a complementary restriction, we reduce the number of possible initializing vectors by means of establishing a limit in the maximum session duration of $2^{b/2}$, the probability of having this theoretic issue to become a practical issue in real operation would be roughly $\vartheta(2^{-b})$.

The overall result is a background flow of random and non-repeating inner vectors that the message plain data shall traverse over xor operators before being ciphered.

This random flow guarantees that even without the presence of any entropy at all in the plaintext, the cryptogram will be full of it, since no cryptogram block will repeat (apart of those spurious and innocuous cases that the birthday paradox makes that a same value for X_i is repeated because of some "lucky" combination of O_i and I_{i-1} values).

At this point, ++AE confidentiality goals stated in section A.2 become evident thanks to the previous transformation of, even null-entropy, P_i blocks in pseudorandom, non repeating and unpredictable X_i ones that are then ciphered by the block cipher $E_k()$in ECB mode to produce the final cryptogram blocks.

To finalize with ++AE confidentiality analysis, observe that if access to the X_i vectors was possible, then some adaptive chosen text attack could try to combine arithmetically these Xs values in order to reset the internal flow to some weak state to leak information about the P_i. In such case, ++AE confidentially level would be reduced down to ECB mode. But, the fact in ++AE is that the Xs vectors are secret and unpredictable if the underlying block cipher is a "good" one. Hence, chosen plaintext attacks, or even adaptative chosen plaintext attacks, do not provide any advantage compared with guessing at random.

A.2.2 Alteration caused by the plaintext in the inner vectors sequence

Although not strictly required to substantiate ++AE plaintext confidentiality strength, it is interesting at this point to have a look on how the background random flow, that is, the sequences of the inner vectors, gets modified by the injection of non null plaintext blocks.

The following paragraphs analyze both the impairment on the background flow and how that background flow transforms P_i blocks into X_i ones. To simplify things, let' suppose that all the plaintext blocks are 0 but $P_i \neq 0$, then we have:

$$\begin{aligned}
&I_i = P_i \oplus O_{i\text{-}1} = P_i \oplus I'_i = I'_i + \delta_i^I;\\
&O_i = I_i + I_{i\text{-}1} + O_{i\text{-}1} = (I_i + I'_i) + I'_{i\text{-}1} = (\delta_i^I + 2 \cdot I'_i + I'_{i\text{-}1} = O'_i + \delta_i^O;\\
&X_i = O_i \oplus I_{i\text{-}1} = (((P_i \oplus I'_i)_i + I'_i) + I'_{i\text{-}1}) \oplus I'_{i\text{-}1} = X'_i \oplus \delta_i^X;
\end{aligned} \qquad \text{(A6)}$$

Where

- I'_i, O'_i and X'_i are the corresponding "background" i-th vectors that would be obtained if P_i=0;
- δ_i^I, ($\delta_i^O = \delta_i^I$) and δ_i^X are the "alteration" vectors produced by the transformation of P_i into X_i.

That is, when P_i traverses the random sequencing chaining of *I*s and *O*s vectors, the following chain of events happens:

- The vector I_i is altered by the xor sum with P_i. That means, in arithmetic terms, that if the *j*-th from P_i is 1, then depending whether the same bit in $O_{i\text{-}1}$ is also set, or not, I_i will be decremented or incremented, respectively, with the value 2*j*. Since O_i is random and unpredictable, the alteration introduced in I'_i vector will be a vector $\delta_i^O = \delta_i^I$ with as many random bits as the entropy contained in P_i, and what is more important, that alteration will not be "linear" neither in terms of xor sums nor in terms of arithmetic ones.
- O_i is altered (linearly) with same entropy variation than Ii, since in fact ($\delta_i^O = \delta_i^I$);
- Finally, the X_i vector suffers also a non-linear alteration with the same amount of entropy that will be transformed in a maximum entropy difference even in the case that Pi contains just a few '1' bits. Observe that although the entropy alteration of X_i vector is just the same that the entropy contained in P_i, the "background" X_i value (i..e the one with all P_i=0) has already high entropy due to the random and secret nature of the initializing vectors and the behavior of the inner vectors recurrence sequence.

Although the "alterations" in the I_i and O_i vectors can be small, it will cause a chaotic avalanche in subsequent steps caused by three main factors:

- δ_i^I / δ_i^O alteration does not depend linearly on *Pi* value;
- the produced modification in the inner vectors is equivalent in some way to substitute the initial values of the linear recurrent sequences that the inner vectors are, and therefore their behavior will change in an unpredictable way for the next steps / blocks;
- when the new background flow finds a new non-null P_t block, the alteration δ_i^I / δ_i^O introduced by P_t will be unpredictable and non-linearly dependant depending on the values of previous plaintext blocks;

In any case, if the entropy contained in P_i is small, the chaotic "avalanche" it causes in the inner sequences will require several steps to acquire a major magnitude and the actual behavior will depend on the sequence of the plaintext blocks and the unpredictable behavior of the inner sequences. Nonetheless, this behavior that is not critical for the confidentiality service will become paramount for integrity one. For that case we will see that during the decryption of a cryptogram we face an equivalent scenario but with the low-entropy plaintext blocks substituted by a high entropy blocks produced by the deciphering boxes D_k(). Then, any change in the injected sequence will not cause an "avalanche" but an "earthquake".

A.3 Data Integrity Analysis

A.3.1 Integrity Strength for the Associated Data

Disregarding the computational resources a potential attacker could spend to build fake associated data *AD*' , the best chance to achieve such purpose would not be better than $2^{-(h-1.25)}$, where *h* is the number of authentication bits appended to the message (*b* bits if the native mechanism if a block-wide authentication tag is used with, or without, *ICV* bits stealing padding).

A.3.1.1 Demonstration

Figure 2 shows how the associated data blocks AD_1, ..., AD_M are processed in order to protect their integrity. For each block AD_i an accumulated authentication tag, G_i, is calculated as a combination of the tag of the previous block, G_{i-1}, and the result of ciphering that block, $E_k(AD_i)$. Taking and transforming equation (4) to express it in terms of the delta operator:

$$G_i = G_{i-1} \oplus (\ G_{i-1} \oplus E_k(AD_i)) \oplus (E_k(AD_i) + G_{i-1}) = G_{i-1} \oplus (\ G_{i-1}\ \Delta\ E_k(AD_i)\ ) = G_{i-1} \oplus \Delta_i; \qquad \text{(A7)}$$

That is, the combination above introduced takes the form of the xor sum of the tag up to the previous block with the delta vector applied to it and to the ciphered version of AD_i.

Finally, the initial G_0 takes the value from one of the ++AE initializing vectors, IV_a and the last AD tag block, G_M, takes the role of the corresponding initializing vector for the encryption process of the plaintext *P*, $IV'_a = G_M$.

A useful expression that puts some light on the AD authentication mechanism tag:

$$G_M = G_{M-1} \oplus \Delta_M = G_{i-2} \oplus \Delta_{i-1} \oplus \Delta_i = IV_a \oplus \Delta_1 \oplus \Delta_2 \oplus \ldots \oplus \Delta_M; \qquad \text{(A8)}$$

NOTE: This demonstration is based on the (temporal) assumption that if a fake associated data, *AD'*, generates a $G'_M \neq G_M$. This fake *AD* will be later detected in the end of the overall decryption and verification process. In section A.3.2.1 is further analyzed that process.

Since the initial G_0 and the outputs of the block cipher are random and secret, it is obvious at this point that so are the subsequent G_i blocks. And given the entropy properties of the delta vector presented in section A.4, at the *i*-th step *b* random bits from the previous step are xor-summed with (*b*-1.25) random bits from the delta operator that, as it will be shown later, strongly depend both on the ciphered version of AD_i and on the actual position they are injected in the chain. At the end of the AD process, the last tag, G_M, will contain (*b*-1.25) random bits strongly depending both on each one of the associated data blocks and their position within *AD* string. The remaining 1.25 bits up to completing the block size will be also random and secret but will be determined mainly by the rightmost bit of *IVa* (in fact, $[G_M]_1 = [IV_a]_1$ since the rightmost bit of any delta vector is always 0).

It is also obvious that a forgery attack cannot be based in any manner on synthesizing fake values for any associated data block (as combination of *AD* contents, neither authentic nor false). The reason is just as simple as the block cipher algorithm, *Ek*(), would introduce (*b*-1.25) random bits totally unpredictable in the tags chaining. Once we have discarded fake AD block synthesis by means of combination of several other blocks, there remain just three types of integrity attacks to be analyzed:

- Removal of associated data (either a block or the complete *AD* string);
- Insertion of a spurious block.
- Insertion of some authentic block (either from the same string or a different taken from the same session) in a fake position;

In the case of complete removal of the associated data, it is obvious that an uncontrollable error of (b-1.25) bits will be introduced in the initializing vector, since $IV'_a = IV_a$ will be used instead of the right value $IV'_a = G_M$. So no more attention will be paid to this simple case apart than as an integral part of the analysis of the strength of AE++ plaintext integrity.

For the remaining cases it is also obvious that either the removal or insertion of an *AD* block, will also introduce (b-1.25) erroneous bits in the tag chaining. In order to cancel that error, if such thing is possible, it will have to be cancelled introducing, at least, a second modification in the associated data. Fortunately, since the error is unknown and the cipher block is unpredictable, the attacker will not capable to synthesize a value for a different block as function of *AD* and the only chance the attack progress is requires that the error is compensated by:

a) inserting the removed block in a different position. That event would happened if, and only if, G_M depends only on AD_i values but not on their specific order;
b) repeating the inserted block in a different position. It is immediate from (A8) that the error compensation would happen if, and only if, $G_i = G_{i-1} \oplus f(AD_i)$ to have a self-correcting error duplication[3];
c) In the case of insertion of an authentic AD_i block in a non authentic position, either repeating it at a different position (equivalent to case (b)) or removing it from its original position (equivalent to case (a)).

With regard to the repetition strategy, since if y is a random value then $x\Delta y = f(x)$ only with a probability of $2^{-(b-1.25)}$, the attack will have that success probability. So, repeating a given block twice does not improve the attack chances.

On the other hand, the reordering approach requires on its turn that G_M does not depend on the specific ordering of the *AD* blocks. Let's see under which circumstances this can happen and that the probability associated to that event is also negligible.

Let's suppose that G_M does not depend on the particular ordering of AD_i blocks. To reflect that fact, we can write:

$$G_M = IV_a \oplus \Delta_1 \oplus \Delta_2 \oplus ... \oplus \Delta_M = f_M(IV_a, AD_1, AD_2, ..., AD_M) \qquad \text{(A9)}$$

where $f_M()$ is some particular application over M+1 b-bit binary variables.

On the other hand, since independently of the actual position of AD_M the correct final tag is given by $G_M = (IV_a \oplus \Delta_1 \oplus \Delta_2 \oplus ...) \oplus \Delta_M$ where the first term between brackets does not depend on AD_M, we can derive that if the application $f_M()$ in (A9) exists then there exists also a $f_{M-1}()$ application such that:

$$G_{M-1} = IV_a \oplus \Delta_1 \oplus \Delta_2 \oplus ... \oplus \Delta_{M-1} = f_{M-1}(IV_a, AD_1, AD_2, ..., AD_{M-1})$$

An the process can be iterated down to $G2$, concluding that if such application $f_M()$ exists, then there exists also a $f_2()$ application such that:

$$G_2 = IV_a \oplus \Delta_1 \oplus \Delta_2 = f_2(IV_a, AD_1, AD_2);$$

[3] Observe this is a particular version of the more general condition stated for case (a), that is, that G_M depends only on AD_i values but not on their specific order.

Let's see that such $f_2()$ application does not exists and, therefore and by *reductio ad absurdum,* no $f_M()$ application exists and, thus, G_M does depend on specific AD blocks ordering. If AD1 and AD2 are interchanged, then the tag G'_2 obtained at the second step would be:

$G'_2 = G'_1 \oplus (G'_1 \Delta E_k(AD_1)) = (-G_0 \oplus (G_0 \Delta E_k(AD_2))) \oplus ((G_0 \oplus (G_0 \Delta E_k(AD_2))) \Delta E_k(AD_1)) =$ ❶

If we assume linearity in the delta-carry vector, i.e. that $(x \oplus y)\Delta z = (x \Delta z) \oplus (y \Delta z)$, then:

❶ $= G_0 \oplus (G_0 \oplus E_k(AD_2)) \oplus (G_0 \Delta E_k(AD_1)) \oplus ((G_0 \Delta E_k(AD_2)) \Delta E_k(AD_1)) =$ ❷
and, since the combination of the first and third terms is just the tag G_1:

❷ $= G_1 \oplus (G_0 \oplus E_k(AD_2))) \oplus ((G_0 \Delta E_k(AD_2)) \Delta E_k(AD_1)) =$ ❸

If we assume now in the delta-carry vector is transitive, i.e. that $(x \Delta y)\Delta z = (x \Delta z)\Delta y$, then:

❸ $= G_1 \oplus (G_0 \oplus E_k(AD_2))) \oplus ((G_0 \Delta E_k(AD_1)) \Delta E_k(AD_2)) =$
$= G_1 \oplus (G_0 \oplus E_k(AD_2))) \oplus (\quad G_1 \oplus G_0 \quad) \Delta E_k(AD_2)) =$ ❹

And assuming also linearity in the delta-carry vector for the third term above, then:

❹ $= G_1 \oplus (G_0 \oplus E_k(AD_2))) \oplus (G_1 \Delta E_k(AD_2)) \oplus (G_0 \Delta E_k(AD_2)) =$ ❺

finally, since the 2nd term equals the 4th one, we have:

❺ $= G_1 \oplus (G_1 \Delta E_k(AD_2)) = G_2$

That is, in order G_2 is an application only depends on AD_1 and AD_2 values but not on their actual location in the chain, the carry-delta operator shall exhibit the transitive property at least 1 time and the linearity property 2 times.

As explained in section A.4 on Δ operator properties, although this operator is neither transitive nor linear in a regular sense, if we define these properties in a "probabilistic" sense, if the operands are random variables then the result can exhibit those properties in a fraction of the cases, or equivalently, with a given probability. Alternatively we can indicate how an specific property is satisfied indicating the information entropy difference between the actual result of the operator and the hypothetic result that would be obtained if the property was satisfied in the 100% of the cases. Table A1 below tabulates some values of the probability a property is satisfied, its $\log_2$ value and the associated entropy difference for several values of block size. For a more general study, please, refer to section A.4.

	Linearity $(x \oplus y)\Delta z = (x \Delta z) \oplus (y \Delta z)$			**Associativity** $(x \Delta y)\Delta z = (x \Delta z)\Delta y$			**2 Lin. & 1 Ass**
b	**P$_=$**	**Log2 P$_=$**	**H** (bits)	**P$_=$**	**Log2 P$_=$**	**H** (bits)	**Log2 P**
1	0,000000000000000	0,000	0,000	1,000000000000000	0,000	0,000	0,000
8	0,496093750000000	-5,203	6,747	0,047771801108618	-4,388	6,535	-14,793
16	0,499984741210937	-11,839	14,656	0,001258865589324	-9,634	14,111	-33,313
32	0,499999999767169	-25,121	30,475	0,000000874567541	-20,125	29,262	-70,366
64	0,500000000000000	-51,683	62,114	0,000000000000422	-41,107	59,564	-144,474
128	0,500000000000000	-104,808	125,390	0,000000000000000	-83,073	120,168	-292,688
256	0,500000000000000	-211,057	251,944	0,000000000000000	-167,003	241,377	-589,118
512	0,500000000000000	-423,557	505,051	0,000000000000000	-334,863	483,794	-1181,976
1024	0,500000000000000	-848,555	1011,26	0,000000000000000	-670,583	968,629	-2367,693

Table A1: Linearity and associativity of Δ operator

In the above table it is appreciated that although the Δ operator is neither transitive nor linear, in a "probabilistically" sense it exhibits those properties with relatively significant probability (from a cryptographic point of view). For instance, if we checks the Δ associativity with three 64 bit random operands "only" 2^{41} times, then the result will show transitive behavior 1 time in average (equivalently one can say that with respect to the associativity property, the Δ operator, applied to 64 bit random vectors, performs as associative plus a random error of 59,56 equivalent bits.

Nevertheless, in order a position change in some *AD* blocks is unnoticed it is necessary that, at least, 2 equivalent operations with the carry-delta operator behave as transitive and another one exhibits linearity. The overall result shown at the right hand column in Table A1: the probability a reordering gets undetected because of "probabilistic" associativity and linearity properties of the carry delta vector is negligible and smaller in all cases than 2^{-b}.

There will remain, of course, some chance that without exhibiting such probabilistic associativity and linearity the impairment introduced by a block reordering gets luckily undetected, but in any case such probability will not be greater than $2^{-(b-1.25)}$, according to the entropy injected by each *AD* block within the tag chaining.

To conclude with this section, it can asserted now that disregarding the computational resources an attacker could devote to build a fake associated data string the probability of not being detected by some alteration in the G_M tag will not be greater than 2-(b-1.25) and, moreover, in the remotely case that the G_M tag is not altered the attacker cannot be aware of such event.

A.3.2 Plaintext Integrity

Disregarding the computational resources a potential attacker could spend to build a fake cryptogram C', the best chance to achieve such purpose would not be better than $2^{-(h-1.25)}$, where h is the number of authentication bits appended to the message (b bits if the native mechanism if a block-wide authentication tag is used with, or without, *ICV* bits stealing padding).

A.3.2.1 Demonstration

Draft Version 0.9: To be completed

Since the first subprocess in the decryption procedure is to pass the cryptogram blocks directly through a battery of deciphering algorithm boxes, it is obvious that any attack based on synthesizing one or several fake blocks from some combination, or alteration, of the authentic ones will produce an unpredictable and random deciphered block. As it will be shown immediately, this error will propagate over the inner vectors till the verification of the authentication tag causing the detection of the attack. Moreover, the impact caused in the last *ICV* block (or *MDC* one depending on which verification method is used) by that error measured in entropy units will be of b bits, that is, no entropy is lost error over the block chaining path of the decryption procedure as it can be seen in section A.3.2.1.1.

This case can be shown equivalent to the injection of any fake Ci value selected without any particular criteria.

Now, analyse how a block removal, repetition or reordering causes an unrecoverable error (as per AD analysis: they should be reduced to a necessary condition on the autocancelling when two contiguous blocks are reordered.

A.3.2.1.1 Error Propagation Analysis

Demo concept script for DraftV0.9:

- From (A6) Oi = () + Oi-1 ---> once an error enters with its *b*-bits entropy in the inner sequences, that error is maintained by linear and direct chaining through Os vectors.
- Other paths include couples of xor and + sums that reduce the propagated entropy (to be developed) by the tail / lsb of the error.
- At the end, IN / ON maintain b-bits of entropy caused by any erroneous Ti block injected at any location (or by each ionjected Ti value & its specific position in the chain) and
- Moreover, a modification of the associated data causes an error of (b-1.25) bits in IV'a. That error is also propagated over the entropy-conservative chaining path
- ++ specific analyisis of bit stealing from ICV for padding showing it is the same to verify the ICV decrypted in native mode than to check a part in RN and the rest in a MDC" computed by the decryption process
- ++ specific analysis for the optional reduction on authentication bits

A.4 Carry Delta Operator ($x\Delta y=(x+y)\oplus(x\oplus y)$) Characterization

Let x and y be two independent and random b-bit binary vectors such as at least one of them is a totally random (i.e. each value it takes follows a uniform distribution from the total set of b-bit possible values). It is a well known fact that either x⊕y and $x+y$ maintain the entropy / randomness of x and y. But which is the differential entropy between x⊕y and $x+y$? That is, if we define $x\Delta y=(x+y)\oplus(x\oplus y)$, then how much entropy does contain the result and which are the algebraic properties of this operator?

Moreover, it can be observed that the only difference between the bits in the i-th positions of $(x\oplus y)$ and $(x+y)$ comes from whether a carry from the bit addition in the (i-1)-th position has to be applied for the sum of the i-th position. Thus, it is immediate that the i-th bit of Δ will be 1 if such carry bit occurred and 0 otherwise. That is, $x\Delta y$ is just a b-bit vector that contains all the bit carries that appear during the computation of $(x+y)$.

In this section the relevant properties of the Δ operator, are studied and characterized:

- Section A.4.1 - Randomness propagation from the operands. It is shown that
- Section A.4.2 - Relevant Δ algebraic properties:
  - Section A.4.2.1 – Commutativity $x\Delta y= y\Delta x$. The Δ operator results to be commutative.
  - Section A.4.2.2 – Associativity: $x\Delta(y\Delta z)$ $(x\Delta y)\Delta z$. It is shown that although not associative in a classical sense, the Δ operator exhibits a residual associative behavior and that if the operands are random binary vectors, then there's a (very) small probability Δ behaves as associative.
  - Section A.4.2.3 - Linearity: $x\Delta(y\oplus z) \overset{?}{=} (x\Delta y)\oplus(x\Delta z)$: As per the associativity property, it is also shown that although not linear the in a classical sense with respect xor addition, the Δ operator exhibits a residual linearity behavior and that if the operands are random binary vectors, then there is still a (very) small probability Δ behaves as linear. NOTE: we could call this property also 'distributiveness' of Δ over ⊕ but we prefer 'linearity' as a more meaningful name for our purposes.

To sum up, the algebraic properties exhibited by the carry delta operator make it worth to be defined as a strange but interesting operator for application in data hashing-like processes. Its lack of linearity and its acceptable randomness propagation properties make it interesting, for instance, to generate hash tags dependent both on the values of the data blocks and their specific positions within the sequence. On more generic terms: an adequate combination of xor and modular additions can be an interesting building block for those data processing procedures where chaotic behavior is pursued and where its input data already exhibits high entropy.

A.4.1 Information Entropy in the Carry-Delta Vectors (or Δ_i Randomness)

Let's see which is the probability for any of the bit positions $[\Delta]_i$ of being 1. That is, which is the probability, $P_i = P\{[\Delta]_i = 1\}$, of having a bit carry from previous position at the addition modulo-2^n of *A* and *B*:

- $P_1 = 0$, since being the first bit summed, no bit carry has to be applied from previous one;
- $P_2 = 1/4$, since only if the first two summed bits are simultaneously 1, the bit carry is also 1;
- $P_3 = \frac{1}{4}(1-P_2) + \frac{3}{4}P_2$, since if no carry was applied in the previous position then there would be a probability of ¼ that the bit sum in that position produces a carry and otherwise such probability would be ¾.
- and $P_i = \frac{1}{4}(1-P_{i-1}) + \frac{3}{4}P_{i-1} = \frac{1}{4} + \frac{P_{i-1}}{2}$ for the general case.

Since P_i is a monotonously increasing sequence and, as a probability, it is bounded by 1 then it will converge for $i \to \infty$ towards a specific P value that will be given by:

$$P = \frac{1}{4} + \frac{P}{2} \quad \Rightarrow \quad P = \frac{1}{2}.$$

Figure 1 below illustrates that the convergence of P_i towards the ½ is actually extremely fast. For instance, for the 10th bit, P_{10} approximates ½ just with an error of 10^{-3}. On other words, we can conclude that, except for a very few of the less significant bits, the carry-delta vectors exhibit very good entropy/randomness as the inner vectors do for all of their bits. Thus, let's quantify the total entropy contained in a Δ vector, or on other terms, the equivalent length of Δ in terms of random bits.

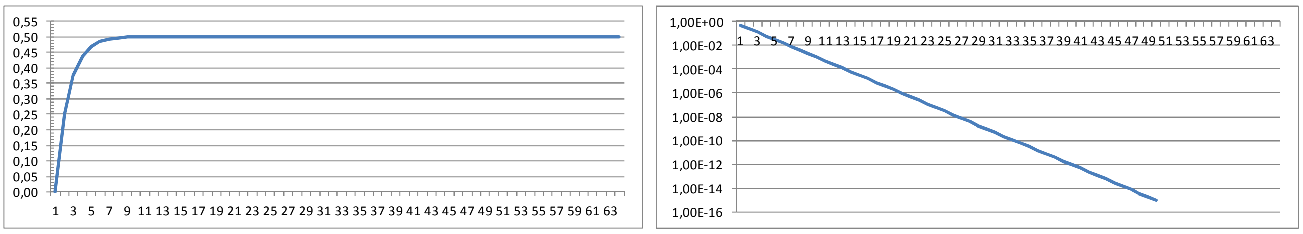

Figure A1¡Error! Marcador no definido.**: (a) P_i vs *i*; (b) log (1/2- P_i) vs *i***

According to Shannon definition of Information Entropy, each bit of Δ_i exhibits an entropy defined by:

$$H([\Delta]_i) = -P_i \cdot \log_2 P_i - (1-P_i)\cdot \log_2(1-P_i).$$

From where the total entropy for the whole carry-delta vector is:

$$H(\Delta) = \sum_{i=1}^{n} H([\Delta]_i) = \sum_{i=1}^{n}\left(-P_i \cdot \log_2 P_i - (1-P_i)\cdot \log_2(1-P_i)\right). \qquad (A10)$$

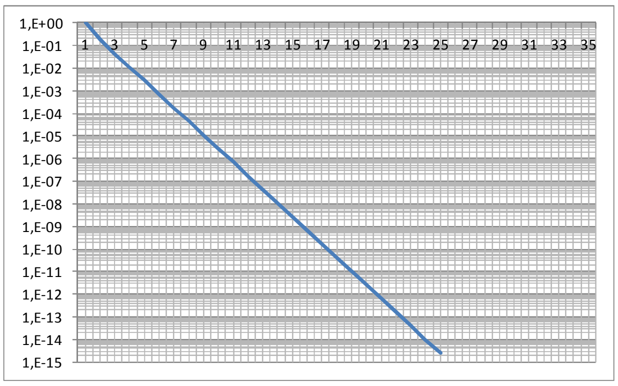

Figure A2¡Error! Marcador no definido.**: log(1 - H([Δ]$_i$)) vs *i***

Figure A2 above shows that the entropy per bit is quickly maximized since it converges very rapidly towards 1 bit of information entropy per each 'physical' bit. For instance, for the 6-th bit of Δ its randomness entropy is above $1\text{-}10^{-3}$. On the other hand, Table A2 below indicates according to (A10) the total entropy, H(Δ), for different values of *n*, that for $n \geq 3$ the total entropy is above (n-5/4). That is, almost equal to the block size for any practical *n* size (64, 128, 256, 512, 1024, 2048 ...) since only 1,25 equivalent bits do not exhibit any entropy.

n **(physical bits)**	**1**	**2**	**3**	**4**	**8**	**16**	**32**	**64**	**128**	**256**
H(Δ) (random bits)	0	0,811278	1,765712	2,75441	6,750667	14,75065	30,75065	62,7506	126,7506	254,7506

Table A2: Total entropy of the carry-delta vector, Δ, for different block sizes

We can conclude that, as for + and ⊕ operators, the carry-delta Δ operator when applied to two random variables also produces a different random vector that almost preserves the entropy of the operands. The total entropy loss is equivalent to slightly less than 1.25 bits: one bit is lost in the lsb, since it is always 0, and the rest is lost in the subsequent lsb positions while their associated '1' probability converges, quickly, to ½. Although in principle any entropy loss is undesirable in a cryptographic operator, the other properties of this operator make it worth this small cost.

A.4.2 Algebraic properties of the carry-delta operator

A.4.2.1 Commutativity: $x\Delta y = y\Delta x$

Given the definition of the carry-delta operator as $x\,\Delta\,y = (x + y) \oplus (x \oplus y)$, it is immediate that it is a commutative operator.

A.4.2.2 Associativity: $x\Delta(y\Delta z) \overset{?}{=} (x\Delta z)\Delta y$

Before starting with this associativity analysis, it shall be remarked that if the carry-delta vector was associative (in the regular sense) then, since it is also commutative, we would have:

$x\Delta(y\Delta z) = x\Delta(z\Delta y) = (x\Delta z)\Delta y = (x\Delta y)\Delta z.$

But, as we will see, Δ is not transitive in such "regular" sense. Anyway, we are to see that there is some residual behavior of this property if we use "associativity" concept just in a "probabilistic" sense (i.e. for some values of the operands the property is complied and for others, not). In such situation one could define such "probabilistic" transitivity according to one of the following conditions:
a) $x\Delta(y\Delta z) = (x\Delta z)\Delta y$;
b) $x\Delta(y\Delta z) = (x\Delta y)\Delta z$;
c) $x\Delta(y\Delta z) = (x\Delta z)\Delta y = (x\Delta y)\Delta z$;

We will focus just in the case (a) above. Nonetheless, observe that although ($(x\Delta z)\Delta y$) and ($(x\Delta y)\Delta z$) will be (not totally) independent random variables their study and behavior are totally equivalent. Regarding the third case, it corresponds to a more stringent concept for the "probabilistic" associativity definition that is not of interest here.

The table below shows the probabilistic truth tables for the a generic *i*-th bit position of both ($x\Delta(y\Delta z)$) and ($(x\Delta z)\Delta y$) values. The probabilistic definition comes from the fact that complementary to the bit values exhibited in the three leftmost columns of the table, to compute the values on the right one has to take into account the probability of having a carry bit propagated from the computation of the immediately lower bit position. This probability has been already characterized in section A.4.1. Moreover, in order to simplify the calculations it has been assumed that the carry probability distribution when one of the operands is another carry-delta vector instead of a pure random vector like *x*, *y* or *z*. This assumption is taken on the basis that the P_i distribution for a carry-delta vector converges quickly towards the ½ value. Finally, the rightmost column indicates the probability that both vectors match either on the value 1 or the value 0

x	y	z	$y\Delta z$	$x\Delta z$	$x\Delta(y\Delta z)$	$(x\Delta z)\Delta y$	$P(\ x\Delta(y\Delta z) = (x\Delta z)\Delta y)$[4]
0	0	0	0	0	0	0	1 (=0)
0	0	1	P_i	P_i	P_i^2	P_i^2	$P_i^4 + (1- P_i^2)(1- P_i^2)$
0	1	0	P_i	0	P_i^2	P_i	$P_i^3 + (1- P_i^2)(1- P_i)$
0	1	1	1	P_i	P_i	$(2-P_i)\cdot P_i$	$(2-P_i)\cdot P_i^2 + (1-P_i)(1-(2-P_i)\cdot P_i)$
1	0	0	0	P_i	P_i	P_i^2	$P_i^3 + (1- P_i)(1- P_i^2)$
1	0	1	P_i	1	$(2-P_i)\cdot P_i$	P_i	$(2-P_i)\cdot P_i^2 + (1-P_i)(1-(2-P_i)\cdot P_i)$
1	1	0	P_i	P_i	$(2-P_i)\cdot P_i$	$(2-P_i)\cdot P_i$	$(2-P_i)^2\cdot P_i^2 + (1-(2-P_i)\cdot P_i)^2$
1	1	1	1	1	1	1	1 (=1)

Table A3: Probabilistic truth table for Δ associativity

From the table above it becomes evident that although quite similar, depending the carries propagation from the lower bit positions the two vectors will be different.

Now, in order to calculate the match probability for $(x\Delta(y\Delta z))$ and $((x\Delta z)\Delta y)$ values, we define $P^{=}_i$ as the aggregated probability of having a match in the the *i*-th bits for whatever values are taken by *x*, *y* and *z* in that position, that from the entries in the rightmost column in the table and a little patient lead to:

$P^{=}_i = (8 - 14\cdot Pi + 23\cdot Pi^2 - 20\cdot Pi^3 + 16\cdot Pi^4 - 6\cdot Pi^5 + Pi^6) / 8;$; where $i = 1, \ldots, b$.

As per the carry-delta entropy, the above match probability for bit position starts at value 1 for the rightmost bit (the lsb of a Δ is always 0) and quickly converges to the value 0,6347 (or towards an equivalent entropy per bit of 0.947 bits). On the other hand, if the working block size is *b* bits, the probability that all of them match can be calculated as the product of the above formula for i= 1, …, *b*:

$$P_{=} = \prod_{i=1}^{b} P_i^{=} = \frac{1}{8}\prod_{i=1}^{b}\left(8 - 14\cdot P_i + 23\cdot P_i^2 - 20\cdot P_i^3 + 16\cdot P_i^4 - 6\cdot P_i^5 + P_i^6\right);$$

Moreover, we define the differential entropy between $(x\Delta(y\Delta z))$ and $((x\Delta z)\Delta y)$ values as the Shannon entropy applied to the bit match probability accumulated for all the *b* bits (intuitively, it indicates the number of different bits on the two vectors):

$$\mathrm{H}(\Delta) = \sum_{i=1}^{n} \mathrm{H}(P_i^{=}) = \sum_{i=1}^{n}\left(-P_i^{=}\cdot\log_2 P_i^{=} - (1-P_i^{=})\cdot\log_2(1-P_i^{=})\right).$$

[4] That is, the probability that both of them are 1 plus the probability that both of them are 0.

The following table illustrates some values of the total match probability (both in lineal an log2 scales) and differential entropy for different blocksizes.

	Associativity $(x\Delta y)\Delta z \overset{?}{=} (x\Delta z)\Delta y$		
b	**P$_=$**	**Log2 P$_=$**	**H** (bits)
1	1,000000000000000	0,000	0,000
8	0,047771801108618	-4,388	6,535
16	0,001258865589324	-9,634	14,111
32	0,000000874567541	-20,125	29,262
64	0,000000000000422	-41,107	59,564
128	0,000000000000000	-83,073	120,168
256	0,000000000000000	-167,003	241,377
512	0,000000000000000	-334,863	483,794
1024	0,000000000000000	-670,583	968,629

Table A4: Some figures for the Δ probabilistic associativity

A.4.2.3 Linearity: $x\Delta(y\oplus z) \overset{?}{=} (x\Delta y)\oplus(x\Delta z)$

Analogously to the associativity analysis, we are now to characterize the probabilistic linearity exhibited by the carry-delta vector in relation with the xor addition.

The table below shows the probabilistic truth tables for the a generic *i*-th bit position of both ($x\Delta(y\oplus z)$) and ($(x\Delta y)\oplus(x\Delta z)$) values. The calculation of this new truth table is very similar to the one performed for the associativity property. Nonetheless, in this case it shall be remarked that, since there is no composition of Δ operators, the calculated probabilities are absolutely accurate since no simplification assumption is needed. Again, the probabilistic concept comes from the fact that complementary to the bit values exhibited in the three leftmost columns of the table, to compute the values of some Δ on the right, one has to take into account the probability of having a carry bit propagated from the immediately lower bit position (see section A.4.1). Finally, the rightmost column indicates the probability that both vectors match either on the value 1 or the value 0

x	y	z	$y\oplus z$	$x\Delta y$	$x\Delta z$	$x\Delta(y\oplus z)$	$(x\Delta y)\oplus(x\Delta z)$	P($x\Delta(y\Delta z) = (x\Delta z)\Delta y$)[5]
0	0	0	0	0	0	0	0	1
0	0	1	1	0	P_i	P_i	P_i	$P_i^2+(1-P_i)^2$
0	1	0	1	P_i	0	P_i	P_i	$P_i^2+(1-P_i)^2$
0	1	1	0	P_i	P_i	0	$2\cdot P_i\cdot(1-P_i)$	$1-2\cdot P_i\cdot(1-P_i)$
1	0	0	0	P_i	P_i	P_i	$2\cdot P_i\cdot(1-P_i)$	$P_i\cdot(2\cdot P_i\cdot(1-P_i))+(1-P_i)\cdot(1-2\cdot P_i\cdot(1-P_i))$
1	0	1	1	P_i	1	1	$1-P_i$	$1-P_i$
1	1	0	1	1	P_i	1	$1-P_i$	$1-P_i$
1	1	1	0	1	1	P_i	0	$1-P_i$

Table A5: Probabilistic truth table for Δ linearity

From the table above it becomes evident that although quite similar, depending the carries propagation from the lower bit positions the two vectors will be different.

[5] That is, the probability that both of them are 1 plus the probability that both of them are 0.

Now, in order to calculate the match probability for ($x\Delta(y\oplus z)$) and ($(x\Delta y)\oplus(x\Delta z)$) values, we define $P^{=}_{i}$ as the aggregated probability of having a match in the the i-th bits for whatever values are taken by x, y and z in that position, that from the entries in the rightmost column in the table and a little patient lead to:

$P^{=}_{i} = (2 - 3\cdot Pi + 3\cdot Pi^{2} - Pi^{3}) / 2$; where $i = 1, \ldots, b$.

On the other hand, if the working block size is b bits, the probability that all of them match can be calculated as the product of the above formula for $i = 1, \ldots, b$.

Observe that $P^{=}_{i} = 1$, for the rightmost bit (i=0), and for $i \gg 1$, P_i converges quickly towards 1/2 and consequently:

$P^{=}_{i} \rightarrow 9/16 = 0{,}5625$.

Analogously to the previous analysis so far done, one can define the differential entropy per bit and total entropy for the entire working block size. In the case of differential entropy per bit, it starts at 0 bit for the rightmost position and converges quickly towards 0,989 bits (it establishes at that value from the 10th bit on.

The following table illustrates some values of the total match probability (both in lineal an log2 scales) and differential entropy for different blocksizes.

	Linearity $x\Delta(y\oplus z) \overset{?}{=} (x\Delta y)\oplus(x\Delta z)$		
b	**$P_=$**	**Log2 $P_=$**	**H** (bits)
1	1,000000000000000	0,000	0,000
8	0,027155634503633	-5,203	6,747
16	0,000272878413245	-11,839	14,656
32	0,000000027411577	-25,121	30,475
64	0,000000000000000	-51,683	62,114
128	0,000000000000000	-104,808	125,390
256	0,000000000000000	-211,057	251,944
512	0,000000000000000	-423,557	505,051
1024	0,000000000000000	-848,555	1011,265

Table A6: Some figures for the Δ probabilistic Linearity