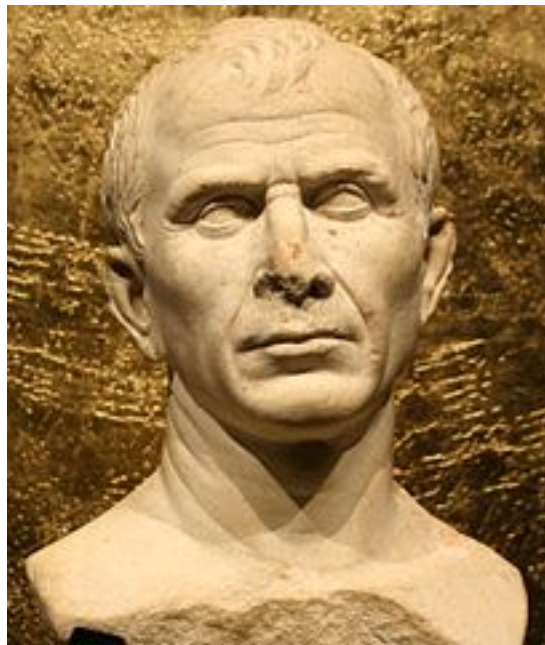


# Julius: Secure Mode of Operation for Authenticated Encryption Based on ECB and Finite Field Multiplications

Lear Bahack\*

Submission to the CAESAR competition, version 1.0, March 2014



Gaius Julius Caesar, 100 BC – 44 BC. Source: Mclelat, GNU, Creative Commons via Wikimedia Commons.

---

\*Weizmann Institute of Science, Rehovot, Israel. E-mail: [lear.bahack@gmail.com](mailto:lear.bahack@gmail.com)

## Abstract

We present two new blockcipher modes of operation for authenticated encryption with associated data, designed to achieve the maximal possible security in case of misused IV, while being efficient as the Galois/Counter Mode (GCM). Both of the modes are provably secure up to the birthday bound, are suitable for both software and hardware, and are based on  $GF(2^{128})$  multiplications by a secret element of the field.

The Julius-CTR mode can be viewed as a certain variation combining the GCM, SIV and Unbalanced Feistel Network, while the Julius-ECB mode can be viewed as a certain variation of the Naor-Reingold mode. We specify two versions for each mode: a regular version and a compact version, having different ciphertexts redundancies. Several variants aimed to achieve increased security, parallelization, and efficiency are briefly explored.

Based on the two Julius modes of operation and the AES-128 blockcipher, we propose a family of four specific algorithms for authenticated encryption with associated data to the CAESAR competition.

## 1 Introduction

Symmetric key authenticated encryption (AE) is in a sense the most basic and fundamental usage of cryptography. Although today's cryptography is far broader and contains complicated algorithms aiming to achieve other (more complicated) goals, the vast majority of applications use "complicated" cryptographic algorithms only in addition to a "basic" symmetric key AE algorithm.

Years of research and development made modern symmetric cryptography converged to the following heuristic: symmetric key algorithm / cryptosystem / protocol should be composed of a well-studied subalgorithm called a "cryptographic primitive", and of a certain manner / scheme in which the primitive is to be used. Common cryptographic primitives are Block Ciphers; Pseudo Random Number Generators (PRNG); and compression hash functions. They all achieve different basic cryptographic goals. Apart from enabling simple and robust designs of cryptosystems, a major benefit of the mentioned heuristic is the possibility to assure the cryptographic security of the complete algorithm by separately analyzing the security of the primitive and of the scheme.

During the last two decades, symmetric primitives have been designed and thoroughly researched by great cryptographers. We point the AES competition for block-cipher primitive; the *eSTREAM* competition for PRNG

primitives; and the *SHA-3* competition for hash function as significant events which have led to the formation of trustable primitives.

On the other hand cryptographic schemes, such as mode of operation for blockcipher encryption and authentication, have received significantly less attention and were often regarded as "trivial" or "not interesting". Perhaps the reason for the reduced attention is the combination of fairly secure modes on the one hand and on the other hand the fact that a weakness of an encryption scheme is less fatal than a weakness of a block cipher primitive, which might enable a key recovery attack.

A known proverb says that a chain is only as secure as its weakest link. In the case of blockcipher based AE algorithms, today's leading blockcipher algorithms such as the famous AES are definitely not the weakest link. The recent revelations of Edward Snowden indicate that the NSA, considered by many to be the most powerful intelligence agency in the world, has not been able to break any of the standard symmetric primitives, the most important of which being the AES.

We believe that the weakest link of AE algorithms is the end user, which in reality may use the cryptographic scheme wrongly. The second weakest link, which is less severe in AE algorithms than in encryption only algorithms, is the mode being vulnerable to chosen ciphertext attacks (CCA). Since the underlying block ciphers of AE algorithms are secure, the efforts of improving AE algorithms should be given to the design and research of new modes that are efficient and yet preserve the maximal possible security in case of a wrong usage. Being CCA secure is a desirable (but not crucial) property too.

We hope that the CAESAR competition will drive attention to the field of AE algorithms and modes of operation, and that five years from now we shall have trusted secure and efficient modes of operation that are better than the widely used Galois/Counter Mode (GCM) [12].

Our contribution is two new efficient and parallelizable modes of operation called Julius-ECB and Julius-CTR that achieve maximal security in the sense of indistinguishability between the real AE algorithm and an ideal AE algorithm, where the distinguisher is allowed to adaptively choose the message, including the IV. Thus, our suggested AE algorithms preserve security in case of a nonce misuse, which we consider to be the weakest link in the chain.

## 2 Specification

We specify first the two blockcipher modes Julius-ECB and Julius-CTR, each having a regular and a compact version. The specification is based on a pseudo random permutation over 16 byte strings primitive, which is not part of the modes and thus is not specified. Next we specify a set of 8 concrete and parameterized algorithms for authenticated encryption with associated data. The pseudo random permutation primitive used in all the proposed algorithms is the standard AES-128 blockcipher, whose specification is given in [6].

We start by defining the notations in used, and highlighting four simple mathematical facts and properties that are essential for this document.

### Notations

1. Bytes are integers in  $\{0, 1, \dots, 255\}$ , and are used as our atomic data unit. Messages, plaintexts, associated data, ciphertexts, initialization vectors (IV), and binary representations of certain lengths are all regarded as strings of bytes. We denote the string of  $k$  consecutive zero bytes by  $ZEROES(k)$ .
2. The concatenation  $str1 || str2$  of two (byte) strings is defined as the string whose length is  $len1 + len2$ , where  $len1$  and  $len2$  are the lengths of  $str1$  and  $str2$  correspondingly, such that its first  $len1$  bytes are the same as of  $str1$  and its last  $len2$  bytes are the same as of  $str2$ . The concatenation operation is associative and thus we don't use parentheses in case of a multiple concatenation.
3. The XOR  $A \oplus B$  of the two bytes  $A = \sum_{i=0}^7 2^i a_i$  and  $B = \sum_{i=0}^7 2^i b_i$  where  $a_0, \dots, a_7, b_0, \dots, b_7 \in \{0, 1\}$ , is the byte  $A \oplus B = \sum_{i=0}^7 2^i (a_i \oplus b_i)$ , where we define  $0 \oplus 0 = 1 \oplus 1 = 0$  and  $0 \oplus 1 = 1 \oplus 0 = 1$ . The XOR  $str1 \oplus str2$  of two byte strings  $str1, str2$  of the same length  $len$  is defined as the string:  $str1[0] \oplus str2[0] || \dots || str1[len - 1] \oplus str2[len - 1]$ .
4. A block is defined as a string of exactly 16 bytes. We denote the two blocks:  $0 || 0 || \dots || 0$  and  $0 || 0 || \dots || 0 || 1$  by  $ZERO\_BLK$  and  $ONE\_BLK$  correspondingly.  $E(blk)$  and  $E^{-1}(blk)$  stands for the blockcipher encryption and decryption of the block  $blk$ .
5. For a block  $blk = b[0] || b[1] || \dots || b[15]$ , and an integer  $len \in \{0, 1, \dots, 16\}$  we denote the string  $b[0] || \dots || b[len - 1]$  of the first/leftmost bytes of  $blk$

by  $left(blk, len)$  and the string  $b[16-i] \parallel \dots \parallel b[15]$  of the last/rightmost bytes of  $blk$  by  $right(blk, len)$ .

6. We represent  $GF(2^{128})$ , the finite field with  $2^{128}$  elements, by the set of all polynomials over  $GF(2)$  of degree 127 or less, in the following way: the field addition of two such polynomials is the same as their sum over  $GF(2)$  and their field multiplication is obtained by their  $Z_2[x]$  multiplication modulus the irreducible polynomial  $x^{128} + x^7 + x^2 + x + 1$ .
7. We regard blocks as elements of  $GF(2^{128})$  in the following way: the corresponding element of the block  $blk = b[0] \parallel \dots \parallel b[15]$  is

$$\sum_{j=0}^{15} \left( x^{8j} \cdot \sum_{i=0}^7 x^i \cdot b_{15-j,i} \right)$$

where for all  $j \in \{0, \dots, 15\}$  we have  $blk[j] = \sum_{i=0}^7 2^i b_{j,i}$  and  $b_{j,0}, \dots, b_{j,7} \in \{0, 1\}$ .

8. We regard a byte string of length which is an integer multiplication of 16 as a polynomial over  $GF(2^{128})$  in the following way: let  $\beta_0, \dots, \beta_{len-1}$  be the  $GF(2^{128})$  field elements corresponding to the blocks  $blk[0], \dots, blk[16 \cdot len - 1]$ , then the polynomial corresponding to the string  $blk[0] \parallel \dots \parallel blk[16 \cdot len - 1]$  is  $\sum_{d=0}^{16 \cdot len - 1} x^d \cdot \beta_{16 \cdot len - 1 - d}$ . Moreover, we define the evaluation of this string over the field element  $\alpha$  as the corresponding block of  $\sum_{d=0}^{16 \cdot len - 1} \alpha^d \cdot \beta_{16 \cdot len - 1 - d}$ .
9. The inputs for the two Julius modes are:
  - IV string of length  $IV\_LEN$ :  $IV = IV[0] \parallel \dots \parallel IV[IV\_LEN]$ . The non-negative integer  $IV\_LEN$  is a (constant) parameter of the mode.
  - Length of the plaintext, which is an integer between 0 and  $2^{64} - 1$ , inclusive.<sup>1</sup> The given length  $plen$  is represented by the 8 bytes string  $plen[0] \parallel \dots \parallel plen[7]$  so that  $plen = \sum_{i=0}^7 2^{8i} plen[7 - i]$ .
  - Length of the associated data, which is an integer between 0 and  $2^{64} - 1$ , inclusive. The given length  $adlen$  is represented by the 8 bytes string  $adlen[0] \parallel \dots \parallel adlen[7]$  so that  $adlen = \sum_{i=0}^7 2^{8i} adlen[7 - i]$ .
  - Plaintext byte string:  $plain = plain[0] \parallel \dots \parallel plain[plen-1]$ .

---

<sup>1</sup> $2^{64} - 1 = 18,446,744,073,709,551,615$

- Associated data byte string:  $ad = ad[0] \parallel \dots \parallel ad[adlen-1]$ .
- A pseudo random permutation  $E$  over 16 bytes strings.

## Mathematical properties and facts

1. The finite field of cardinality  $p^n$ , for any prime number  $p$  and positive integer  $n$ , is unique. We shall use this fact in the two Julius variants that have different representations of  $GF(2^{128})$  described in section 5.
2. For any  $\alpha, \mu \in GF(2^{128})$  and non negative integer  $k$ , the evaluations of the strings  $\mu \parallel \mu \cdot \alpha \parallel \mu \cdot \alpha^2 \parallel \dots \parallel \mu \cdot \alpha^{2^k-1}$  and  $\mu \parallel \mu \cdot (\alpha + 1) \parallel \mu \cdot \alpha \parallel \mu \cdot \alpha^2 \parallel \dots \parallel \mu \cdot \alpha^{2^k}$  over  $\alpha$  are both zero.
3. The evaluation of the string  $str1 \oplus str2$  over a certain element is the same as the field addition of the evaluations of  $str1$  and  $str2$  over the same certain element.
4. Replacing the last block  $blk[len-1]$  in the blocks concatenation

$$blk[0] \parallel \dots \parallel blk[len - 1]$$

by the evaluation of the concatenation string over a certain field element, is an involution.

## 2.1 The Julius Modes

### 2.1.1 Julius-ECB Regular Version

#### Padding

Let  $pres$  and  $adres$  be the smallest non-negative integers so that  $plen + pres$  and  $adlen + adres + IV\_LEN$  are both multiplications of 16. We pad the message as follows:

$$\begin{aligned} padded\ message &= ONE\_BLK \parallel IV \parallel adlen \parallel plen \parallel associsteddata \parallel \\ &\parallel ZEROES(adres + pres + 16) \parallel plain \end{aligned}$$

The substring of the last/rightmost  $pres + 16 + plen$  bytes is denoted by  $reg = blk[0] \parallel \dots \parallel blk[blen-1]$  where  $blen = 1 + \lceil \frac{plen}{16} \rceil$  and  $blk[0], \dots, blk[blen - 1]$  are blocks.

## The Julius Involution

We derive a secret element of  $GF(2^{128})$  from the pseudo random permutation  $E$ , by taking the corresponding element of  $E(ZERO\_BLK)$ . We denote this element by  $\delta$  and call it the *derived key*. Using the (rest of the) padded message and the derived key we define an involution over the string  $reg$ .

Let  $seed$  be the evaluation of the padded message over the field element  $\delta$ . We compute  $\mu = E(seed)$  and use it to form a  $pres + 16 + plen$  bytes mask as follows: if  $blen$  is even, the mask is:

$$\mu \parallel \mu \cdot \delta \parallel \mu \cdot \delta^2 \parallel \dots \parallel \mu \cdot \delta^{blen}$$

else, the mask is

$$\mu \parallel \mu \cdot (\delta + 1) \parallel \mu \cdot \delta \parallel \mu \cdot \delta^2 \parallel \dots \parallel \mu \cdot \delta^{blen}$$

The mask is then XORed into the string  $reg$ .

## ECB

The output (i.e. ciphertext) is the ECB encryption of the string  $reg$ : let  $reg = blk[0] \parallel \dots \parallel blk[blen-1]$  (note that the blocks  $blk[0], \dots, blk[blen-1]$  have been updated by the Julius Involution). The ciphertext string is then:

$$E(blk[0]) \parallel \dots \parallel E(blk[blen-1])$$

### 2.1.2 Julius-ECB Compact Version

#### Padding

Let  $res$  be the smallest non-negative integer so that  $res + 8 + IV\_LEN + adlen + plen$  is a multiplication of 16. We pad the message as follows:

$$\begin{aligned} padded\ message &= ONE\_BLK \parallel IV \parallel adlen \parallel plen \parallel associsteddata \parallel \\ &\parallel ZEROES(res + 8) \parallel plain \end{aligned}$$

The substring of the last/rightmost  $8 + plen$  bytes is denoted by

$$reg = b[0] \parallel \dots \parallel b[j-1] \parallel blk[0] \parallel \dots \parallel blk[blen-1]$$

where  $blen = \lfloor \frac{8+plen}{16} \rfloor$ ,  $j = plen + 8 - 16 \cdot blen$ ,  $b[0], \dots, b[j-1]$  are bytes and  $blk[0], \dots, blk[blen-1]$  are blocks.

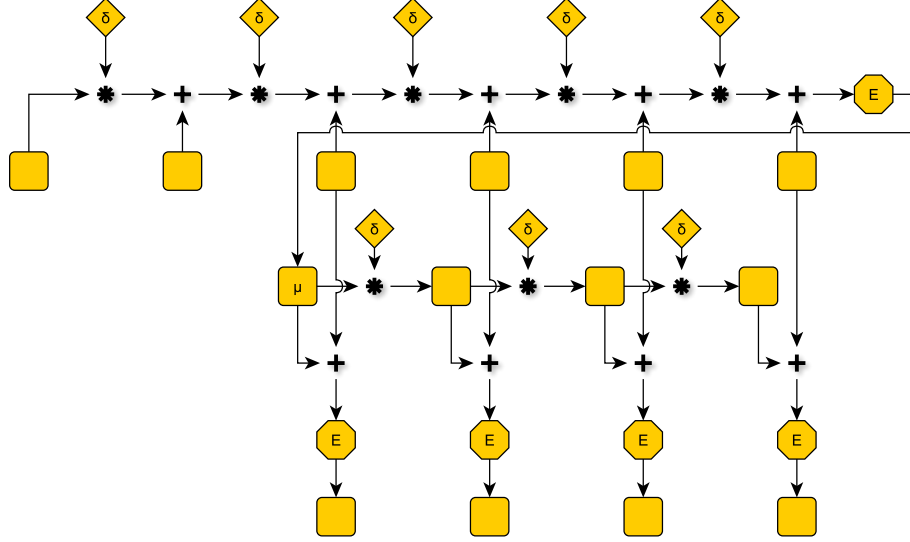


Figure 2.1: Partial diagram of Julius-ECB regular version.  $blen$  is even.

### The Julius Involution

We derive a secret element of  $GF(2^{128})$  from the pseudo random permutation  $E$ , by taking the corresponding element of  $E(ZERO\_BLK)$ . We denote this element by  $\delta$  and call it the *derived key*. Using the (rest of the) padded message and the derived key we define an involution over the string  $reg$ .

Let  $seed$  be the evaluation of the padded message over the field element  $\delta$ . We compute  $\mu = E(E(seed))$  and

$$\omega = ZEROES(16 - j) \parallel right(E(E(seed) \oplus ONE\_BLK), j)$$

and use them to form a  $8 + plen$  bytes mask as follows: if  $blen$  is even, the mask is:

$$right(\omega, j) \parallel \mu + \omega \cdot \delta \parallel \mu \cdot \delta \parallel \mu \cdot \delta^2 \parallel \dots \parallel \mu \cdot \delta^{blen}$$

else, the mask is

$$right(\omega, j) \parallel \mu + \omega \cdot \delta \parallel \mu \cdot (\delta + 1) \parallel \mu \cdot \delta \parallel \mu \cdot \delta^2 \parallel \dots \parallel \mu \cdot \delta^{blen}$$

The mask is then XORed into the string  $reg$ .



## Modified ECB

The output (i.e. ciphertext) is the following modified ECB encryption of the string  $reg$ : let  $reg = blk[0] \parallel \dots \parallel blk[blen-1]$  (note that the blocks  $blk[0], \dots, blk[blen-1]$  have been updated by the Julius Involution),  $A = b[0] \parallel \dots \parallel b[j-1] \parallel left(blk[0], 16-j)$ , and  $B = right(E(A), 16-j) \parallel right(blk[0], j)$ .

The ciphertext string is then:

$$left(E(A), j) \parallel E(B) \parallel E(blk[1]) \parallel \dots \parallel E(blk[blen-1])$$

### 2.1.3 Julius-CTR Regular Version

#### Padding

Let  $pres$  and  $adres$  be the smallest non-negative integers so that  $plen + pres$  and  $adlen + adres + IV\_LEN$  are both multiplications of 16. We pad the message as follows:

$$\begin{aligned} padded\ message &= ONE\_BLK \parallel IV \parallel adlen \parallel plen \parallel associsteddata \parallel \\ &\parallel ZEROES(adres) \parallel plain \parallel ZEROES(pres + 16) \end{aligned}$$

The substring of the last/rightmost  $pres + 16 + plen$  bytes is denoted by  $reg = blk[0] \parallel \dots \parallel blk[blen-1]$  where  $blen = 1 + \lceil \frac{plen}{16} \rceil$  and  $blk[0], \dots, blk[blen-1]$  are blocks.

#### CTR

We derive a secret element of  $GF(2^{128})$  from the pseudo random permutation  $E$ , by taking the corresponding element of  $E(ZERO\_BLK)$ . We denote this element by  $\delta$  and call it the *derived key*. Let  $seed$  be the evaluation of the padded message over the field element  $\delta$ . We replace the last block  $blk[blen-1]$  by  $\mu = E(seed)$  and use this value to generate a strong pseudo random stream that will be then XORed into the rest of  $reg$ .

For each  $i \in \{0, \dots, blen-2\}$  we denote by  $brep(i)$  the block-width binary representation of  $i$ , meaning  $i = \sum_{j=0}^{15} 2^{8j} b[15-j]$  for  $brep(i) = b[0] \parallel \dots \parallel b[15]$ .

The output (ciphertext) is:

$$blk[0] \oplus E(\mu \oplus brep(0)) \parallel blk[1] \oplus E(\mu \oplus brep(1)) \parallel \dots \parallel blk[blen-2] \oplus E(\mu \oplus brep(blens-2)) \parallel \mu$$

### 2.1.4 Julius-CTR Compact Version

#### Padding

Let  $res$  be the smallest non-negative integer so that  $res + 8 + IV\_LEN + adlen + plen$  is a multiplication of 16. We pad the message as follows:

$$padded\ message = ONE\_BLK || IV || adlen || plen || associsteddata || \\ || ZEROES(res) || plain || ZEROES(8)$$

The substring of the last/rightmost  $8 + plen$  bytes is denoted by

$$reg = b[0] || \dots || b[j - 1] || blk[0] || \dots || blk[blen - 1]$$

where  $blen = \lfloor \frac{8+plen}{16} \rfloor$ ,  $j = plen + 8 - 16 \cdot blen$ ,  $b[0], \dots, b[j - 1]$  are bytes and  $blk[0], \dots, blk[blen - 1]$  are blocks.

#### Modified CTR

We derive a secret element of  $GF(2^{128})$  from the pseudo random permutation  $E$ , by taking the corresponding element of  $E(ZERO\_BLK)$ . We denote this element by  $\delta$  and call it the *derived key*. Let  $seed$  be the evaluation of the padded message over the field element  $\delta$ . We replace the last block  $blk[blen - 1]$  by  $seed$  and use  $\mu = E(seed)$  to generate a strong pseudo random stream that will be then XORed into the rest of  $reg$ .

For each  $i \in \{0, \dots, blen - 2\}$  let  $brep(i)$  be the same as in the specification of the regular version. Let

$$A = (b[0] || \dots || b[j - 1]) \oplus right(E(\mu), j)$$

and

$$B = blk[blen - 2] \oplus E(\mu \oplus brep(blens - 1))$$

The output (ciphertext) is:

$$blk[0] \oplus E(\mu \oplus brep(1)) || blk[1] \oplus E(\mu \oplus brep(2)) || \dots \\ \dots || blk[blen - 3] \oplus E(\mu \oplus brep(blens - 2)) || B || \mu \oplus E(B)$$

## 2.2 Our Suggested Set of Algorithms

We suggest using the AES-128 standard blockcipher in any of the four modes: Julius-ECB regular version; Julius-ECB compact version; Julius-CTR

regular version; and Julius-CTR compact version, with the constant<sup>2</sup> parameter `IV_LEN` being 8 or 16 bytes. In terms of the CAESAR competition call for submissions [1], we have constant key length of 16 bytes; constant secret message number length of 0 bytes; constant public message number (IV) length of 8 or 16 bytes only; plaintext maximal length of  $2^{64} - 1$  bytes; and associated data maximal length of  $2^{64} - 1$  bytes.

The four algorithms whose `IV_LEN` is 8 bytes should only be used in case the IV mechanism is based on a counter. Stateless IV mechanism that randomly chooses the IV should use 16 bytes IV only. Assuming there is no repetition of the IV, the 8 and 16 bytes IV versions of each of the four modes are equally secure.

The two algorithms Julius-ECB regular version and Julius-CTR regular version should always be preferred over the Julius-ECB compact version and Julius-CTR compact version. The latter might be preferred for bandwidth restricted applications. The security of Julius-ECB regular version and Julius-ECB compact version is the same as of Julius-CTR regular version and Julius-CTR compact version correspondingly, except for a negligible advantage of the Julius-ECB versions in the unlikely case of partial key recovery (see subsection 3.3). Thus choosing between Julius-ECB and Julius-CTR should be based on difficulty of implementation or even personal taste. We note that Julius-CTR is inverse-free and hence might be easier to implement, yet on the other hand we think that Julius-ECB is more elegant and we like it best.

## 3 Security

### 3.1 Security Goals

Security of plaintext confidentiality and message integrity is measured by the advantage of a limited resources adversary in distinguishing between the real AE algorithm and an ideal AE algorithm, which is formalized as follows: for each triple of IV; associated data; and plaintext length  $plen$ , the ideal algorithm uniformly and randomly select an injection function  $\{0, 1\}^{8 \cdot plen} \rightarrow \{0, 1\}^{8 \cdot plen + 8 \cdot IV\_len}$  and use it to encrypt all messages with the certain IV; associated data and plaintext length  $plen$ . As for deciphering, the ideal algorithm outputs the single preimage of the given ciphertext in

---

<sup>2</sup>This parameter is a constant in the sense that it must not be changes between different messages processed by the same key, unlike the plaintext and associated data lengths arguments.

whenever a preimage exists, and returns  $\perp$  otherwise.

Let A be an adversary capable of calculating  $p$  AES encryptions or decryptions (of any 128 bit key, which might be altered between different encryptions and decryptions) that is given adaptive chosen (including chosen IV) enciphering and deciphering queries so that the total length of queries' messages is  $q$  blocks at most. Let B be an adversary with the same computational abilities of A that is given adaptive chosen enciphering only queries, so that the total length of their messages is  $q$  blocks at most. Our security goals are then:

1. The advantage of A with respect to Julius-ECB regular version or Julius-CTR regular version is about  $\frac{p}{2^{128}} + \frac{q^2}{2^{128}}$ .
2. The advantage of A with respect to Julius-ECB compact version or Julius-CTR compact version is about  $\frac{p}{2^{128}} + \frac{q}{2^{64}} + \frac{q^2}{2^{128}}$ .
3. The advantage of B with respect to Julius-ECB compact version or Julius-CTR compact version is about  $\frac{p}{2^{128}} + \frac{q^2}{2^{128}}$ .

Using the terms of the CAESAR competition call for submissions, as long as the total length of processed messages (of the same key) is much smaller than  $2^{64}$  blocks, the claimed security is specified in the following table:

Security of	Julius-ECB regular ver.	Julius-ECB compact ver.	Julius-CTR regular ver.	Julius-CTR compact ver.
Plaintext Confidentiality	$2^{128}$	$2^{128}$	$2^{128}$	$2^{128}$
Message Integrity	$2^{128}$	$2^{64}$	$2^{128}$	$2^{64}$

Table 1: Security goals

We explicitly note that our security goal for IV reuse is maximal, and that security goals are the same for 8 and 16 bytes IV lengths.

### 3.2 Security Proofs

The Julius modes of operation are provably secure in the conservative informational theoretic settings. The proof of Julius-ECB is similar to the proof of the Naor-Reingold mode [14], while the proof of Julius-CTR is similar

to the proofs of SIV [17] and Unbalanced Fiestel [19], and both proofs are based on the classic security proof of the GCM mode. Yet there are subtle details that are unique to the Julius modes and should be carefully analyzed. We shall soon publish the described security proofs in a separate document.

### **3.3 Further Security Analysis**

#### **3.3.1 Partial Security in Case of Partial Key Recovery**

We consider the not very likely case where the derived key has been compromised while the AES key hasn't. Two possible scenarios causing the partial key recovery are side channels attacks (discussed next) and a chosen ciphertext attack based on a surprising ability to receive decipherments of forged ciphertexts that have failed authentication. Such a non-trivial ability might be achieved in case of an implementation bug or a by a malware having access to the (unremoved) failed decipherments.

#### **Message Integrity**

Knowing the derived key, an adversary can alter any Julius-CTR ciphertext in such a way that the decipherment has the same seed as the original plaintext, and hence passes authentication. Alternation is done simply by XORing a delta message whose corresponding polynomial is zero at the derived key, thus the adversary can choose arbitrarily all the message's blocks except for a single block that will be (very efficiently) determined by the others. The original plaintext is not required to be known.

Forging a message in the Julius-ECB mode is a bit more difficult: an adversary knowing the derived key and a set of AES encryption values, may compose a ciphertext containing only blocks whose AES decryptions are known to the attacker, so that the polynomial evaluation of the corresponding AES block decryptions yields a seed whose AES encryption is known and fits to the first block's corresponding AES decryption. This can be done relatively efficiently, yet not all arbitrary messages (up to a single derived block) can be successfully forged.

#### **Message Confidentiality**

While a partial key recovery almost abolishes message integrity, message confidentiality is surprisingly preserved both in Julius-ECB and Julius-CTR. In a sense this confidentiality is even stronger than confidentiality of modes

like GCM, EAX [3], and OCB, as a misused IV does not affect much the confidentiality.

The explanation is simple: new messages will most probably have new seeds that have not been obtained before, and all invocations of the AES will most probably be new too. That is true up to the birthday bound, assuming the legitimate plaintexts are not related to the derived key, which is a reasonable assumption as end users practically don't know the cryptographic keys they are using.

This partial security notion is formalized by the following two-stage game: an oracle randomly and uniformly chooses a block-size permutation, and uses it in the Julius-ECB mode (for example). In the first stage the adversary may ask the oracle to encrypt messages of her (adaptive) choice (the IV is chosen too) whose total length is no more than  $q$  blocks. In the second stage, the adversary commits to a set of new messages of her choice whose total length is  $q$  blocks at most, and afterward the oracle reveals the secret derived key. Later on, the oracle flips a coin and with probability  $\frac{1}{2}$  encrypts correctly all the set's messages, and with probability  $\frac{1}{2}$  gives a set of random strings of the corresponding lengths. The mode is said to preserve confidentiality in case of partial key recovery if the advantage of the adversary guessing the right situation in the game is small.

Due to the partial preserved security in case of a partial key recovery, a reasonable heuristic is to refresh the derived key more frequently than the general AES key. A possible way to do so is variant 5.3, and there are many other more. We note that doing so might require protocol adjustments such as anti-replay mechanism or additional key agreement protocol.

### 3.3.2 Vulnerability to Side Channels Attacks

The algorithms Julius\_ECB and Julius\_CTR are not vulnerable to side channels attacks (SCA), as abstract algorithms are never vulnerable to SCA: this kind of attack is based on certain information the adversary gets in addition to the output of the abstract algorithm. Examples of such possible information are: the time taken to execute invocation of the algorithm, the rate of electricity consumption used by the implementing device, electromagnetic radiation of the device, and even the sound the device produce. Thus an abstract algorithm can never be vulnerable to SCA by itself, yet its possible implementations might certainly be.

The field of SCA cryptanalysis is relatively new, and in the near future we expect that fundamental researches will be published, and things will get clearer. Most recently a surprising and practical cryptanalysis of the

fundamental RSA algorithms has been obtained [7], and we feel that there are yet more results to come.

The logical security proofs given here can only indicate that as long as the implementation of AES and of multiplication by a secret element of  $GF(2^{128})$  are not vulnerable to SCA, the Julius algorithms are not vulnerable either. Although we have no specific knowledge in this field, we think it is reasonable to assume that there are certain implementations which are vulnerable and nevertheless it is possible to design certain efficient implementations that will not be vulnerable to SCA.

We would like to note that we consider neither Julius-ECB nor Julius-CTR as potentially more vulnerable than the widely used GCM. Another important note is that a side channel attacker usually needs to be physically close to the attacked device, hence for many simple applications a reasonable security is achieved even in the case of vulnerable implementation.

### 3.3.3 Are There Weak Keys?

Not really. An AES key so that the resulting authentication key is zero<sup>3</sup> causes:

1. The authentication of both Julius-ECB and Julius-CTR to be dependent on the last message's block only.
2. The Julius-CTR encryption to become a stream cipher which is independent of the IV, and thus is breakable by a single pair of known plaintext and ciphertext, or alternatively by just two known ciphertexts.
3. The Julius-EBC encryption to become equivalent to ECB (the only difference being additional whitening key which is independent of the IV).

Such a key can most naturally be considered a weak key. However, the arbitrary AES key 04GA7BB208D7E99CE271BAF6FE3ABAB2 (written in Hexadecimal) can be similarly shown to be even weaker: an adversary that encrypt and decrypt using this specific key will be able to forge and decrypt any message, even without knowing a single pair of plaintext and ciphertext, in case this specific key has been used. The same argument can be used to

---

<sup>3</sup>It is unclear whether such a key exists: assuming AES-128 is a sequence of  $2^{128}$  randomly and independently selected permutations in  $S_{2^{128}}$ , the probability that there is a key so that  $E(0) = 0$  is about  $1-1/e$ .

shown that any possible key is a weak key, and the obvious conclusion is that the Julius algorithms are disastrous.

More seriously, a single key by itself can never be considered a weak key. On the other hand, a cryptosystem can be considered as having weak keys, in case there is a significant probability that the random key is in a certain subset of keys all having the same undesirable property. One might most naturally suspect that the Julius algorithms have weak keys for which the sequence  $\delta, \delta^2, \delta^3, \dots$  has a short cycle, much the same way that the GCM and GMAC algorithms have already been suspected [18].

Fortunately, it follows from the security proofs that the probability of having such "weak" keys is insignificant. Nevertheless we would like to provide an intuitive explanation for that: there are exactly  $\frac{2^{64}-1}{17}$  possible derived keys  $\delta$  such that the resulting sequence  $\{\delta^i\}_{i=0}^{\infty}$  has a (not necessarily minimal) cycle of  $\frac{2^{64}-1}{17}$ . Since the maximal message length is of  $2^{60}$  blocks, which is more than  $\frac{2^{64}-1}{17}$ , the set of those  $\frac{2^{64}-1}{17}$  possible derived keys causes an undesirable phenomenon. On the other hand, for any set of  $\frac{2^{64}-1}{17}$  distinct derived keys  $\{\delta_k\}_{k=1}^{\frac{2^{64}-1}{17}}$  there is a message of  $\frac{2^{64}-1}{17}$  blocks such that all those derived keys are roots of the corresponding polynomial over  $GF(2^{128})$ . If the authenticated key is in the set  $\{\delta_k\}_{k=1}^{\frac{2^{64}-1}{17}}$ , then any two  $\frac{2^{64}-1}{17}$  blocks messages whose XOR the specified message will be encrypted similarly, which is obviously undesirable. So the suspected set of keys is not weaker than any other set of authenticated keys with the same cardinality, meaning there aren't really weak keys.

### 3.3.4 Is Julius a Trapdoor?

No. Apart from providing a rigorous security analysis and discussing the design rationale of the submitted AE algorithms, and apart from the designer's required statements, submissions to the CAESAR competition are expected to discuss the possible ways in which a weakness could be hidden in the cipher. Since the proposed algorithms use the widely trusted AES blockcipher in a certain scheme for which a conservative informational theoretic security proof is given, and uses no other keys or peculiar constants, the only possible ways we could have hidden a weakness are the following:

#### Secretly Knowing How to Break the AES

Since all currently published state of the art cryptanalyses of AES are not feasible, and that there seems to be no promising direction where a feasible



cryptanalysis might be found, having a secret method to break the AES is would be a very unlikely breakthrough. Moreover, there is an indication that even the NSA has not such ability, as we stated in the introduction.

Important note: since the security proof is based on the assumption that AES cannot be efficiently distinguished from a truly random permutation of 16 bytes, it is theoretically possible that an efficient distinguishing algorithm of the AES is enough for breaking Julius, despite having no efficient key recovery attack on the AES. However, when it comes to blockciphers, a distinguishing algorithm can be used in most cases to recover the key: by exhaustive search over a small part of the whitening key (a single byte for example), the distinguishing algorithm might be improved (when the guessed key is right) or get worse (when the guess is wrong), thus revealing what this part of the key is.

### **The Security Proof Being False Yet Seems to be True**

Informational theoretic indistinguishability proofs tend to be confusing, and it is not unreasonable that a certain fault in such a proof will not be detected by the authors of a paper and the paper's reviewers too. However we believe it is extremely unlikely that a fault will not be detected by many readers, during the five years of the competition.

Mathematical proofs, unlike mathematical conjectures, can be relatively easily verified (it only takes polynomial time!). This is especially true for birthday-security proofs that are essentially based on the inexistence of certain value repetitions, and not on high level mathematics. We encourage the reader to take the time to carefully validate each of the proofs' steps by herself or himself. Being able to exploit side channels of common implementations

Since the Julius algorithms are based on the same operations as of the GCM - meaning AES encryption and decryption, and multiplication by a secret element of the  $GF(2^{128})$  field – any method exploiting side channels of Julius implementations should be also applicable to the GCM widely used standard. To the best of our knowledge there are no such known attacks.

As the central goal of CAESAR competition is to replace the GCM by other algorithms, a reasonable entity that is able to exploits the widely used GCM would not participate in such a competition, and would definitely not encourage the cryptographic community to research the side channels aspects of modes of operation. However, we do.

In conclusion, Julius is not a trapdoor.

## 4 Features

### Maximal Security

The most notable feature of Julius is that all versions achieve the maximal possible security of a AE algorithm, meaning indistinguishability between the algorithm and a series of independent and uniformly random permutations, indexed by the IV and the messages' lengths. Thus a nonce misuse, meaning the use of the same nonce more than once, is bearable. In fact, applications whose messages are not likely to repeat (e.g. messages are long and have enough entropy) may not use IV at all and yet be secure! Note that we do not recommend zero-length IV, though.

The main advantage of Julius over GCM is in the case of nonce misuse: while Julius achieves the maximal possibly security for both confidentiality and integrity, the confidentiality of GCM completely disappears in case of a nonce repetition.

### Security in Case of Partial Key Recovery

In the not very likely case where the derived key has been recovered by a side channels attack or an implementation bug, plaintext confidentiality is preserved in the sense described in section 3.

### Provable Security

Security of both Julius-ECB and Julius-CTR is guaranteed by informational theoretic indistinguishability proofs. Meaning, breaking the cipher or forging a message can only be done by breaking AES, or exploiting side channels and implementation weaknesses (if there are any).<sup>4</sup>

### Efficiency

Julius uses about a single AES encryption per a single message's block, which by essence is the minimal number of blockcipher calls any (informational theoretic secure) mode of operation is required to make [2]. The non-blockcipher operation of Julius, meaning the field multiplication by the derived key, should be relatively fast compared to AES encryptions when correctly implemented on modern devices.

---

<sup>4</sup>That is, unless the adversary is very very lucky or that astronomical amount of data have been processed using the same key.

We note however that a fast implementation of Julius is not trivial: naively implemented finite field multiplication may take even longer than AES encryption. Implementation for a relatively old CPU which does not support carry less multiplications should not be based on implementing general  $GF(2^{128})$  multiplications. Rather, multiplication by the derived key is a certain linear transformation that should be implemented using preprocessed tables (general  $GF(2^{128})$  multiplications are unneeded). Hardware devices might use a different representation of  $GF(2^{128})$  (and of the derived key), as described in section 5.

On the other hand, implementation for a newer CPU which support carry less multiplication is relatively easy, and does not require different field representation or preprocessing tables [8]. We expect that in the near future CPUs will not only support built-in carry less multiplications but also be able to calculate binary polynomials modulus  $x^{128} + x^7 + x^2 + x + 1$ , and thus to calculate  $GF(2^{128})$  multiplications extremely efficiently.

### **Parallelizability**

Since most of the blockcipher operations in Julius are independent, meaning the input of one call to the blockcipher does not depends on the output of another call for the blockcipher, and since consecutive field multiplications by the derived key have nice algebraic properties, the Julius algorithms are highly parallelizable. While the blockcipher encryptions in Julius are naturally parallelizable, the field operations can be computed in parallel using (easily) preprocessed exponentiation of the derived key.

### **Replaceability**

The Julius algorithms are in fact general blockcipher modes of operation, and in our proposal we just suggest using them with the standard AES-128 blockcipher. If for some reason one wishes to use a different blockcipher, for example a wider blockcipher or one that is dedicated for a certain platform and thus is more efficient, the Julius modes of operation can still be used. Moreover, we believe that the widely used AES will one day be replaced by a newer standard, while a good mode of operation can last forever. Julius-CTR is inverse-free

The Julius-CTR version is inverse-free, meaning that both enciphering and deciphering make use of blockcipher encryptions only, and thus implementation of the blockcipher decryption is unneeded. This property might be beneficial for certain restricted platforms.

## Simplicity

The Julius modes of operation both have simple and elegant designs. We believe that simplicity is an important feature of a cryptographic algorithm, since it allows an easier implementation and reduces the probability of having bugs, which might make the algorithm breakable. Moreover, hiding a backdoor in software that implements a simply cryptographic algorithm is more difficult than doing so in software that implements a cumbersome algorithm.

## 5 Variants

### 5.1 Different representations of $GF(2^{128})$

The core Julius-ECB and Julius-CTR modes of operation are independent of the representation of  $GF(2^{128})$ . The regular Julius-ECB and Julius-CTR versions do not use concatenation or any other non-field operations, except for calling the permutation  $E$  or its inverse  $E^{-1}$  for deciphering (in Julius-ECB only).<sup>5</sup> Thus, replacing the specified field addition and multiplication of two blocks  $X$  and  $Y$  by  $zan^{-1}(zan(X) + zan(Y))$  and  $zan^{-1}(zan(X) \cdot zan(Y))$  correspondingly for an arbitrary bijection

$$zan : \{0, 1\}^{128} \rightarrow Z_2[x]/(x^{128} + x^7 + x^2 + x + 1)$$

does not affect much the mode, since the outcome is equivalent to applying the original mode with the permutation  $zan^{-1} \circ E \circ zan$  over the modified message obtained by block-wise applying  $zan^{-1}$  over the original message.

Since  $zan^{-1} \circ E \circ zan$  is a uniformly random permutation if and only if  $E$  is a uniformly random permutation, we can say that the core Julius modes are in a sense independent of the field representation. However, since AES with a random 128 bit key is not equivalent to  $zan^{-1} \circ AES \circ zan$  for an arbitrary  $zan$ , altering the field representation should be considered a variant of the original algorithm.

We consider two such variants, which might turn out to be much more efficient for certain platforms. There are sure many more possible varieties and tradeoffs that should be explored in order to optimize performance – those are just two techniques that can be reparameterize or even used in conjunction.

---

<sup>5</sup>And except for padding the message, which we shall ignore.

### Representation by an Irreducible Polynomial Over $GF(2^{128})$ Subfield

Let  $poly(x)$  be an irreducible monic polynomial of degree 16 over the field  $GF(2^8)$ .  $GF(2^{128}) \cong GF(2^8)/(poly(x))$ , and thus a 16 bytes block can represent an element of  $GF(2^{128})$  by using its 16 bytes as the coefficients of a (less than 16 degree) polynomial in  $GF(2^8)[x]$ . This alternative representation might enable more efficient  $GF(2^{128})$  multiplication for a CPU which does not support carry less multiplication, provided that  $GF(2^8)$  multiplications can be implemented by tables.  $GF(2^{128})$  addition is still equivalent to XOR, assuming that  $GF(2^8)$  addition of two bytes is equivalent to their XOR.

We note that preprocessing key-dependent tables for  $GF(2^{128})$  multiplication by the derived key in the original representation

yields a better performance than this alternative representation. On the other hand, doing so significantly increases the running time of the key setup.

### Key Dependent Secret Representation

We consider the variant where  $GF(2^{128})$  is represented by  $GF(2^8)/(irr(x))$  and  $irr(x)$  is a secret key-dependent irreducible binary polynomial of degree 128. In this variant  $irr(x)$  is derived from the blockcipher key instead of the secret field element  $\delta$ , which is replaced by the secret field element  $x^{128}$ . The idea of achieving universal hashing through secret finite field representation is in fact not new, however to the best of our knowledge it has never been formalized from this point of view.

Since evaluation in  $GF(2^8)/(irr(x))$  using  $\delta = x^{128}$  is equivalent to calculating the message's corresponding polynomial modulus  $irr(x)$ , this kind of universal hashing has been named "division hashing" and was treated differently than the more common "evaluation hashing" [10, 20, 15]. Indeed, those two hashes are not completely equivalent as the division hash has a smaller key space (i.e. only irreducible polynomials of degree 128). However, each of the division hash keys is equivalent to a certain different evaluation key. From this point of view, we can say that division hashing is in a sense a variant of evaluation hashing where the keys are randomly selected from a subset that is approximately 128 time smaller (in our example) than the evaluation hashing key space.

Division hashing can be most efficiently implemented in certain hardware, and the reduced security caused by the smaller universal hashing key space is insignificant as long as the same key is not used to process astronomical amount of data (note the blockcipher key space isn't reduced).

The main difficulty of this variant is the need to generate an irreducible binary polynomial. There are relatively efficient algorithms for this propose (e.g. taking an element of  $GF(2^{128})$  and checking whether its minimal polynomial is irreducible) however they are not well implemented in hardware platforms.

## 5.2 Encryption Only Mode / Chosen Ciphertext Secure Mode

One may wish to use the core Julius modes as encryption-only modes of operation, or just have the AE modes be secure to chosen ciphertexts attacks where improper forged messages are decrypted despite failing authentication (see subsection 3.3 for the motivation). For this propose we suggest to use the Naor-Reingold scheme, meaning to change Julius-ECB so that the Julius involution will be applied also to the original Julius-ECB ciphertexts.

## 5.3 Changing the Derived Key Rapidly

Since certain implementations of Julius might be vulnerable to side channels attacks which might recover the derived key  $\delta$ , it is reasonable to refresh or replace the derived key faster than the blockcipher key (provided that the key setup of the derived key is not too time consuming). One possible way to do so is by having additional 128 bits key  $m$ , and for each new message computing  $\delta = m \oplus E(IV \oplus m)$ . We note that an additional key is essential, since we already assume that an adversary might recover  $\delta$  and thus receive arbitrary values of the blockcipher. On the other hand, this simple Even-Mansour like construction is secure enough.

We note that an anti-replay mechanism might be needed in order to reject decipherment messages having a repeated IV.

## 5.4 Increased Parallelization

The Julius modes' non-field operations, i.e. blockcipher calls, are naturally and fully parallelizable. The field operations on the other hand, can only be parallelized by precomputing powers of  $\delta$ . If one wishes to increase parallelization without having a longer key setup, a simple variant might be used.<sup>6</sup> We suggest to modify the Julius involution by parti-

---

<sup>6</sup>We note that replacing the masks  $\mu \parallel \mu \cdot \delta \parallel \mu \cdot \delta^2 \parallel \dots \parallel \mu \cdot \delta^{blen}$  or  $\mu \parallel \mu \cdot (\delta + 1) \parallel \mu \cdot \delta \parallel \mu \cdot \delta^2 \parallel \dots \parallel \mu \cdot \delta^{blen}$  of the Julius involution by OCB-like alternative masks is not enough in order to make Julius-ECB more parallelizable, since evaluation of the padded message is still a

tioning the padded message into  $k$  chunks of not too many blocks, so that each chunk will be evaluated separately. Then we take the concatenation  $E(seed_1) \parallel \dots \parallel E(seed_k)$  of the encrypted chunk's evaluations, evaluate and encrypt it, and use the obtained value to create a series  $\{\mu_i\}_{i=1}^k$  of  $k$  different  $\mu$  values, each used to mask a different chunk in the same manner as in the original Julius involution.

## 6 Design Rationale

### 6.1 Origin of the Idea

The origin of Julius is one of the impossibility results presented in our recent paper [2] which develops the theory of simple linear blockcipher modes of operation. A simple linear mode is any blockcipher mode of operation which is based solely on non-secret linear (or affine) binary transformations and invocations of the underlying blockcipher. Many famous encryption and authentication schemes such as ECB, OCB, CBC, CMAC, CMC and SIV are simple linear modes, since block XORing, truncations, and multiplication by a constant element of a finite field are all non-secret affine transformations. On the other hand, GCM poly1305-AES [4] and the Julius modes of operations are not simple linear, since field multiplication is not linear and a multiplication by a (constant) secret element is linear but not non-secret.

Our mentioned impossibility result states that a simple linear mode of operation for blockcipher decipherment cannot be chosen-plaintext secure unless a  $k$ -blocks long message requires at least  $2k - 1$  invocations of the underlying blockcipher. In our notion of security the adversary is allowed to choose the plaintext as well as the IV. To the best of our knowledge all known informational theoretic chosen plaintext and ciphertext secure modes of operation, such as EME, CMC [9] and the Luby-Rackoff scheme [11], are simple linear modes that require at least  $2k$  invocation of the blockcipher for enciphering a  $k$ -blocks long message.<sup>7</sup>

A natural rising question is whether adaptive chosen plaintext and ciphertext security can be achieved by a mode of operation based on a pseudo random permutation and additional secret key, such that the mode's operations are limited to invocation of the permutation and affine transforma-

---

problem.

<sup>7</sup>That is except for our new mode Symmetric CBC (SCBC) presented in the same paper, which requires exactly  $2k-1$  invocations for such a message.

tions derived from the secret key, and the mode uses a single invocation of the permutation for each single message block. We answered the question in the affirmative by construction two such modes of operations, based on which we designed the Julius-ECB and Julius-CTR modes.

Since the CAESAR competition is more practically than theoretically oriented, and since real-world efficiency of a mode is not measured just by the total number of blockcipher invocations per a certain message length, we have made couple of changes to the original constructions: first, we have followed the GCM by replacing the additional key with the derived key  $E(0)$ . Second, we have traded the chosen-ciphertext security, which is not essential in AE algorithms, for an increased efficiency.

We have not traded the chosen-IV security, which we view as an essential part of chosen plaintext security, and that's the major advantage of Julius. We have chosen to submit two different algorithms instead of our favorite Julius-ECB only. This diversity is beneficial, as there might be applications for which Julius-CTR suits better, possibly due to its inverse-free property.

## 6.2 General Philosophy

Our theoretically-oriented philosophy is that the ideal mode of operation should use only a single blockcipher invocation per a message's block (asymptotically), and the inputs for the blockcipher should not be dependent on outputs of other blockcipher invocations. This ECB-like permutation, which is at the heart of the ideal mode, is highly parallelizable and hence makes it ideal in a sense. The other non-blockcipher operations, are supposed to be much faster, and thus can almost be ignored.

Moreover, we believe that CPU and hardware architectures should be adjusted to cryptography (so that those non-blockcipher operations could truly be ignored) at least as much as cryptography should be adjusted to CPU and hardware architectures. Sophisticated and impressive cryptographic algorithms such as poly1305-AES [4] have been designed in order to overcome limited architectures, while this is obviously unneeded. For example, supporting polynomial multiplications over  $GF(2)$ , which is very beneficial for software implementations of binary fields' multiplications, is not more difficult than supporting binary integer multiplication, which is provided in any CPU.

Fortunately, today's hardware manufacturers are much more cryptographic-aware, and carry-less 64bit multiplication is provided by many new CPUs since Intel's 2010 Westmere processor [8]. We believe this trend should and



will continue within the next few years, and hope the decisions of the CAESAR committee will take this into considerations.

### 6.3 Justifying Some Choices

#### The Underlying Blockcipher

We chose AES as our underlying blockcipher since it is widely believed to be absolutely secure, and has efficient software and hardware implementations. The AES standard specifies three algorithms: AES-128, AES-192 and AES-256, having 128, 192 and 256 bit keys, and 10, 12 and 14 rounds, correspondingly. While one might psychologically feel that AES-192 and AES-256 are more secure than AES-128, there is no evidence for that. On the contrary: the key schedule of both AES-192 and AES-128 has been shown to be vulnerable to related key attacks [7] while the AES-128 hasn't. Due to that, and to the fact that AES-128 is a bit faster, we have chosen AES-128.

#### The Regular and Compact Versions

Two versions are provided for each of the two Julius-ECB and Julius-CTR modes. The regular versions have an overhead of 16 bytes or more (depends on the exact lengths of the plaintext and the associated data), while the compact versions have always exactly 8 bytes overhead. In order to achieve that, the compact versions use truncations of blocks and thus are a slightly more complicated.

The probability of a forged message to pass authentication is no less than  $\frac{1}{2}$  to the power of the overhead length measured in bits, however the more known plaintexts and ciphertexts the adversary knows the higher is the probability of finding a certain collision which enables the recovery of the derived key and thus the ability to forge messages. In a theoretical sense, authentication with an overhead of 16 bytes is not stronger than 8 bytes overhead authentication, since the expected number of enciphering or deciphering queries the adversary is going to make before being able to forge a message is the same. In a practical sense, both 16 and 8 bytes overhead authentication are fine.

We provide the two versions so that a more conservative user may use the regular versions, and a user who is more concerned with the overhead's costs may use the compact versions.

## **Paddings**

The intermediate paddings of Julius-ECB and Julius-CTR are different, as the 8 or more zero bytes are placed between the associated data and the plaintext in the Julius-ECB padding and right after the message in the Julius-CTR padding. This enables the Julius-CTR regular version to be slightly less complicated as the compact version, since there is no need to decrypt the last ciphertext block. As for Julius-ECB, placing the zeroes before the plaintext enables a faster authentication verification: there is no need to complete the calculations of the Julius involution when the message fails authentication.

We note that both padding of Julius-ECB and Julius-CTR place the associated data right after the IV and lengths, and before the plaintext. This enables to start the authentication verification while the decipherment of the ciphertext has not been completed.

## **Plaintext and Associated Data Length Limitation**

The formal maximal length of plaintext or associated data is of  $2^{64}-1$  bytes, and since messages are not expected to grow so long in the near future, practically there is no length limitation at all. There are two reasons for this unachievable limitation: first, security is no longer guaranteed in case a single key is being used for authenticating and encrypting more than  $2^{64}-1$  bytes, not even when all those bytes are within a single message. Second, the lengths of a certain message's plaintext and associated data are being authenticated by being included in the message's padding. We use 8 bytes for each of the two lengths, and therefore the limit is of  $2^{64}-1$  bytes.

## **The IV Lengths.**

While all the described modes of operation may receive IV of any (reasonable) length, officially we suggest only two possible lengths, 8 bytes and 16 bytes, intended for different IV mechanisms:

- Counter based IV mechanism don't need more than 8 bytes of IV since a single AES key is not likely to be used in more than  $2^{64}$  different messages encryptions (and if it is, security is no longer guaranteed).
- Randomly selected IV of less than 16 bytes will be repeated within less than  $2^{64}$  messages, which is undesirable. On the other hand, there

is no point having a longer IV since repetition of the seed will occur within  $2^{64}$  messages anyway.

We have decided not to recommend widely used IV lengths such as 12 bytes, since we are afraid that a developer who lacks basic cryptographic understanding might believe that a certain intermediate IV value is secure for random IV mechanism, while it isn't. Anyone who do have sufficient cryptographic background and wishes to have a different IV length or any other modification will easily be able to do so securely.

## Acknowledgments

The designer would like to thank Orr Dunkleman, Adi Shamir, Stefano Tessaro and Boaz Tsaban for discussing the algorithms. Yet another contribution of Boaz was to brilliantly suggest the name *Julius*. Moreover, the designer would like to thank the international team of cryptographers who founded the CAESAR competition for encouraging the research of AE algorithms and modes of operation.

## References

- [1] Caesar: Competition for authenticated encryption: Security, applicability, and robustness, 2013. URL: <http://competitions.cr.yp.to/caesar-call.html>.
- [2] Lear Bahack. On simple linear modes of operation for blockcipher encryption and authentication. To be published soon, 2014.
- [3] Mihir Bellare, Phillip Rogaway, and David Wagner. The eax mode of operation. In *FSE*, pages 389–407, 2004.
- [4] Daniel J. Bernstein. The poly1305-aes message-authentication code. In *FSE*, pages 32–49, 2005.
- [5] Alex Biryukov, Orr Dunkelman, Nathan Keller, Dmitry Khovratovich, and Adi Shamir. Key recovery attacks of practical complexity on aes-256 variants with up to 10 rounds. In *EUROCRYPT*, pages 299–319, 2010.
- [6] Joan Daemen and Vincent Rijmen. Rijndael for aes. In *AES Candidate Conference*, pages 343–348, 2000.

- [7] Daniel Genkin, Adi Shamir, and Eran Tromer. Rsa key extraction via low-bandwidth acoustic cryptanalysis. *IACR Cryptology ePrint Archive*, 2013:857, 2013.
- [8] Shay Gueron and Michael E. Kounavis. Efficient implementation of the galois counter mode using a carry-less multiplier and a fast reduction algorithm. *Inf. Process. Lett.*, 110(14-15):549–553, 2010.
- [9] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In *CRYPTO*, pages 482–499, 2003.
- [10] Hugo Krawczyk. Lfsr-based hashing and authentication. In *CRYPTO*, pages 129–139, 1994.
- [11] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988.
- [12] David A. McGrew and John Viega. The security and performance of the galois/counter mode (gcm) of operation. In *INDOCRYPT*, pages 343–355, 2004.
- [13] Kurt Mehlhorn and Uzi Vishkin. Randomized and deterministic simulations of prams by parallel machines with restricted granularity of parallel memories. *Acta Inf.*, 21:339–374, 1984.
- [14] Moni Naor and Omer Reingold. The nr mode of operation. URL: <http://www.wisdom.weizmann.ac.il/naor/PAPERS/nr-mode.ps>, 1997.
- [15] Wim Nevelsteen and Bart Preneel. Software performance of universal hash functions. In *EUROCRYPT*, pages 24–41, 1999.
- [16] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. Ocb: a block-cipher mode of operation for efficient authenticated encryption. In *ACM Conference on Computer and Communications Security*, pages 196–205, 2001.
- [17] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In *EUROCRYPT*, pages 373–390, 2006.
- [18] Markku-Juhani Olavi Saarinen. Cycling attacks on gcm, ghash and other polynomial macs and hashes. In *FSE*, pages 216–225, 2012.

- [19] Bruce Schneier and John Kelsey. Unbalanced feistel networks and block cipher design. In *FSE*, pages 121–144, 1996.
- [20] Victor Shoup. On fast and provably secure message authentication based on universal hashing. In *CRYPTO*, pages 313–328, 1996.

## **Appendix: Required and Additional Statements**

The designer has not hidden any weaknesses in this cipher.

The Julius mode of operation, or any of its many variants described in this document, is not and will not be subject to patents. 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. The designer will not be responsible for the usage of Julius or any of its variants. One should have no claims to the designer regarding the usage of Julius.

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 he disagrees with published analyses then he is 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.

The submitter consents to possible modification of the submitted algorithms by the committee, as long as the core ideas of the algorithms are preserved and the modified algorithms are provable birthday-secure.

Lear Bahack,  
Designer and submitter.