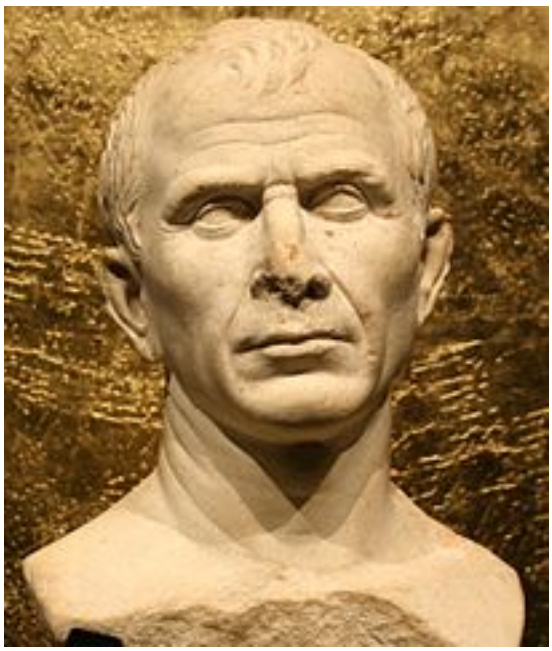
# Julius: Secure Mode of Operation for Authenticated Encryption Based on ECB and finite field multiplications

Lear Bahack*

Submission to the CAESAR competition, version 1.0, March 2014



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

*Weizmann Institute of Science, Rehovot, Israel. E-mail: lear.bahack@gmail.com

**Abstract**

We present two new block cipher 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 is a completely new scheme. We specify two versions for each mode: a regular version and a compact version, having deferent ciphertexts redundancies. Several variants aimed to achieve increased parallelization, security beyond the birthday bound, and security to chosen ciphertext attacks, are briefly explored.

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

# 1 Introduction

# 2 Specification

We specify first the two block cipher 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 block cipher, whose specification is given in [9].

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 $\{0, 1, \ldots, 256\}$, 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 $a_0, .., a_7, b_0, .., b_7 \in \{0, 1\}$ , is the byte $A = \sum_{i=0}^{7} 2^i (a_i \oplus b_i)$, 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: $str1[0] \oplus str[0] \| \ldots \| str[len-1] \oplus str[len-1]$.

4. A block is defined as a string of exactly 16 bytes. We denote the two blocks: $0 \| 0 \| \ldots \| 0$ and $0 \| 0 \| \ldots \| 0 \| 1$ by $ZERO\_BLK$ and $ONE\_BLK$ correspondingly. $E(blk)$ and $E^{-1}(blk)$ stands for the block cipher encryption and decryption of the block $blk$.

5. For a block $blk = b[0] \| b[1] \| \ldots \| b[15]$, and an integer $len \in \{0, 1, \ldots, 16\}$ we denote the string $b[0] \| \ldots \| b[i-1]$ of the first/leftmost bytes of $blk$ by $left(blk, len)$ and the string $b[16–i] \| \ldots \| b[15]$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 $blk = b[0] \| .. \| b[15]$ is

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

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

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 $\beta_0, \ldots \beta_{len-1}$ be the $GF(2^{128})$ field elements corresponding to the blocks $blk[0], \ldots, blk[len-1]$, then the polynomial corresponding to the string $blk[0] \| ... \| blk[len–1]$ is $\sum_{d=0}^{len-1} x^d \cdot \beta_{len-1-d}$. Moreover, we define the evaluation of this string over the field element $\alpha$ as the corresponding block of $\sum_{d=0}^{len-1} \alpha^d \cdot \beta_{len-1-d}$.

9. The inputs for the two Julius modes are:

- IV string of length $IV\_LEN$: $IV = IV[0] \| \ldots \| IV[IV\_LEN]$. The non-negative integer $IV\_LEN$ is a (constant) parameter of the mode.

- Length of the plaintext, which is an integer between 0 and $2^{64} - 1 = 18,446,744,073,709,551,615$. The given length $plen$ is represented by the 8 bytes string $plen[0] \| \ldots \| plen[7]$ so that $plen = \sum_{i=0}^{7} 2^{8i} plen[7-i]$.

- Length of the associated data, which is an integer between 0 and $2^{64}-1 = 18,446,744,073,709,551,615$. The given length $adlen$ is represented by the 8 bytes string $adlen[0]\|\ldots\|adlen[7]$ so that $adlen = \sum_{i=0}^{7} 2^{8i} adlen[7-i]$.
- Plaintext byte string: $plain = plain[0]\|\ldots\|plain[plen–1]$.
- Associated data byte string: $ad = ad[0]\|\ldots.\|ad[adlen–1]$.
- 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 two Julius variants having different representations of $GF(2^{128})$.
2. For any $\alpha, \mu \in GF(2^{128})$ and non negative integer $k$, the evaluations of the strings $\mu\|\mu\cdot\alpha\|\mu\cdot\alpha^2\|....\|\mu\cdot\alpha^{2k-1}$ and $\mu\|\mu\cdot(\alpha+1)\|\mu\cdot\alpha\|\mu\cdot\alpha^2\|....\|\mu\cdot\alpha^{2k}$ 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

   $$blk[0]\|\ldots\|blk[len-1]$$

   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:

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

The substring of the last/rightmost $pres+16+plen$ bytes is denoted by $reg = blk[0]\|\ldots\|blk[blen–1]$ where $blen = 1+\lceil\frac{plen}{16}\rceil$ and $blk[0],\ldots,blk[blen-1]$ 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:

$$\mu \| \mu \cdot \delta \| \mu \cdot \delta^2 \| .... \| \mu \cdot \delta^{blen}$$

else, the mask is

$$\mu \| \mu \cdot (\delta + 1) \| \mu \cdot \delta \| \mu \cdot \delta^2 \| .... \| \mu \cdot \delta^{blen}$$

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 = $reg = blk[0] \| \ldots \| blk[blen–1]$ (note that the blocks $blk[0], \ldots, blk[blen-1]$ have been updated by the Julius Involution). The ciphertext string is then:

$$E(blk[0]) \| \ldots \| E(blk[blen-1])$$

### 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:

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

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

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

where $blen = \lfloor \frac{8+plen}{16} \rfloor$, $j = plen + 8 - 16 \cdot blen$, $b[0], \ldots, b[j-1]$ are bytes and $blk[0], \ldots, blk[blen-1]$ 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 $\mu = E(E(seed))$ and

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

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

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

else, the mask is

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

The mask is then XORed into the string reg.

#### Modifyed ECB

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

The ciphertext string is then:

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

### 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:

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

The substring of the last/rightmost $pres + 16 + plen$ bytes is denoted by $reg = blk[0] \| \ldots \| blk[blen–1]$ where $blen = 1 + \lceil \frac{plen}{16} \rceil$ and $blk[0], \ldots, blk[blen - 1]$ 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 $i \in \{0, ..., blen-2\}$ we denote by $brep(i)$ the block-width binary representation of $i$, meaning $i = \sum_{j=0}^{15} 2^{8j} b[15-j]$ for $brep(i) = b[0]\|...\|b[15]$.

The output (ciphertext) is:

$$blk[0] \oplus E(\mu \oplus brep(0)) \| blk[1] \oplus E(\mu \oplus brep(1)) \| ... \| blk[blen-2] \oplus E(\mu \oplus brep(blen-2)) \| \mu$$

#### 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:

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

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

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

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

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

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

and

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

The output (ciphertext) is:

$$\begin{aligned} & blk[0] \oplus E(\mu \oplus brep(1)) \| blk[1] \oplus E(\mu \oplus brep(2)) \| ... \\ & ... \| blk[blen-3] \oplus E(\mu \oplus brep(blen-2)) \| B \| \mu \oplus E(B) \end{aligned}$$

### 2.2 Our Suggested Set of Algorithms

We propose and recommend 8 AE algorithms that are all the possible combinations of (Block cipher, IV_LEN, mode of operation) $\in$ {AES-128}$\times${8 bytes, 16 bytes}$\times${Julius-ECB regular, Julius-ECB compact, Julius-CTR regular, Julius-CTR compact}

# 3 Variants

# 4 Design Rationale

## 4.1 Origin of the Idea

The origin of Julius is one of the impossibility results presented in our recent paper [3] which develops the theory of simple linear block cipher modes of operation. A simple linear mode is any block cipher mode of operation which is based solely on non-secret linear (or affine) binary transformations and invocations of the underlying block cipher. 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 fields are all non-secret affine transformations. On the other hand, GCM poly1305-AES [] 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 (concerning binary fields only) but not non-secret.

Our mentioned impossibility result states that a simple linear mode of operation for block cipher decipherment cannot be chosen-plaintext secure unless a $k$-blocks long message requires at least $2k - 1$ invocations of the underlying block cipher. 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 [] and the Luby-Rackoff scheme [], are simple linear modes that require at least $2k$ invocation of the block cipher for enciphering a $k$-blocks long message . [1]

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 transformations 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 block cipher 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

[1] 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.

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 block cipher encryptions-only property.

## 4.2 General Philosophy

Our theoretically-oriented philosophy is that the ideal mode of operation should use only a single block cipher invocation per a message's block (asymptotically), and the inputs for the block cipher should not be dependent on outputs of other block cipher 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-block cipher 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-block cipher 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 [6] 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 [11]. 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.

## 4.3 Justifying Some Choices

### The Underlying Block Cipher.

We chose AES as our underlying block cipher 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 [10] 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).
- Random based IV of less than 16 bytes will cause IV repetition 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]

[2] Caesar: Competition for authenticated encryption: Security, applicability, and robustness, 2013. URL: http://competitions.cr.yp.to/caesar-call.html.

[3] Lear Bahack. On simple linear blockcipher modes of operation. To be published soon, 2014.

[4] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT*, pages 409–426, 2006.

[5] Mihir Bellare, Phillip Rogaway, and David Wagner. The eax mode of operation. In *FSE*, pages 389–407, 2004.

[6] Daniel J. Bernstein. The poly1305-aes message-authentication code. In *FSE*, pages 32–49, 2005.

[7] 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.

[8] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[9] Joan Daemen and Vincent Rijmen. Rijndael for aes. In *AES Candidate Conference*, pages 343–348, 2000.

[10] Daniel Genkin, Adi Shamir, and Eran Tromer. Rsa key extraction via low-bandwidth acoustic cryptanalysis. *IACR Cryptology ePrint Archive*, 2013:857, 2013.

[11] 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.

[12] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In *CRYPTO*, pages 482–499, 2003.

[13] Hugo Krawczyk. Lfsr-based hashing and authentication. In *CRYPTO*, pages 129–139, 1994.

[14] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988.

[15] Ueli M. Maurer, Krzysztof Pietrzak, and Renato Renner. Indistinguishability amplification. In *CRYPTO*, pages 130–149, 2007.

[16] David A. McGrew and John Viega. The security and performance of the galois/counter mode (gcm) of operation. In *INDOCRYPT*, pages 343–355, 2004.

[17] 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.

[18] Chanathip Namprempre, Phillip Rogaway, and Tom Shrimpton. Ae5 security notions: Definitions implicit in the caesar call. *IACR Cryptology ePrint Archive*, 2013:242, 2013.

[19] Wim Nevelsteen and Bart Preneel. Software performance of universal hash functions. In *EUROCRYPT*, pages 24–41, 1999.

[20] 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.

[21] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In *EUROCRYPT*, pages 373–390, 2006.

[22] Markku-Juhani Olavi Saarinen. Cycling attacks on gcm, ghash and other polynomial macs and hashes. In *FSE*, pages 216–225, 2012.

[23] Bruce Schneier and John Kelsey. Unbalanced feistel networks and block cipher design. In *FSE*, pages 121–144, 1996.

[24] 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.