YAES v2[1]

Designers: Antoon Bosselaers and Fre Vercauteren
ESAT/COSIC, KU Leuven, Belgium

Submitters: Antoon Bosselaers and Fre Vercauteren
frederik.vercauteren@esat.kuleuven.be

2014.05.14

[1] YAES is shorthand for Yet Another AES-based Authenticated Encryption Scheme

1 Specification

1.1 Parameters

YAES has three parameters: key length, nonce length and tag length. The key length will be equal to 16 bytes (128 bits), the nonce length can be between 1 and 16 bytes (but maximum of 127 bits set), and the tag length is between 8 and 16 bytes.

1.2 Recommended parameter sets

Primary recommended parameter set `yaes128v2`: 16-byte (128 bit) key, 16-byte (127 bit, this is not a typo) nonce, 16-byte (128 bit) tag.

1.3 Authenticated encryption

Notation and basic building blocks

- $\{\}^*$: set of finite length binary strings
- 0^k: a string of k zero bits
- 1^k: a string of k one bits
- $\oplus$: bit-wise exclusive OR
- $||$: concatenation
- **A**: associated data
- **M**: plaintext
- **K**: key
- **N**: public message number
- **C**: ciphertext
- **T**: tag
- $|X|$: bit length of binary string $X \in \{\}^*$ where the length of the empty string is 0.
- a binary string $a_0a_1a_2a_3a_4a_5\cdots$ is represented as a byte array A with

$$A[i] = \texttt{0x}a_{8i}a_{8i+1}a_{8i+2}a_{8i+3}a_{8i+4}a_{8i+5}a_{8i+6}a_{8i+7}\,,$$

  in particular $A[0] = \texttt{0x}a_0a_1a_2a_3a_4a_5a_6a_7$.
- $\text{msb}_b(X)$: first b bits of binary string $X \in \{\}^*$ where $b \leq |X|$
- $A0^*$: if $|A| < 128$, $A \,||\, 0^{128-|A|}$ else $\text{msb}_{128}(A)$
- $A10^*$: $A \,||\, 10^*$. Note that due to the 0^* operator, the result is always 128 bits.
- $|X|_b := \lceil |X|/b \rceil$: number of b-bit blocks in binary string $X \in \{\}^*$
- $X[0] \,||\, X[1] \,||\, \cdots \,||\, X[x] \xleftarrow{b} X$ with $x = |X|_b$ denotes b-bit partitioning of X (last block can be partial block)
- $\mathbb{F}_{2^{128}}$: the Galois field with 2^{128} elements represented as $\mathbb{F}_{2^{128}}\langle\theta\rangle := \mathbb{F}_2[x]/f(x)$ where $f(x) = x^{128} + x^7 + x^2 + x + 1$. We note that $f(x)$ is a primitive polynomial, i.e. the element θ (or x by abuse of notation) generates the full multiplicative group
- a 128-bit string $a = a_0 \cdots a_{127}$ can be naturally interpreted as an element of $\mathbb{F}_{2^{128}}$ as $a_0 + a_1x + \cdots + a_{127}x^{127}$

– $x \cdot a$ (resp. $x^i \cdot a$) with $a \in \mathbb{F}_{2^{128}}$: multiplication by x (resp. x^i), note that this can be implemented by a right shift and an XOR (resp. repeated right shifts and XORs). In particular, given the bit string representation of $a = a_0 a_1 \cdots a_{127}$, the bit string representation of $x \cdot a$ is given by

$$x \cdot a = a_{127}(a_0 \oplus a_{127})(a_1 \oplus a_{127})a_2 a_3 a_4 a_5 (a_6 \oplus a_{127}) a_7 \cdots a_{126} .$$

– `AES128Round`(S, K): one full AES-128 round with 16-byte state S and 16-byte round key K. This corresponds to the instruction `_m128_aesenc_si128(S, K)` on processors that support AES-NI.
– `AES128RoundKey`(K, i): returns the round key in the i-th round of AES128, where by definition `AES128RoundKey`$(K, 0)$ returns K
– `AES128[r]Rounds`(M, K, i): r full AES-128 rounds starting from the i-th AES-128 round with M 16-byte message and K the 16-byte AES key (the round keys are derived from K using the standard AES-128 key schedule starting from the i-th round). The pseudo-code of `AES128[r]Rounds`(M, K, i) is given below.

Algorithm `AES128[r]Rounds`(M, K, i)

1. $S \leftarrow M$
2. **for** $j = 0$ **to** $r - 1$ **do**
3. $\quad K_j \leftarrow$ `AES128RoundKey`$(K, i + j)$
4. $\quad S \leftarrow$ `AES128Round`(S, K_j)
5. **return** S

– `AES128`(M, K): standard full AES-128 with 16-byte message M and 16-byte key K

The inputs to the authenticated encryption function `YAES128_ENC` are a public message number **N** of maximum 127 bits, associated data **A**, a plaintext **M**, where the total combined length of **A** and **M** is maximum 2^{48} bytes, a key **K** of 16 bytes and a tag length $64 \leq \tau \leq 128$. The output is the ciphertext **C** of exactly the same bit length as **M** and a tag **T** of length τ. The function uses two subroutines `AD128` and `EF128` for authenticating associated data and encrypting and authenticating the message respectively.
The inputs to the authenticated decryption function `YAES128_DEC` are a public message number **N**, associated data **A**, a ciphertext **C**, a key **K** and a tag **T** of length τ. The output is the plaintext **M** or the error symbol $\perp$ when the input is deemed invalid.
The full specification of the `YAES128` encryption/decryption functions are given on the next page.

Algorithm $\texttt{YAES128\_ENC}(\mathbf{N}, \mathbf{A}, \mathbf{M}, \mathbf{K}, \tau)$

1. **if** $|\mathbf{A}| > 0$ **then** $\mathrm{TA} \leftarrow \texttt{AD128}(\mathbf{A}, \mathbf{K})$
2. **else** $\mathrm{TA} = 0^{128}$
3. $(\mathbf{C}, \mathrm{TE}) \leftarrow \texttt{EF128}(\mathbf{N}, \mathbf{M}, \mathbf{K})$
4. $\mathbf{T} = \mathrm{TE} \oplus \mathrm{TA}$
5. **return** $(\mathbf{C}, \mathrm{msb}_\tau(\mathbf{T}))$

Algorithm $\texttt{YAES128\_DEC}(\mathbf{N}, \mathbf{A}, \mathbf{C}, \mathbf{K}, \mathbf{T}, \tau)$

1. **if** $|\mathbf{A}| > 0$ **then** $\mathrm{TA} \leftarrow \texttt{AD128}(\mathbf{A}, \mathbf{K})$
2. **else** $\mathrm{TA} = 0^{128}$
3. $(\mathbf{M}, \mathrm{TE}) \leftarrow \texttt{DF128}(\mathbf{N}, \mathbf{C}, \mathbf{K})$
4. $\mathbf{T}' = \mathrm{TE} \oplus \mathrm{TA}$
5. **if** $\mathrm{msb}_\tau(\mathbf{T}') = \mathbf{T}$ **then return** $\mathbf{M}$
6. **else return** $\perp$

Algorithm $\texttt{EF128}(\mathbf{N}, \mathbf{M}, \mathbf{K})$

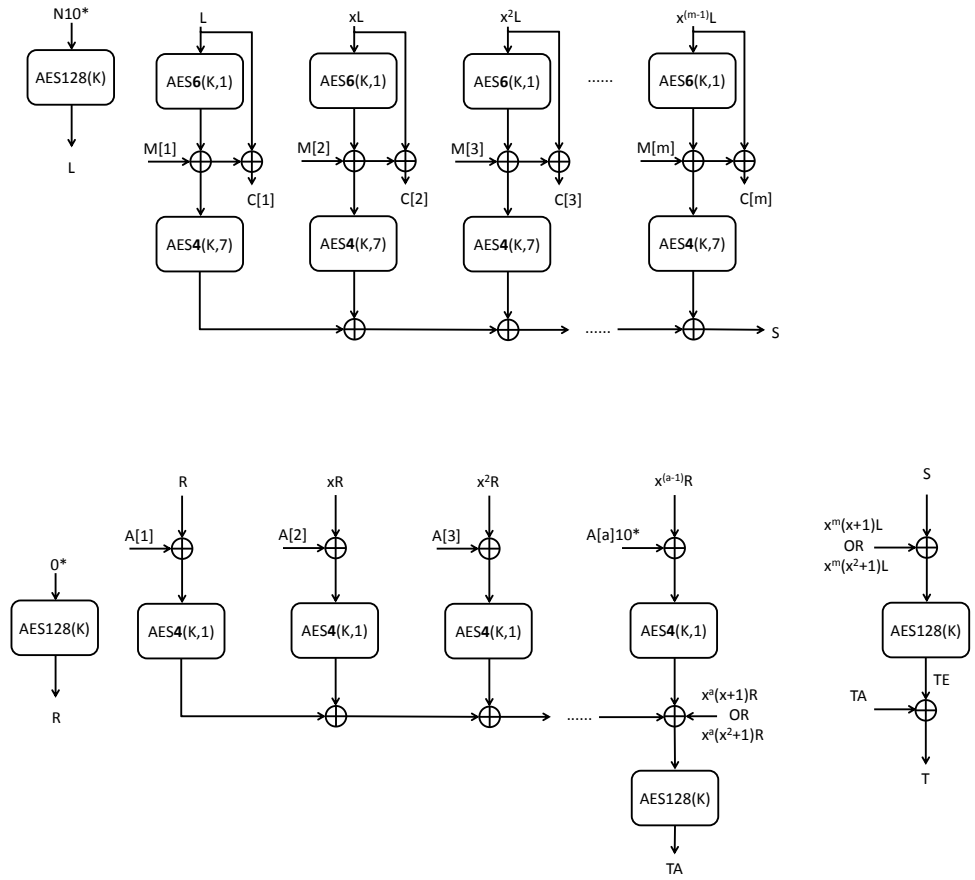
1. $m \leftarrow |\mathbf{M}|_{128}$; $\mathbf{M}[1] \,||\, \cdots \,||\, \mathbf{M}[m] \xleftarrow{128} \mathbf{M}$
2. $L \leftarrow \texttt{AES128}(\mathbf{N}10^*, \mathbf{K})$
3. $S \leftarrow 0^{128}$
4. **for** $i \leftarrow 1$ **to** m **do**
5. $\quad V \leftarrow \texttt{AES128[6]Rounds}(L, \mathbf{K}, 1)$
6. $\quad V \leftarrow V \oplus \mathbf{M}[i]10^*$
7. $\quad S \leftarrow S \oplus \texttt{AES128[4]Rounds}(V, \mathbf{K}, 7)$
8. $\quad \mathbf{C}[i] \leftarrow \mathrm{msb}_{|\mathbf{M}[i]|}(V \oplus L)$
9. $\quad L \leftarrow x \cdot L$
10. **if** $|\mathbf{M}[m]| = 128$ **then** $L \leftarrow (x+1) \cdot L$
11. **else** $L \leftarrow (x^2+1) \cdot L$
12. $\mathrm{TE} \leftarrow \texttt{AES128}(S \oplus L, \mathbf{K})$
13. **return** $(\mathbf{C}, \mathrm{TE})$

Algorithm $\texttt{DF128}(\mathbf{N}, \mathbf{C}, \mathbf{K})$

1. $c \leftarrow |\mathbf{C}|_{128}$; $\mathbf{C}[1] \,||\, \cdots \,||\, \mathbf{C}[c] \xleftarrow{128} \mathbf{C}$
2. $L \leftarrow \texttt{AES128}(\mathbf{N}10^*, \mathbf{K})$
3. $S \leftarrow 0^{128}$
4. **for** $i \leftarrow 1$ **to** c **do**
5. $\quad V \leftarrow \texttt{AES128[6]Rounds}(L, \mathbf{K}, 1)$
6. $\quad \mathbf{M}[i] \leftarrow \mathbf{C}[i] \oplus \mathrm{msb}_{|C[i]|}(V \oplus L)$
7. $\quad V \leftarrow V \oplus \mathbf{M}[i]10^*$
8. $\quad S \leftarrow S \oplus \texttt{AES128[4]Rounds}(V, \mathbf{K}, 7)$
9. $\quad L \leftarrow x \cdot L$
10. **if** $|\mathbf{C}[c]| = 128$ **then** $L \leftarrow (x+1) \cdot L$
11. **else** $L \leftarrow (x^2+1) \cdot L$
12. $\mathrm{TE} \leftarrow \texttt{AES128}(S \oplus L, \mathbf{K})$
13. **return** $(\mathbf{M}, \mathrm{TE})$

Algorithm $\texttt{AD128}(\mathbf{A}, \mathbf{K})$

1. $a \leftarrow |\mathbf{A}|_{128}$; $\mathbf{A}[1] \,||\, \cdots \,||\, \mathbf{A}[a] \xleftarrow{128} \mathbf{A}$
2. $R \leftarrow \texttt{AES128}(0^{128}, \mathbf{K})$
3. $S \leftarrow 0^{128}$
4. **for** $i \leftarrow 1$ **to** a **do**
5. $\quad V = \mathbf{A}[i]10^* \oplus R$
6. $\quad S \leftarrow S \oplus \texttt{AES128[4]Rounds}(V, \mathbf{K}, 1)$
7. $\quad R \leftarrow x \cdot R$
8. **if** $|\mathbf{A}[a]| = 128$ **then** $R \leftarrow (x+1) \cdot R$
9. **else** $R \leftarrow (x^2+1) \cdot R$
10. $T \leftarrow \texttt{AES128}(S \oplus R, \mathbf{K})$
11. **return** T



2 Security goals

We have the following security statements for a nonce respecting adversary and for the recommended parameters:

- key recovery attack: security 128 bits
- confidentiality of plaintext: security 64 bits
- integrity of plaintext/associated data: security 55 bits

The numbers given above mean the following: the advantage of the adversary for key recovery goes up as $k/2^{128}$ where k is the number of key guesses. The advantage of the adversary for breaking the indistinguishability-from-random goes up as $q^2/2^{128}$ where q is roughly the total number of blocks that are submitted to encryption oracle. The advantage of the adversary of producing a valid forgery (i.e. ciphertext/tag pair that validates and was not generated by encryption oracle) goes up as $q^2\epsilon + q^2/2^{128}$ where q is roughly the total number of blocks that are submitted to encryption oracle

and ϵ is such the implicit hash function used is ϵ-AXU (almost XOR universal). The lower number for the integrity of plaintext/associated data therefore thus comes from the fact that the best bound on 4-round AES is only ϵ-AXU with $\epsilon = 2^{-113}$.

3 Security analysis

The security of `YAES128` can be derived from general principles in that it combines a tweaked CTR mode with an ϵ-AXU hash function based approach for the authentication part. Although we did not bother to write down a full security proof, this can be done along the lines of the security proof for CTR-GCM (without the complications of long nonces) or OCB3.
The security of `YAES128` is only guaranteed under the following assumptions:

- Keys are uniform random.
- The nonce really is a nonce, and therefore should not be used more than once.
- The decrypted plaintext should only be revealed upon successful verification of the tag.

More interesting are the practical implications of not using the full AES block cipher. The first assumption is that `AES128Round`$[6](x^iL, K, 1) \oplus x^iL$ should be indistinguishable from independent uniform strings. Although we are not aware of any formal analysis of such constructions, we expect this assumption to hold up in practice. Note that unlike classical CTR mode, the input x^iL is unknown to the attacker, as is the output. Secondly, it is well known that 4 round AES is ϵ-AXU with $\epsilon = 2^{-113}$, which leads to a somewhat less than ideal security margin for authentication. However, since the total length of the associated data $\mathbf{A}$ and the plaintext $\mathbf{M}$ is limited to 2^{48} bytes (i.e. 2^{44} blocks), we do not expect to find a collision with probability substantially higher than 2^{-24}.
Note that the recent LOCAL attack on ALE does not generalize to `YAES128` since no state bytes are leaked directly into the ciphertext.

4 Features

`YAES128` has the following features:

- Fully parallel: `YAES128` is fully parallel on the message block level, both for encryption and authentication, and therefore can be trivially pipelined by encrypting consecutive blocks simultaneously. Note that the required mask at any given position can be computed efficiently by a simple square and multiply algorithm. This implies that even on multicore processors a single message could be encrypted/decrypted by splitting it over the different cores.
- Tweaked CTR mode: the encryption itself is a simple tweaked CTR, where the input (in this case all zero) and output are masked. Note that this is different from standard CTR mode, where the input to the block cipher is known (since it is derived from the public message number $\mathbf{N}$). Furthermore, given any plaintext/ciphertext pair, it is easy to also derive the output of the block cipher in normal CTR mode. Due to the tweaks this is no longer possible.

– Block cipher encryption only: `YAES128` only uses the AES block cipher in encryption mode. This is one major advantage over AES-OCB3 where also decryption is required; this implies that an implementation of `YAES128` will always be smaller than AES-OCB3 both in software and hardware.
– No "full" finite field multiplications: the only finite field operations are simple XOR or multiplication by the constant x. This is a major advantage over AES-GCM that requires a full finite field multiplier, which not only leads to a large circuit in hardware, but also implies extra functionality to be implemented in software.
– Re-use of existing implementations: it is easy to see that `YAES128` can be derived trivially from existing AES-CTR or AES-OCB3 mode implementation by making very minor modifications, mostly redefining how the counter is updated and the injection of the plaintext block after the 6th round and the extraction of the corresponding ciphertext block.
– Efficiency: given the above remark, we can see that the efficiency of `YAES128` will be very similar to AES-OCB3. As an indication of the expected performance, we provide cycle counts of AES-OCB3 using the same doubling approach for masking (recent work by Bogdanov et. al): 1.94 cpb for 128 byte message, 0.77 cpb for 1024 byte message, and 0.72 byte for 2048 byte message. This shows that `YAES128` will perform very well both for short and longer messages.
– Low overhead: compared to AES-CTR mode, the overhead is roughly two full AES encryptions when there is no associated data and one extra full AES encryption to handle associated data.
– Medium state size: `YAES128` uses four 128-bit states for the round keys, tweaks, message blocks and authentication state.
– Incremental if decryption available: modifying one plaintext block modifies the corresponding ciphertext block (that can be recomputed very efficiently), but also modifies the tag. To update the tag, it is easy to see that one would also need `AES128` decryption, so if one is willing to accept this cost, `YAES128` can be made incremental.
– Precomputation and preprocessing: similar to AES-GCM, it is possible to precompute the keystream without seeing **M** or **A**; it is possible to preprocess **A** without seeing **M** or the public message number **N**; it is possible to preprocess **M** without seeing **A**; and in general it is possible to preprocess various parts of **A** and **M** without seeing the other parts.
– Security: `YAES128` is a tweaked version of well-understood techniques: tweaked CTR-mode for encryption and a PMAC1 like construction as MAC. Note however that `YAES128` is not simply encrypt-then-MAC, since the ciphertext itself is not MAC-ed. We refer to Section 3 for security properties of `YAES128`.
– Public message number **N** is nonce: it is easy to see that `YAES128` (like other stream cipher based authenticated encryption modes where the stream only depends on the key **K** and **N**) loses all security when **N** is re-used with the same key **K** to encrypt another message. Since the same keystream will be generated, a simple XOR of the ciphertext leaks the XOR of the corresponding plaintexts.

– No decryption misuse resistance: it is easy to see that the plaintext produced by the decryption function of `YAES128` should not be used before the tag has been verified.
– Recommended parameters: for simplicity there is currently only one recommended parameter set: 128-bit key $\mathbf{K}$, 127-bit nonce $\mathbf{N}$, 128-bit tag. A 256-bit version can be easily derived in analogy with the 128-bit version: the 14 rounds of normal AES, would be split in 8 rounds for encryption and 6 rounds for authentication. Currently no statements are made on the resulting security bounds.

5 Design rationale

The design of `YAES128` is driven by the following set of requirements:

– Block cipher based, but only using the encryption function of the block cipher for both encryption and decryption.
– Fully parallelizable on the message block level. This rules out all the chaining modes of operation or designs that serially update state.
– Use of standard components that have been studied extensively and with readily available implementations including side channel protection.
– Performance comparable to or better than AES-OCB.
– Online and single pass.
– Small to medium sized state.
– No "full" finite field multiplications, since this adds another component that typically requires large area in hardware implementations.
– No requirements on nonce-misuse-resistance or decryption-misuse-resistance.

Taken the above requirements into account, the building blocks are limited to the AES round function and masking, for instance using the now classical Galois Field doubling approach. We remark however that other masking schemes would be possible such as the ones based on Gray codes or LFSRs.
The resulting mode can be viewed in various several ways. The first one is as a masked CTR-mode with reduced round AES for encryption, with a MAC based on the incremental hash function design instantiated with 4-round AES. Note that the difference with standard CTR mode is that both the input and output are masked. This implies that although the encryption itself is a reduced round AES, one actually never obtains a proper input/output pair. The mode is also somewhat related to OTR: given an input message M, derive a new message M' of double the length of M with $M'[2i] = M[i]$ and $M'[2i+1] = 0^n$ with n the block size in OTR. The ciphertext then simply corresponds to the concatenation of the left halves of the OTR output, whereas the right halves are used to compute the tag.
Finally we remark that "The designers have not hidden any weaknesses in this cipher".

6 Intellectual property

The designers/submitters of this proposal have note filed any patent applications nor have any intention to file future patent applications. As far as the designers/submitters

are aware, there are no intellectual-property constraints relevant to use of the cipher. If any of this information changes, the submitter/submitters will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

7 Consent

The submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitters understand that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

Acknowledgments

The designers would like to thank Elena Andreeva, Danny De Cock, Vincent Rijmen and Elmar Tischhauser for their insights into authenticated encryption schemes and Danilo Gligoroski for pointing out a discrepancy between the algorithms and their illustrations.

Appendix: Test Vectors

```
Key=     00000000000000000000000000000000
Adata=
Nonce=   09f911029d74e35bd84156c5635688c0
Plain=
Cipher=
Tag=     f62f040d7c7cc30bfcf449fb1fd6fb8c

Key=     00000000000000000000000000000000
Adata=   0011223344556677 8899
Nonce=   09f911029d74e35bd84156c5635688c0
Plain=   8e
Cipher=  c0
```

```
Tag=     163924ce11d81c2d373a86e6825e4efb

Key=     00000000000000000000000000000000
Adata=   00112233445566778899aabbccddeeff
Nonce=   09f911029d74e35bd84156c5635688c0
Plain=   00000000000000000000000000000000
Cipher=  4e23b0f5e59aaca555507f246ac4859e
Tag=     7d6f1549741a21776c219411349272a7

Key=     7f7e7d7c7b7a79787776757473727170
Adata=   00112233445566778899aabbccddeeff
Nonce=   09f911029d74e35bd84156c5635688c0
Plain=   7468697320697320736f6d6520706c61
Cipher=  63bb6fef9b3210aa760dd284c1b05e55
Tag=     92ca38160c25c09cabfd2cc4510a2861

Key=     7f7e7d7c7b7a79787776757473727170
Adata=   00112233445566778899aabbccddeeff ffeeddccbbaa99887766
Nonce=   09f911029d74e35bd84156c5635688c0
Plain=   7468697320697320736f6d6520706c61 696e7465787420746f20
Cipher=  63bb6fef9b3210aa760dd284c1b05e55 3961d827b9140ba5dec5
Tag=     83946736086461fd3caea2517aba89c4
```