mixFeed

Designers/Submitters:
Bishwajit Chakraborty - Indian Statistical Institute,Kolkata
Mridul Nandi - Indian Statistical Institute, Kolkata, India

bishu.math.ynwa@gmail.com
mridul.nandi@gmail.com

March 22, 2019

1 Introduction

In this document, we propose a new scheme for authenticated encryption with associated data (AEAD) based on AES'128/128 [6] block cipher. Here, we introduce a new mode which we call *Minimally Xored Feedback* mode (mixFeed) based on any block cipher with some involved key-scheduling algorithm. Our mode (on top of the n-bit block cipher) requires only n-bit xor to process each n-bit blocks. The name can also be justified for the fact that we use a mixture of Plaintext and Ciphertext as the feedback to the underlying blockcipher.

Another aspect of the mixFeed is that, we use nonce-dependent key. This would help to get higher security beyond conventional model (such as reasonable security against leakage of nonce-dependent key).

2 mixFeed Specification

2.1 Notations and Conventions

We fix positive even integers n, κ, and t to denote the *block size*, *key size*, and *tag size* respectively in bits. Our input nonce size is one byte less than the block size. $\mathsf{E}n/\kappa$ denotes a block cipher family E, parametrized by the block length n and key length κ. In this paper we use AES'128/128 and so $n = 128$, nonce size $= 120$ and $\kappa = 128$. Note that AES'128/128 is same as the original AES128/128 except that we use mixcolumn operation at the last round.

We fix the tag size to be 128. Note that one can always truncate the tag to a small size if required.

We use $\{0,1\}^+$ and $\{0,1\}^n$ to denote the set of all non-empty (binary) strings, and n-bit strings, respectively. λ denotes the empty string and $\{0,1\}^* = \{0,1\}^+ \cup \{\lambda\}$. For all practical purposes: we use little-endian format of indexing, and assume all binary strings are *byte-oriented*, i.e. belong in $(\{0,1\}^8)^*$. For any string $B \in \{0,1\}^+$, $|B|$ denotes the number of bits in B, and for $0 \le i \le |B|-1$, b_i denotes the i-th bit of B, i.e. $B = b_{|B|-1} \cdots b_0$. where b_0 is the least significant bit (LSB) and $b_{|B|-1}$ is the most significant bit (MSB). Given a nonempty bit string B of size $x < n$, we denote $\mathsf{pad}(B)$ as $0^{n-x-1}1B$. Thus we always pad the extra bits from MSB side. When $x = n$, we define $\mathsf{pad}(B)$ as B itself. The chop function chops either the most significant or least significant bits. For $k \le n$, and $B \in \{0,1\}^n$, $\lfloor B \rfloor_k := B_{k-1} \ldots B_0$ and $\lceil B \rceil_k := B_{n-1} \ldots B_{n-k}$.

For $B \in \{0,1\}^+$, $(B_{\ell-1}, \ldots, B_0) \xleftarrow{n} B$, denotes the n-bit *block parsing* of B into $(B_{\ell-1}, \ldots, B_0)$, where $|B_i| = n$ for $0 \le i \le \ell-2$, and $1 \le |B_{\ell-1}| \le n$. For $A, B \in \{0,1\}^+$, and $|A| = |B|$, $A \oplus B$ denotes the "bitwise XOR" operation on A and B. For $A, B \in \{0,1\}^+$, $A \| B$ denotes the "string concatenation" operation on A and B.

We will use a compact representation of if-else statement by the following expression $P \ ? \ b \ : \ c$ where P is some mathematical statement. This evaluates to b if P is true and c otherwise. P_1 & P_2 ? $b_1 : b_2 : b_3 : b_4$ evaluates to b_1 if both P_1 and P_2 are true, to b_2 if only P_1 is true, to b_3 if only P_2 is true and to b_4 if none of P_1 , P_2 are true.

Block Cipher: A block cipher with key size κ and block size n is a family of permutations over n-bits indexed by κ bit key. For a fixed key $k \in \{0,1\}^\kappa$, we write $E_k(\cdot) = E(k, \cdot)$. Many block cipher uses some non-trivial key-scheduling algorithm which produces round keys for each round to mask the block cipher state. Let ϕ corresponds to the function which updates the key. In other words, if K is the key of the block cipher for the current execution, $\phi(K)$ will denote the updated key. We will see details of this key update function for AES'128/128 in more details later.

2.2 Our Recommendation

In Algorithm 1 we describe our specification mixFeed based on any block cipher E. We propose (primary submission) mixFeed where E is instantiated by AES'128/128 where the last round also calls MixColumns operation of AES128/128. For the sake of completeness we describe it in Algorithm 2. This does not change any security level of AES'128/128, but it adds uniformity over all rounds.

2.3 Provenance of Constants used in Tweak Control

Our mode uses a 4-bit constant $t_3 \| t_2 \| t_1 \| t_0$ for processing the last block of associated data and the last block of message which distinguishes different cases regarding completeness of the last blocks. This constant value is decided from the inputs of the hardware API and are explained as follows.

- eoi : t_3 is called the end of input control bit. This bit is set to 1 if and only if the current data block being processed is the final block of the input. For all other data block processing t_3 is set to 0.

Algorithm 1 Encryption/Decryption algorithm in mixFeed. Here, λ denotes the empty string. $\perp, \top$ denotes the abort and accept symbols respectively. By $*$, we mean that the exact value is not bothered.

function $\textsc{mixFeed}_{[\mathsf{E}]}.\mathsf{enc}(K, N, A, M)$
  $((a, \delta_A), (m, \delta_M)) \leftarrow \mathsf{Fmt}(A, M)$
  if $a = 0, m = 0$ **then**
    $(T, *) \leftarrow \mathsf{E}_K(N \| 0^6 10)$
    return (λ, T)
  else if $a = 0$ **then** $(K_N, *) \leftarrow \mathsf{E}_K(N \| 0^7 1)$
  else $(K_N, *) \leftarrow \mathsf{E}_K(N \| 0^8)$
  $(Y, K) \leftarrow \mathsf{E}_{K_N}(N \| 0^8)$
  $C \leftarrow \lambda$
  if $a \neq 0$ **then** $(*, Y, K) \leftarrow \mathsf{proc\_txt}(Y, K, A, \delta_A, +)$
  if $m \neq 0$ **then** $(C, T, *) \leftarrow \mathsf{proc\_txt}(Y, K, M, \delta_M, +)$
  return (C, T)

function $\textsc{mixFeed}_{[\mathsf{E}]}.\mathsf{dec}(K, N, A, C, T)$
  $((a, \delta_A), (m, \delta_C)) \leftarrow \mathsf{Fmt}(A, C)$
  if $a = 0, m = 0$ **then**
    $(T', *) \leftarrow \mathsf{E}_K(N \| 0^6 10)$
    return $(T = T')?\ \top : \perp$
  else if $a = 0$ **then** $(K_N, *) \leftarrow \mathsf{E}_K(N \| 0^7 1)$
  else $(K_N, *) \leftarrow \mathsf{E}_K(N \| 0^8)$
  $(Y, K) \leftarrow \mathsf{E}_{K_N}(N \| 0^8)$
  $M \leftarrow \lambda$
  if $a \neq 0$ **then** $(*, T, K) \leftarrow \mathsf{proc\_txt}(Y, K, A, \delta_A, +)$
  if $m \neq 0$ **then** $(M, T', *) \leftarrow \mathsf{proc\_txt}(T, K, C, \delta_C, -)$
  if $T \neq T'$ **then**
    return $\perp$
  else
    return $(M, \top)$

function $\mathsf{Fmt}(A, M)$
  $(A_{a-1}, \ldots, A_0) \xleftarrow{n} A$
  $(M_{m-1}, \ldots, M_0) \xleftarrow{n} M$
  $\delta_A \leftarrow (n \mid |A_{a-1}|)\ \&\ (m = 0)?\ 12 : 4 : 14 : 6$
  $\delta_M \leftarrow (m \mid |M_{m-1}|)?\ 13 : 15$
  return $((a, \delta_A), (m, \delta_M))$

function $\mathsf{proc\_txt}(K_1, Y_0, D, \delta_D)$
  $(D_{d-1}, \ldots, D_0) \xleftarrow{n} D$
  for $i = 0$ **to** $d - 1$ **do**
    $(X_{i+1}, D'_i) \leftarrow \mathsf{Feed}(Y_i, D_i, +)$
    $(Y_{i+1}, K_{i+2}) \leftarrow \mathsf{E}_{K_{i+1}}(X_{i+1})$
  $X_{d+1} \leftarrow Y_d \oplus 0^{n-4} \| \delta_D$
  $(Y_{d+1}, K_{d+2}) \leftarrow \mathsf{E}_{K_{d+1}}(X_{d+1})$
  return (D', Y_{d+1}, K_{d+2})

function $\mathsf{Feed}(Y, D, \mathrm{dir})$
  $D' \leftarrow D \oplus \lfloor Y \rfloor_{|D|}$
  if $\mathrm{dir} = "+"$ **then**
    $B \leftarrow \lceil \mathsf{pad}(D') \rceil_{n/2} \| \lfloor \mathsf{pad}(D) \rfloor_{n/2}$
  if $\mathrm{dir} = "-"$ **then**
    $B \leftarrow \lceil \mathsf{pad}(D) \rceil_{n/2} \| \lfloor \mathsf{pad}(D') \rfloor_{n/2}$
  $X \leftarrow B \oplus Y$
  return (X, D')

- eot: t_2 is called the end of type control bit. This bit is set to 1 if and only if the current data block being processed is the last block of the same type i.e. it is the last block of message/ associated data. For all other data block processing t_2 is set to 0.

- partial: t_1 is called the partial control bit. this bit is set to 1 if data block currently being processed is a partial block, i.e. it's the data size is less than the required block size. For all other data block processes it is set to 0.

- Type: t_0 is called the type control bit and it identifies the data being processed. For the final message block processing, t_0 is set to 1. For all other data processing, t_0 is set to 0.

While processing a last data block of a type, the input of the block cipher is decided based on the 4 control bits. Fmt function outputs the δ_A, δ_M values by simply giving the integer representation of $t_3 \| t_2 \| t_1 \| t_0$. For example if we are in the last message block and it is partial then $t_3 = 1, t_2 = 1, t_1 = 1, t_0 = 1$, making $\delta_M = 15$ In Algorithm 1. Similarly if we are processing the last associate data block which is complete and the message length is non-zero, then $t_3 = 0, t_2 = 1, t_1 = 0, t_0 = 0$ making $\delta_M = 4$.

i	1	2	3	4	5	6	7	8	9	10	11
RCON(i)	01	02	04	08	10	20	40	80	1b	36	6c

Table 1: The RCON Values

3 Security of mixFeed

Here we describe some possible strategies to attack the mixFeed mode, and give a rough estimate on the amount of data and time required to mount those attacks (see Table 2). In the following discussion:

Algorithm 2 AES'128/128 Block Cipher. To apply a chain of block cipher, we perform an extra round of AES'128/128 Key-Schedule and use that round key as the initial key of the next call of AES'128/128. As described in the Introduction the second output of Emodule only depends on the first input K and we define this function as $\phi(K)$.

function $\mathsf{E}(K; X)$
  $(W_{47}, \ldots, W_0) \leftarrow \mathsf{KeyGen}(K)$
  for $i = 1$ **to** 10 **do**
    $X \leftarrow X \oplus (W_{4i-1}, W_{4i-2}, W_{4i-3}, W_{4i-4})$
    $X \leftarrow \mathsf{SubBytes}(X)$
    $X \leftarrow \mathsf{ShiftRows}(X)$
    $X \leftarrow \mathsf{MixColumns}(X)$
  $X \leftarrow X \oplus (W_{43}, W_{42}, W_{41}, W_{40})$
  $K \leftarrow (W_{47}, W_{46}, W_{45}, W_{44})$
  return (X, K)

function $\mathsf{KeyGen}(K)$
  $(K_{15}, \ldots, K_0) \xleftarrow{8} K$
  for $i = 0$ **to** 3 **do**
    $W_i \leftarrow (K_{4i+3}, K_{4i+2}, K_{4i+1}, K_{4i})$
  for $i = 4$ **to** 47 **do**
    $Y \leftarrow W_{i-1}$
    if $i\%4 = 0$ **then**
      $Y \leftarrow \mathsf{SubWords}(Y \lll 8)$
      $Y \leftarrow Y \oplus \mathsf{RCON}_{i/4}$
    $W_i \leftarrow W_{i-4} \oplus Y$
  return $(W_{47}, \ldots, W_0)$

function $\mathsf{SubBytes}(X)$
  $(X_{15}, \ldots, X_0) \xleftarrow{8} X$
  for $i = 0$ **to** 15 **do**
    $X_i \leftarrow \mathrm{AS}(X_i)$
  return X

function $\mathsf{Shiftrows}(X)$
  $(X_{15}, \ldots, X_0) \xleftarrow{8} X$
  for $i = 0$ **to** 3 **do**
    for $j = 0$ **to** 3 **do**
      $Y_{4i+j} \leftarrow X_{4i+((j+i)\%4)}$
  return Y

function $\mathsf{MixColumns}(X)$
  $M \leftarrow \begin{pmatrix} 2 & 3 & 1 & 1 \\ 3 & 1 & 1 & 2 \\ 1 & 1 & 2 & 3 \\ 1 & 2 & 3 & 1 \end{pmatrix}$
  $Y \leftarrow M \cdot X$
  return Y

- D denotes the data complexity of the attack. This parameter quantifies the online resource requirements, and includes the total number of blocks (among all messages and associated data) processed through the underlying block cipher for a fixed master key. Note that for simplicity we also use D to denote the data complexity of forging attempts.

- T denotes the time complexity of the attack. This parameter quantifies the offline resource requirements, and includes the total time required to process the off line evaluations of the underlying block cipher. Since one call of the block cipher can be assumed to take a constant amount of time, we generally take T as the total number of off line calls to the block cipher.

Security Model	Data complexity ($\log_2 D$)	Time complexity ($\log_2 T$)
IND-CPA	60	112
INT-CTXT	50	112

Table 2: Security Claims. We remark that the given values indicate the amount of data and time required to make the attack advantage close to 1.

Notes on Security on the Modes After making q queries with σ many blocks, adversary observes inputs and outputs of the block cipher with a key which is dependent on the nonce and the current block number. Thus the security of this construction would depend on the nultikey set up. As the least significant 64 bits of inputs are random (during encryption), the multi-key attack (in the ideal cipher model) will have advantage roughly $\sigma T/2^{192}$ where T is the number of ideal cipher calls and σ is the number of encryption blocks. Similar argument will work for all decryption attempts.

We must admit that there is no conventional privacy security in case of nonce misuse. However, the integrity security remains until 2^{32} data in case of nonce misuse.

3.1 Known Security Analysis of AES'128/128

The security of AES'128/128 is same as the security of AES128/128 as mixcolumn is a linear operation which can be peeled off from the output. The security of AES128/128 is well-established in the community.

To the best of our knowledge, the best single-key attack on AES128/128 is the biclique attack by Bogdanov et al. [1], that recovers the key in approx. 2^{126} computations. Although there is a related-key attack on full-round AES-128/192 and AES-128/256, the same attack does not apply to AES128/128, even in the usual XOR related-key setting, let alone the key scheduled related-keys. In fact, [5] shows that AES128/128 is almost as secure in related-key setting as it is in single-key setting. Recent distinguishers on AES128/128 [3, 4, 7, 2], are applicable to round-reduced variants of AES128/128, and hence not applicable in our case.

4 Design Rationale

4.1 Choice of the Mode

Our primary goal is to design a lightweight cipher that should be efficient, provide high performance and able to perform well in low end devices. In addition, we also demand robustness in security.

4.1.1 Nonce dependent key

At the very first step we compute the secret key based on nonce. So, for every encryption we use random keys. Even though due to some side channel analysis the secret key corresponding to a nonce N is released, the master key remains still secret and all encryption using nonce other than N remains good.

4.1.2 Minimally xored mixture feedback

As our name suggests, we use minimum number of xors to process each block. This makes the design simpler and having very low footprint in software. The rational behind having mixture of plaintext and ciphertext feedback is to achieve NIST aimed security. During encryption we ensure 192 bit entropy for each block process. We have 128 bit dynamic secret key and 64 bits LSB of the inputs have influence from 64 bits LSB of the previous block cipher call.

While decrypt, we have 64 bit MSB of the previous outputs goes to the correspond position of the next input. This would provide about 64 bit security for forgery attempts.

4.1.3 Single State

mixFeed has a state size as small as the block size of the underlying cipher, and it ensures good implementation characteristics both on lightweight and high-performance platforms. We moreover need not to hold the original key as we dynamically update the key based on the key scheduling algorithm used for the block cipher computation.

4.1.4 Inverse-Free

mixFeed is a inverse-free authenticated algorithm. Both encryption and verified decryption of the algorithm do not require any decryption call to the underlying twekable block cipher. This reduces the overall hardware footprint significantly, especially in the combined authenticated-encryption, verified-decryption implementations.

4.2 Choice of the Block cipher

4.2.1 Well analyzed and NIST standard

AES128/128 block cipher is well analyzed for long time and it remains secure. Moreover, in this proposal, a weaker security from AES128/128 would suffice. AES128/128 also performs very well in microcontroller based platform. We note that the last mix-column operation is included in our proposal to make it uniform over all rounds. This reduces additional MUX which was required to process last round for the original AES128/128.

4.2.2 Dynamic Key

We compute the key dynamically as key schedules goes on. This helps us not to hold the master key as well not to expose a secret key multiple times. As the key-scheduling of AES128/128 is involved, the related-key security analysis of AES128/128 expected to be much harder than conventional xor-related key.

References

[1] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 344–371, 2011.

[2] Lorenzo Grassi. Mixture differential cryptanalysis: a new approach to distinguishers and attacks on round-reduced AES. *IACR Trans. Symmetric Cryptol.*, 2018(2):133–160, 2018.

[3] Lorenzo Grassi, Christian Rechberger, and Sondre Rønjom. Subspace trail cryptanalysis and its applications to AES. *IACR Trans. Symmetric Cryptol.*, 2016(2):192–225, 2016.

[4] Lorenzo Grassi, Christian Rechberger, and Sondre Rønjom. A new structural-differential property of 5-round AES. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, pages 289–317, 2017.

[5] Khoongming Khoo, Eugene Lee, Thomas Peyrin, and Siang Meng Sim. Human-readable proof of the related-key security of AES-128. *IACR Trans. Symmetric Cryptol.*, 2017(2):59–83, 2017.

[6] NIST. Announcing the ADVANCED ENCRYPTION STANDARD (AES). Fedral Information Processing Standards Publication FIPS 197, National Institute of Standards and Technology, U. S. Department of Commerce, 2001.

[7] Sondre Rønjom, Navid Ghaedi Bardeh, and Tor Helleseth. Yoyo tricks with AES. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, pages 217–243, 2017.

Appendix

Test Vectors

Test Vectors for **AES'128/128**

testvector 1
$Input = 00123456789abcde$
$Key = efcb089475ded60586a7d97c64baf$
$Input = eb2af160413cc3b7b883c03017809ea9$
$Key = 60e85eca556be71e2b61bd666465c495$

testvector 2
$Input = 00123456789abcde$
$Key = efcb089475ded60586a7d97c64baf$
$Input = eb2af160413cc3b7b883c03017809ea9$
$Key = 60e85eca556be71e2b61bd666465c495$

Test Vectors for **mixFeed**

testvector 1
$Key = 000102030405060708090A0B0C0D0E0F$
$Nonce = 000102030405060708090A0B0C0D0E$
$PT =$
$AD = 000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F$
$CT = 6CDB385142B591F8E57D50FC41899B23$

testvector 2
$Key = 000102030405060708090A0B0C0D0E0F$
$Nonce = 000102030405060708090A0B0C0D0E$
$PT = 00$

$AD = 000102030405060708090A0B0C0D$
$CT = E56EDEC0001E1D94074303E6397D238CCF$

testvectors 3
$Key = 000102030405060708090A0B0C0D0E0F$
$Nonce = 000102030405060708090A0B0C0D0E$
$PT = 000102$
$AD = 000102030405060708090A0B0C0D0E$
$CT = 4753140EA6C5D3B01F06BBBC3F55181BB3FFE5$