JAMBU Lightweight Authenticated Encryption Mode and AES-JAMBU (v1)

15 March, 2014

Designers: Hongjun Wu, Tao Huang

Submitters: Hongjun Wu, Tao Huang

Contact: `wuhongjun@gmail.com`

Division of Mathematical Sciences
Nanyang Technological University, Singapore

Table of Contents

1 Introduction

Authenticated encryption mode is one of the commonly used method in the design of authenticated ciphers. The ISO/IEC 19772:2009 [7] standardized several modes for authenticated encryption, including EAX [1], CCM [19], GCM [15] and OCB 2.0 [18]. And a number of other authenticated encryption modes have been proposed, *e.g.,* IAPM [10], CWC [12], HBS [9], BTM [8] and McOE [6].

An important trend in the current development of cryptography is to design lightweight cryptographic primitives since the increasing needs for low-cost embedded systems such as RFID tags, sensor networks and smart cards. Recently, several authenticated encryption schemes have been proposed for the lightweight usage, such as Hummingbird-2 [5], ALE [3], and FIDES [2].

However, all the above mentioned lightweight authenticated encryption schemes are dedicated design and can not be used as a mode of operation to convert an encryption scheme into an authenticated cipher. Moreover, it turns out that it is quite difficult to construct a secure lightweight authenticated cipher. Security flaws were discovered for ALE and FIDES shortly after their publications [4, 11, 20]. Hence, it is meaningful to develop a secure lightweight authenticated encryption mode so that the previous designs of lightweight block ciphers can be converted to lightweight authenticated ciphers.

In this document, we propose a lightweight authenticated encryption mode JAMBU and use AES-128 as the underlying block cipher to construct an authenticated cipher − AES-JAMBU.

2 The JAMBU Mode of Operation

2.1 Preliminary

2.1.1 Operations

The following operations are used in JAMBU:

$\oplus$: bit-wise exclusive OR.
$||$: concatenation.

2.1.2 Notations and Constants

The following notations are used in JAMBU specifications.

0^a	: a bit of '0's.
AD	: associated data (this data will not be encrypted or decrypted).
$adlen$	: bit length of the associated data with $0 \leq adlen < 2^{64}$.
C	: ciphertext.
C_i	: a ciphertext block (the last block may be a partial block).
E_K	: encryption of one block using the secret key K.
IV	: initialization vector used in JAMBU.
K	: secret key used in JAMBU.
$msglen$	: bit length of the plaintext/ciphertext with $0 \leq msglen < 2^{64}$.
m_i	: a data block.
n	: half of the block size used in JAMBU.
N	: number of the associated data blocks and plaintext blocks after padding. $N = N_A + N_P$
N_A	: number of the associated data blocks after padding.
N_P	: number of the plaintext blocks after padding.
P	: plaintext.
P_i	: a plaintext block (the last block may be a partial block).
R	: an additional state used for encryption. The size is half of the block size.
S	: an internal state which will be used for encryption.
T	: authentication tag.
t	: bit length of the authentication tag

2.2 Padding

The following padding scheme is used in JAMBU . For associated data, a '1' bit is padded followed by the least number of '0' bits to make the length of padded associated data a multiple of n-bit. Then the same padding method is applied to the plaintext.

2.3 Initialization

JAMBU uses an n-bit initialization vector(IV). The initialization vector (public message number) is public. And each key/IV pair should be used only once to achieve the maximum security of the scheme.

Let (X, Y) represent the composition of n-bit states X and Y which results in a state of $2n$-bit. The initial state is set as $S_{-1} = (0^n, IV)$. The following operations are used for initialization.

1. $(X_{-1}, Y_{-1}) = E_K(S_{-1})$;
2. $R_0 = X_{-1}$;
3. $S_0 = (X_{-1}, Y_{-1} \oplus 5)$.

The initialization of JAMBU is shown in Fig. 1.

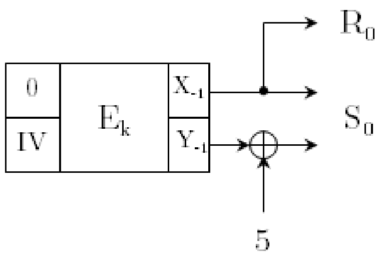

Fig. 1: Initialization of JAMBU .

2.4 Processing the associated data

The associated data is divided into n-bit blocks and processed sequentially. For the last block, the padding scheme is applied to make it a full block. Note that at least one block is processed in the processing of AD. Namely, if the length of AD, *adlen*, is 0, a padded block $1 \parallel 0^{n-1}$ will be processed. Let N_A be the number of AD blocks after padding, the AD is processed as follows.

- For $i = 0$ to $N_A - 1$, we update the states:

$$\begin{aligned} (X_i, Y_i) &= E_K(S_i); \\ U_{i+1} &= X_i \oplus A_i; \\ V_{i+1} &= Y_i \oplus R_i \oplus 1; \\ S_{i+1} &= (U_{i+1}, V_{i+1}); \\ R_{i+1} &= R_i \oplus U_{i+1}. \end{aligned}$$

Fig. 2 shows the processing of two blocks of associated data.

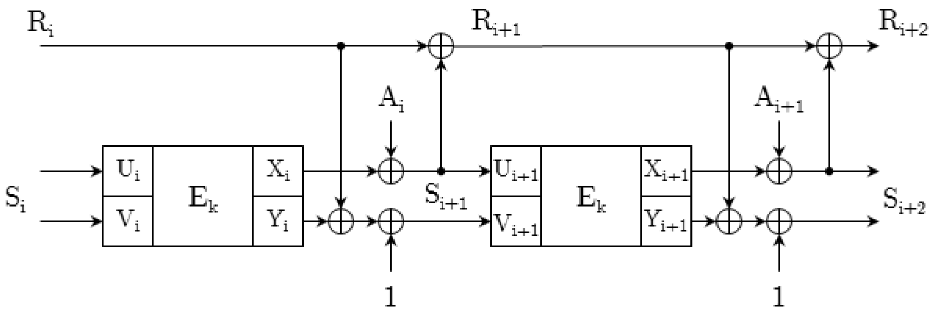

Fig. 2: Processing associated data.

2.5 Encryption of JAMBU

In the encryption of JAMBU, the plaintext is divided into blocks of n-bit. And the last block is padded using the padding scheme specified previously. In each step of the encryption, a plaintext block P_i is encrypted to a ciphertext block C_i.

If the last plaintext block is a full block, a block of "$1||0^{n-1}$" is processed without any output. Fig. 3 shows the encryption of two plaintext blocks.

Let N_P be the number of plaint blocks after padding, the encryption is described as follows:

- For $i = N_A$ to $N_A + N_P - 1$, we perform encryption and update the state:
  $(X_i, Y_i) = E_K(S_i)$;
  $U_{i+1} \quad = X_i \oplus P_{i-N_A}$;
  $V_{i+1} \quad = Y_i \oplus R_i$;
  $S_{i+1} \quad = (U_{i+1}, V_{i+1})$;
  $R_{i+1} \quad = R_i \oplus U_{i+1}$.
  $C_{i-N_A} \quad = P_{i-N_A} \oplus V_{i+1}$ if $i < N_A + N_P - 1$ or the last plaintext block is a partial block; otherwise, C_{N_P-1} will not be computed.
- The final ciphertext block is truncated to the actual length of last plaintext block from the most significant bit side.

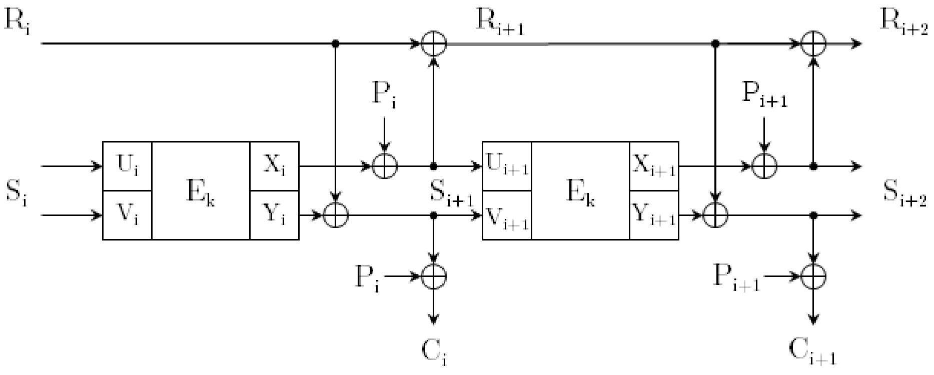

Fig. 3: Processing the plaintext.

2.6 Finalization and tag generation

After all the padded plaintext blocks are processed, suppose the state is S_{N+1} and R_{N+1} ($N = N_A + N_P - 1$), we use following steps to generate the authentication tag, see Fig. 4.

1. $(X_{N+1}, Y_{N+1}) = E_K(S_{N+1})$;
2. $U_{N+2} = X_{N+1}$;
3. $V_{N+2} = Y_{N+1} \oplus R_{N+1} \oplus 3$;
4. $R_{N+2} = R_{N+1} \oplus X_{N+1}$;
5. $S_{N+2} = (X_{N+2}, Y_{N+2})$;
6. Authentication tag is generated as $T = R_{N+2} \oplus X_{N+2} \oplus Y_{N+2}$.

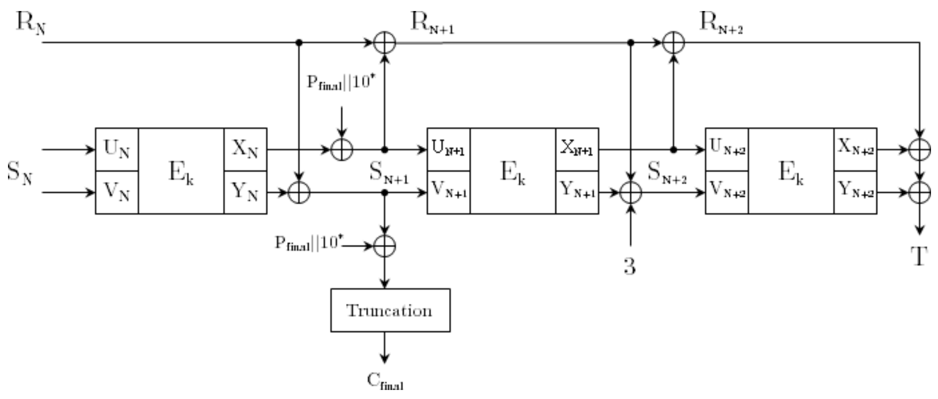

Fig. 4: Finalization and tag generation.

2.7 The decryption and verification

The decryption and verification are similar to the encryption and authentication, except that the ciphertext block is XORed with the sub-state V to compute the plaintext block. For the final block, the ciphertext is padded using the same scheme as the plaintext before the XOR operation. If the length of ciphertext block is a multiple of n, another block of "$1||0^{n-1}$" is processed similar as in the encryption.

A tag T' is generated after the decryption and is compared to the tag T. If the two tags match, the plaintext is outputted.

3 The AES-JAMBU authenticated cipher

In this section, we will apply the JAMBU mode to the most widely used block cipher AES-128 and construct an authenticated cipher AES-JAMBU.

Recommended parameter set In AES-JAMBU, we recommend specific values for the parameters used in JAMBU. The block size is 128-bit and the size of n is 64-bit. The tag length t is 64-bit and the maximum number of message bits protected by one key is 2^{64} bits.

The parameters of AES-JAMBU are summarized in Table 1.

4 Security Goals

The security goals of ASE-JAMBU are given in Table 2. There is no secret message number. The public message number is a nonce. To achieve the maximum security, each key and IV pair should be used to protect only one message. If verification fails, the new tag and the decrypted ciphertext should not be given as output.

Table 1: Parameter set of AES-JAMBU.

Parameters	Length in bits
Plaintext (P)+Associated data (AD)	$< 2^{64}$
Key (K)	128
Tag (T)	64
Initialization vector (IV)	64
State (S)	320[a]

[a] 128-bit block and 128-bit key are used in AES-128. An additional 64-bit state is used in AES-JAMBU.

However, in case that the IV is reused under the same key, the confidentiality of AES-JAMBU is only partially compromised as it only leaks the information of the first block or the common prefix of the message. And the integrity of AES-JAMBU will be less secure but not completely compromised.

Notice that the integrity security in Table 2 includes the integrity security of plaintext, associated data and nonce and under the assumption that the secret key is unknown to the attacker, and 64-bit tag is used. The table assumes that the total length of message (plaintext and associated data) protected by a single key is limited to 2^{64} bits.

Table 2: Security Goals of AES-JAMBU.

	Confidentiality (bits)	Integrity (bits)
AES-JAMBU	128	64

5 The Security Analysis of AES-JAMBU

In this section, we analyze the security of AES-JAMBU which includes the security of encryption and the security of authentication.

5.1 The security of encryption

The encryption of JAMBU can bee seen as a variant of the Output Feedback (OFB) mode from the NIST recommendation [16]. The only difference is that in JAMBU the message block and an additional state block R are XORed with the internal state while in OFB the message is not involved in the state update. As AES is used as its underlying block cipher, the encryption of AES-JAMBU is expected to be as strong as the AES encryption as long as each key/IV is used to protect only one message. Thus, the plaint text block and the additional state will not affect the randomness of the output of AES-OFB, and the confidentiality of AES-JABMU can be implied from AES-OFB.

Note that the security of OFB encryption will be compromised if IV is reused. But in JAMBU, the case is a bit similar to that in CBC/CFB: partial information will be leaked for the first block and the common prefix of two messages.

5.2 The security of message authentication

In this section, we will give our initial analysis on message authentication of AES-JAMBU for the case that the nonce is used only once under a single key and the decrypted plaintext will not be given when the verification fails. The case that the nonce is reused will be discussed in later versions of this document.

To analyze the security the message authentication of JAMBU, we consider two attacks on MAC: the key/state recovery attack and the internal collision attack.

5.2.1 Key/state recovery

Since the secret key of is protected by full AES, AES-JAMBU is not vulnerable to the key recovery attack. And for state recovery attack, AES-JAMBU also has a strong resistance as there are 128-bit unknown state at each step and the secret key is used in the encryption of each step.

5.2.2 Internal collisions

Constructing an internal collision is a commonly used method in the attacks against MACs. A general approach to construct an internal collision is from the birthday attack. This was discussed by Preneel and van Oorschot [17] which showed that all iterated MACs with n-bit internal state can be attacked with $O(2^{n/2})$ queries. For AES-JAMBU, the internal state size is 192 bits. Thus, around 2^{96} messages are needed for an internal collision using birthday attack. When the number of blocks encrypted is 2^{58} (the maximum value defined in the specification), there are around 2^{115} pairs of internal states. So the collision probability is around $2^{-192+115} = 2^{-77}$ which is less than 2^{-64}.

Now we analyze conditions for an internal collision. Suppose we have the first internal collision such that $(R_{i+1}, U_{i+1}, V_{i+1}) = (R_{j+1}, U_{j+1}, V_{j+1})$. We have $R_{i+1} = R_i \oplus U_{i+1}$, $U_{i+1} = X_i \oplus P_i$, and $V_{i+1} = Y_i \oplus R_i$, and the similar expressions holds for the states at step $j+1$. Hence, we can derive the following necessary conditions for the internal collision:

$$Y_i = Y_j \tag{1}$$

$$R_i = R_j \tag{2}$$

$$X_i \oplus X_j = P_i \oplus P_j \tag{3}$$

Note that only the condition (1) and (2) implies $V_{i+1} = V_{j+1}$ which can be observed from the keystream block. By generating around 2^{32} blocks of keystream, we are expected to observe one collision on the keystream block. But this collision is only a necessary condition, and we still need either (1) or (2) holds.

For condition (1), Y_i and Y_j are the output of AES encryption. Without the collision on the input (U_i, V_i) and (U_j, V_j), $Y_i = Y_j$ has probability 2^{-64}. On the other hand, if there is collision on (U_i, V_i) and (U_j, V_j), we get an internal collision on previous state (R_i, U_i, V_i) and (R_j, U_j, V_j) which violates our assumption.

For condition (2), both R_i and R_j are unknown and during the encryption. So the condition (2) can only be satisfied probabilistically, which is 2^{-64}.

For condition (3), when there is no difference on X_i and X_j, it will lead to a collision in previous internal state, which is assumed impossible. And if there is difference, the difference is uncontrollable by an attacker. Hence, the probability for condition (3) is 2^{-64}.

Given the above analysis, the probability for internal collision given the collision on keystream is 2^{-128}. Therefore, even if we can detect 2^{51} collisions on keystream blocks, the probability that an internal collision occurs is less than 2^{-64}.

Another approach is to construct an internal collision by injecting and canceling difference in the state in the decryption and verification process. This technique is particularly useful in attack the authenticated encryption schemes. In AES-JAMBU, if we inject a difference at ciphertext C_i, it will be passed to R_{i+1} and U_{i+1}. After encryption, the difference will propagate to X_{i+1} and Y_{i+1}. Since the difference on R_{i+1}, it is impossible to cancel the difference at the state $(R_{i+2}, U_{i+2}, V_{i+2})$. On the other hand, the difference on $R_{i+2} = R_{i+1} \oplus X_{i+1} \oplus P_{i+1}$ becomes unknown since the difference on X_{i+1} is unknown. Then, at each step, the probability that there is no difference on R and Y is probabilistic, which is 2^{-128}. Hence, even if the difference on X is canceled by modify ciphertext, the probability for a successful forgery is less than 2^{-64}.

Therefore, the AES-JAMBU is strong against the attacks on message authentication.

6 Features

- Lightweight. In addition to the registers used in the underlying block cipher, the JAMBU authenticated encryption mode only requires one additional register with half of the block size. For AES-GCM, two additional registers[1] are needed and each has equal length as the block size. And for fast implementation of GCM operations, a look up table is very helpful. However, when the table is used, a much larger amount of memory will be needed. It makes AES-GCM not suitable for lightweight implementations.
- Partial resistance against IV reuse. When the IV is accidentally reused under the same key, the security of encryption and authentication is not completely

[1] The two registers are used to: store the length of P and AD; store the chaining value for authentication.

compromised. Notice that in AES-GCM, the nonce reuse will lead to the lost of all confidentially and integrity.

7 Performance

7.1 Hardware performance

The core feature of JAMBU is hardware-oriented. In the hardware implementation of authenticated ciphers, the state size is an important factor, especially for low-cost embedded systems. To compare the hardware efficiency of the authenticated encryption modes on the area, We look at the state size when an authenticated encryption mode is applied to a $2n$-bit block cipher. We compare the state size in JAMBU with the existing authentication modes, and the results are given in Table 4. As a lightweight authenticated encryption mode, JAMBU provides the minimum state size for the hardware implementation.

Table 3: The comparison (in state size) for authenticated encryption modes, assuming the underlying block cipher has block size $2n$ bits

Modes	State size	Increments
CCM	$4n$	$2n$
GCM	$6n$	$4n$
OCB3	$6n$	$4n$
EAX	$8n$	$6n$
JAMBU	$3n$	n

7.2 Software performance

We implemented AES-JAMBU in C code using the AES instruction. We tested the speed on the Intel Core i5-2540M 2.6GHz processor (Sandy Bridge) running 64-bit Linux 12.01. The turbo boost is turned off, so the CPU runs at 2.6GHz in the experiment. The compiler being used is gcc 4.5.2, and the options "-O3 -msse2 -maes -mavx" are used. The test is performed by encrypting/decrypting a message repeatedly, and printing out the final message. To ensure that the tag generation is not removed during the compiler optimization process, we use the tag as the IV for processing the next message. To ensure that the tag verification is not removed during the compiler optimization process, we sum up the number of failed verifications and print out the final result.

We tested the speed of CTR, OCB3, GCM and CCM (AES-128 is used in these modes) on the same machine for comparison. The testing programs of CTR, OCB3, GCM and CCM are downloaded following the description given Krovetz and Rogaway in the OCB3 paper [14] and their website [13]. The performance comparison is given in Table 3. For 4096-byte messages, the speed of AES-JAMBU is about 11.6 cpb. Since in AES-JAMBU, each AES encryption is used to process only 8 byte of message, we expect that the speed of AES-JAMBU will be about two times slower than AES-GCM. The speed turns out to be about four times slower in our experiment. The reason may be largely due to some operations in AES-JAMBU are not optimized in our current implementation.

Table 4: The software speed comparison (in cycles per byte) for different message length on Intel Sandy Bridge. A plus sign (+) indicates that the data are from the ALE designers and the performance is measured on Intel i5-2400 microprocessor.

	64B	128B	256B	512B	1024B	4096B
AES-128-CTR$^+$	–	1.61	1.22	0.99	0.87	0.77
AES-128-CCM	7.26	6.31	5.65	5.19	5.17	5.05
AES-128-GCM$^+$	–	4.95	3.88	3.33	3.05	2.90
AES-128-OCB3$^+$	–	2.69	1.79	1.34	1.12	0.88
AES-JAMBU	17.7	14.54	13.05	12.27	11.86	11.60

8 Design rationale

JAMBU is designed to be a lightweight authenticated encryption mode which can offer partial resistance against IV reuse.

To make this mode lightweight, we introduces only an n-bit extra register for a $2n$-bit block size. And we only use the bit-wise XOR operations in the JAMBU mode.

The padding scheme used in JAMBU does not require the length information to be stored in a register. This reduces the memory requirements.

To offer a certain level of security against IV reuse, we use a block cipher encryption in the state update and only half of the state is leaked after encryption. The plaintext is injected into the other half of the state which is unknown for the attacker.

Several constants are XORed with the state in JAMBU. They are used to separate the initialization, associate data processing, plaintext processing, and finalization.

AES-JAMBU uses AES as the underlying block cipher. It can take advantages from the security analysis AES as well as the fast implementation of AES using AES-NI.

The designers have not hidden any weaknesses in this cipher.

9 Intellectual property

JAMBU is not patented and it is free of intellectual property restrictions. 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.

10 Consent

The submitter/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 submitter/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 submitter/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 submitter/submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitter/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 submitter/submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

References

1. M. Bellare, P. Rogaway, and D. Wagner. The EAX mode of operation. In *Fast Software Encryption*, pages 389–407. Springer, 2004.
2. B. Bilgin, A. Bogdanov, M. Knežević, F. Mendel, and Q. Wang. Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware. In *Cryptographic Hardware and Embedded Systems-CHES 2013*, pages 142–158. Springer, 2013.
3. A. Bogdanov, F. Mendel, F. Regazzoni, V. Rijmen, and E. Tischhauser. ALE: AES-Based Lightweight Authenticated Encryption. In *Fast Software Encryption*, 2013.
4. I. Dinur and J. Jean. Cryptanalysis of FIDES. In *Fast Software Encryption–FSE 2014*. Springer, 2014.
5. D. Engels, M.-J. O. Saarinen, P. Schweitzer, and E. M. Smith. The Hummingbird-2 Lightweight Authenticated Encryption Algorithm. In *RFID. Security and Privacy*, pages 19–31. Springer, 2012.
6. E. Fleischmann, C. Forler, and S. Lucks. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In *Fast Software Encryption–FSE 2012*, pages 196–215. Springer, 2012.
7. ISO/IEC 19772:2009. *Information technology – Security techniques – Authenticated encryption.* ISO, Geneva, Switzerland, 2009.

8. T. Iwata and K. Yasuda. BTM: A Single-Key, Inverse-Cipher-Free Mode for Deterministic Authenticated Encryption. In *Selected Areas in Cryptography – SAC 2009*, pages 313–330. Springer, 2009.
9. T. Iwata and K. Yasuda. HBS: A Single-Key Mode of Operation for Deterministic Authenticated Encryption. In *Fast Software Encryption–FSE 2009*, pages 394–415. Springer, 2009.
10. C. S. Jutla. Encryption Modes with Almost Free Message Integrity. In *Advances in Cryptology – EUROCRYPT 2001*, pages 529–544. Springer, 2001.
11. D. Khovratovich and C. Rechberger. The LOCAL attack: Cryptanalysis of the authenticated encryption scheme ALE. In *Selected Areas in Cryptography – SAC 2013*. Springer Berlin Heidelberg, 2013.
12. T. Kohno, J. Viega, and D. Whiting. CWC: A High-Performance Conventional Authenticated Encrption Mode. In *Fast Software Encryption – FSE 2004*, pages 408–426. Springer, 2004.
13. T. Krovetz and P. Rogaway. Authenticated-encryption software performance: Comparison of ccm, gcm, and ocb. Available at `http://www.cs.ucdavis.edu/~rogaway/ocb/performance/`.
14. T. Krovetz and P. Rogaway. The Software Performance of Authenticated-Encryption Modes. In *Fast Software Encryption – FSE 2011*, pages 306–327. Springer, 2011.
15. D. McGrew and J. Viega. The Galois/Counter Mode of Operation (GCM). `http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec.pdf`.
16. NIST. Recommendation for Block Cipher Modes of Operation-Methods and Techniques. NIST special publication 800–38A, 2001 Edition.
17. B. Preneel and P. van Oorschot. MDx-MAC and building fast MACs from hash functions. In *Advances in Cryptology – CRYPTO' 95*, volume 963 of *LNCS*, pages 1–14. Springer, 1995.
18. P. Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In *Advances in Cryptology–ASIACRYPT 2004*, pages 16–31. Springer, 2004.
19. D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). Available from `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmo-des/ccm/ccm.pdf`, 2003.
20. S. Wu, H. Wu, T. Huang, M. Wang, and W. Wu. Leaked-State-Forgery Attack against the Authenticated Encryption Algorithm ALE. In K. Sako and P. Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, volume 8269 of *Lecture Notes in Computer Science*, pages 377–404. Springer Berlin Heidelberg, 2013.