



Pengenalan Enkripsi Terautentikasi (Authenticated Encryption) Versi 1.0

Deputi Bidang Strategi dan Kebijakan Keamanan Siber dan Sandi
Badan Siber dan Sandi Negara

2025

PENGENALAN ENKRIPSI TERAUTENTIKASI (*AUTHENTICATED ENCRYPTION*) VERSI 1.0



Direktorat Kebijakan Teknologi Keamanan Siber dan Sandi
Badan Siber dan Sandi Negara (BSSN)
Jl. Cijayanti No. 70, Kec. Babakan Madang, Kabupaten Bogor
Website : <https://www.bssn.go.id>
Email : humas@bssn.go.id

TIM PERUMUS

Pengarah : Tjahjo Khurniawan

Penanggung Jawab : Soetedjo Joewono

Pereviu : 1. Dr. Sugi Guritman
(Institut Pertanian Bogor)
2. Sari Agustini Hafman, S.Stat., M.Si.
(Badan Siber dan Sandi Negara)

Tim Penyusun :

1. Anjariah Dwi Astuti
2. Adang Rochiyat
3. Ratih Mumpuni Arti
4. Novita Angraini
5. Theresia Natalia
6. Siti Hikmatul Hasanah
7. Dian Yuda Paripurna

Penyunting :

1. Novita Angraini
2. Oky Jati Permana

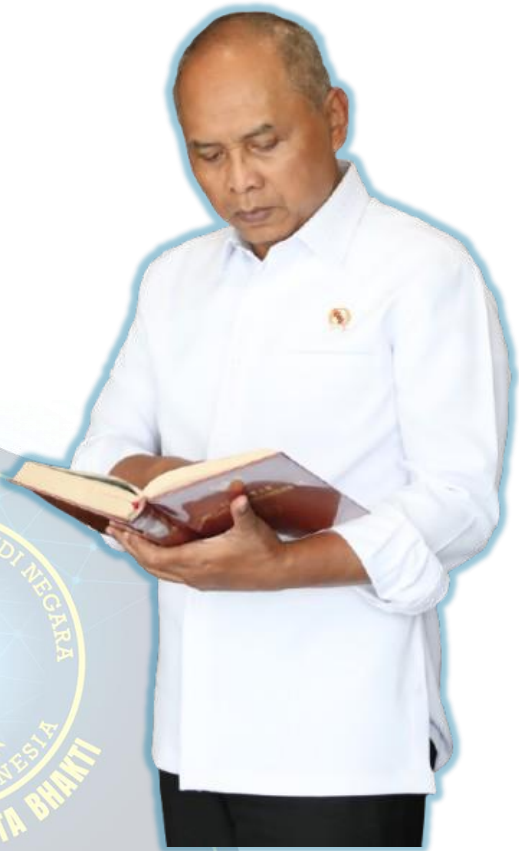


PENGANTAR

Puji syukur kita panjatkan kepada Tuhan Yang Maha Esa karena dokumen *Pengenalan Enkripsi Terautentikasi (Authenticated Encryption) Versi 1.0* ini dapat diselesaikan dan disajikan sebagai salah satu referensi teknis dalam memperkuat keamanan dan keandalan sistem elektronik nasional. Penyusunan dokumen ini merupakan bagian dari upaya BSSN dalam mendorong penerapan mekanisme kriptografi modern yang mampu memenuhi kebutuhan perlindungan data pada era digital.

Transformasi digital yang berlangsung cepat pada berbagai sektor kehidupan telah menghadirkan peluang besar bagi peningkatan efektivitas layanan publik, inovasi industri, dan efisiensi proses bisnis. Namun, transformasi tersebut juga membawa tantangan baru terhadap keamanan data dan keandalan sistem elektronik. Pertukaran data yang semakin masif, penggunaan layanan daring yang semakin luas, serta keterhubungan perangkat dan aplikasi yang semakin kompleks telah memperluas permukaan serangan (*attack surface*) yang berpotensi dimanfaatkan oleh pihak-pihak yang tidak bertanggung jawab. Serangan penyadapan, manipulasi pesan, penyusupan pada komunikasi terenkripsi, hingga pemalsuan identitas digital kini muncul dengan teknik yang lebih canggih dan sulit dideteksi tanpa penerapan mekanisme pengamanan yang kuat, terukur, dan berbasis standar.

Dalam konteks ancaman yang berkembang pesat tersebut, mekanisme pengamanan tradisional yang hanya mengandalkan enkripsi tidak lagi memadai. Pelindungan data harus mampu menjamin tidak hanya



kerahasiaan tetapi juga integritas dan autentikasi pesan secara menyeluruh. AE hadir sebagai solusi kriptografi modern yang menjawab kebutuhan itu. AE merupakan gabungan fungsi enkripsi dan autentikasi dalam satu mekanisme terpadu, yang dapat mencegah serangkaian serangan terhadap komunikasi elektronik, seperti manipulasi *ciphertext*, pemalsuan pesan, dan penggantian data pada kanal komunikasi yang dianggap aman.

Dokumen ini berisi pengenalan menyeluruh mengenai konsep, algoritma, serta mekanisme AE yang banyak digunakan dalam protokol dan sistem komunikasi modern. Selain itu, dokumen ini juga diharapkan dapat menjadi acuan bagi para pemangku kepentingan, baik pengambil kebijakan, perancang sistem, praktisi keamanan siber, maupun akademisi dalam memahami dan mengimplementasikan AE secara tepat, konsisten, dan selaras dengan standar keamanan global.

Saya menyampaikan apresiasi kepada seluruh jajaran BSSN, khususnya Deputy Bidang Strategi dan Kebijakan Keamanan Siber dan Sandi, serta para pakar dan mitra yang telah berkontribusi dalam penyusunan dokumen ini. Semoga panduan ini dapat memberikan manfaat yang nyata dalam memperkuat fondasi keamanan sistem elektronik nasional, serta mewujudkan ruang siber Indonesia yang aman, tangguh, dan terpercaya.

Drs. Nugroho Sulistyو Budi, M.M., M.Han.

Kepala Badan Siber dan Sandi Negara

Desember 2025

PENGANTAR DEPUTI I



Puji syukur kita panjatkan ke hadirat Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya, sehingga dokumen *Pengenalan Authenticated Encryption (AE) Versi 1.0* dapat diselesaikan dengan baik. Dokumen ini disusun sebagai bagian dari upaya memperkuat kebijakan dan penerapan teknologi kriptografi nasional guna mendukung peningkatan ketahanan dan keamanan siber Indonesia.

Perkembangan teknologi informasi yang semakin kompleks menuntut adanya mekanisme keamanan yang tidak hanya menjamin *confidentialitas*, tetapi juga memastikan integritas dan autentikasi data. AE hadir sebagai solusi kriptografi modern yang mengintegrasikan proses enkripsi dan autentikasi dalam satu kerangka yang aman dan efisien.

Kami menyampaikan terima kasih dan apresiasi yang sebesar-besarnya kepada para pakar, akademisi, dan tim penyusun yang telah memberikan kontribusi, masukan, serta dedikasi luar biasa sehingga dokumen ini dapat tersusun secara komprehensif, relevan, dan aplikatif. Dengan adanya dokumen ini, diharapkan para pemangku kepentingan, baik dari pemerintah, industri, akademisi, maupun penyelenggara sistem elektronik dapat memahami prinsip dan penerapan AE sesuai standar nasional dan internasional.

Semoga dokumen ini dapat menjadi rujukan awal bagi penguatan kapasitas nasional dalam penerapan mekanisme kriptografi AE, serta mendorong terwujudnya kemandirian dan kedaulatan teknologi keamanan informasi guna memperkuat ekosistem keamanan siber yang tangguh, adaptif, dan berdaya saing global.

Tjahjo Khurniawan

Deputi Bidang Strategi dan Kebijakan Keamanan Siber dan Sandi

Desember 2025



RINGKASAN EKSEKUTIF

Dokumen Pengenalan Enkripsi Terautentikasi (*Authenticated Encryption* (AE)) ini disusun oleh Badan Siber dan Sandi Negara (BSSN) sebagai langkah strategis dalam memperkuat keamanan dan keandalan sistem elektronik nasional melalui penerapan mekanisme kriptografi modern yang menjamin konfidensialitas, integritas, dan autentikasi data secara terpadu.

Dalam era digital yang ditandai dengan pertukaran data yang cepat dan masif, ancaman terhadap keamanan informasi seperti penyadapan, modifikasi pesan, dan pemalsuan identitas semakin kompleks. Oleh karena itu, dibutuhkan pendekatan kriptografi yang tidak hanya berfokus pada enkripsi, tetapi juga memberikan jaminan integritas dan autentikasi pesan. AE hadir untuk menjawab kebutuhan tersebut dengan menggabungkan fungsi enkripsi dan autentikasi dalam satu mekanisme yang efisien dan aman.

Dokumen ini memberikan pemahaman menyeluruh mengenai konsep dasar AE, mekanisme dan algoritma AE yang umum digunakan, serta contoh penerapannya dalam berbagai *library* kriptografi dan protokol keamanan, seperti protokol Kerberos, TLS, IPsec, SSH, dan IKE. Pembahasan juga mencakup beberapa mekanisme AE yaitu *Key Wrap*, *Counter with CBC-MAC* (CCM), *EAX*, *Encrypt-then-MAC* (EtM), *Galois/Counter Mode* (GCM), *ChaCha20-Poly1305*, dan *EAX Prime* (EAX) yang menjadi standar dalam implementasi kriptografi modern. Secara keseluruhan, dokumen ini berfungsi sebagai rujukan teknis dan kebijakan bagi penerapan AE di Indonesia, guna mendukung upaya peningkatan ketahanan dan keamanan siber Indonesia.

Dokumen ini ditujukan bagi pemangku kepentingan di bidang keamanan siber dan sandi, meliputi pengambil kebijakan, perumus kebijakan teknis, profesional dan praktisi keamanan siber, penyedia layanan dan produk keamanan teknologi informasi, serta akademisi dan peneliti. Melalui dokumen ini, diharapkan seluruh pihak terkait dapat memahami prinsip, standar, dan praktik terbaik penerapan AE dengan tepat, sehingga

implementasi mekanisme AE dapat dilakukan secara terencana, terukur, dan selaras dengan standar keamanan global.

Apabila pengguna menemukan keraguan dalam dokumen ini, maka disarankan untuk melihat referensi asli yang terdapat pada daftar referensi.



DAFTAR ISI

PENGANTAR	iii
PENGANTAR DEPUTI I.....	v
RINGKASAN EKSEKUTIF	vii
DAFTAR ISI.....	ix
DAFTAR TABEL.....	x
DAFTAR GAMBAR	xi
DAFTAR SIMBOL	xii
BAB I PENDAHULUAN	1
A. Ruang Lingkup.....	2
B. Target Audiens	4
C. Sistematika Penyusunan Dokumen	4
BAB II PENGANTAR DAN KONSEP DASAR.....	6
A. Gambaran Umum AE	7
B. Persyaratan dan Dokumen dalam Penggunaan Mekanisme AE.....	8
BAB III MEKANISME <i>AUTHENTICATED ENCRYPTION</i>	15
A. <i>Authenticated encryption mechanism 2 (key wrap)</i>	17
B. <i>Authenticated encryption mechanism 3 (CCM)</i>	24
C. <i>Authenticated encryption mechanism 4 (EAX)</i>	31
D. <i>Authenticated encryption mechanism 5 (encrypt-then-MAC)</i>	38
E. <i>Authenticated encryption mechanism 6 (GCM)</i>	45
F. Mekanisme AEAD_CHACHA20_POLY1305	52
G. Mekanisme EAX Prime (EAX')	59
BAB IV PEMANFAATAN <i>AUTHENTICATED ENCRYPTION</i>	70
A. Mekanisme <i>Key Wrap</i>	71
B. Mekanisme CCM	75
C. Mekanisme EAX dan EAX'.....	79
D. Mekanisme EtM	82
E. Mekanisme GCM	86
F. Mekanisme AEAD	91
BAB V PENUTUP	96
REFERENSI.....	98

DAFTAR TABEL

Tabel 2.1 Properti dari mekanisme AE.....	9
Tabel 3.1 Perbedaan KW, KWP dan TKW [10]	21
Tabel 3.2 Batasan Mandatory Plaintext dan Ciphertext pada KW, KWP dan TKW	23



DAFTAR GAMBAR

Gambar 3.1 Algoritma MAC 5.....	33
Gambar 3.2 Algoritma MAC 1.....	42
Gambar 3.3 Komputasi Algoritma $CMAC'K$	66
Gambar 3.4 Komputasi algoritma $CTR'K$	67
Gambar 3.5 Algoritma enkripsi EAX'	68
Gambar 3.6 Algoritma dekripsi EAX'	69
Gambar 4.1 Proses Pembuatan JWE	78



DAFTAR SIMBOL

- A data tambahan terautentikasi (*additional authenticated data*)
- C *string* data terenkripsi terautentikasi (*authenticated-encrypted data string*)
- D *string* data yang akan diterapkan mekanisme enkripsi terotentikasi
- d algoritma dekripsi *block cipher*; $d_K(Y)$ menotasikan hasil dekripsi *block cipher* terhadap blok sepanjang n -bit blok Y menggunakan kunci rahasia K
- e algoritma enkripsi *block cipher*; $e_K(X)$ menotasikan hasil enkripsi *block cipher* terhadap blok sepanjang n -bit blok X menggunakan kunci rahasia K
- K kunci rahasia *block cipher* yang dibagikan antara *originator* dan *recipient* data, yang akan diterapkan mekanisme enkripsi terautentikasi.
- m jumlah blok pada versi hasil partisi dari D
- n panjang blok (dalam satuan bit) untuk suatu *block cipher*
- t panjang *tag* (dalam satuan bit).
- 0^i blok yang terdiri atas i bit bernilai nol.
- 1^i blok yang terdiri atas i bit bernilai satu.
- $\oplus$ operasi *exclusive-or* (XOR) bit demi bit pada dua *string* bit dengan panjang yang sama.
- $||$ operasi konkatenasi (penggabungan) *string* bit., i.e. jika A dan B adalah dua blok bit, maka $A||B$ adalah blok bit yang diperoleh dengan menggabungkan A dan B dalam urutan tersebut
- $\#$ fungsi untuk mengonversi sebuah bilangan menjadi blok bit sepanjang a -bit
- Jika k merupakan suatu integer dalam rentang ($0 \leq k < 2^a$), maka $\#_a(k)$ merupakan blok a -bit yang, apabila dianggap sebagai representasi biner dari suatu bilangan dengan bit paling signifikan berada di sebelah kiri, bernilai sama dengan k .

- $\#^{-1}$ fungsi untuk mengonversi sebuah blok bit menjadi bilangan
- Jika A merupakan sebuah blok bit, maka $\#^{-1}(A)$ Adalah integer non-negatif unik yang representasi binernya adalah A . Oleh karena itu, jika A memiliki n bit, maka $\#_n(\#^{-1}(A)) = A$.
- $X|_s$ *left-truncation* dari blok bit X
- Jika X memiliki panjang bit yang lebih besar dari atau sama dengan s , maka $X|_s$ adalah blok sepanjang s -bit yang terdiri atas s bit paling kiri dari X .
- $X|^s$ *right-truncation* dari blok bit X
- Jika X memiliki panjang bit yang lebih besar dari atau sama dengan s , maka $X|^s$ adalah blok sepanjang s -bit yang terdiri atas s bit paling kanan dari X .
- $X \ll 1$ pergeseran ke kiri (*left shift*) dari sebuah blok bit X sebanyak satu posisi.
- Bit paling kanan dari $Y = X \ll 1$ selalu diset menjadi nol.
- $X \gg 1$ pergeseran ke kanan (*right shift*) dari sebuah blok bit X sebanyak satu posisi.
- Bit paling kiri dari $Y = X \gg 1$ selalu diset menjadi nol.
- len fungsi yang menerima sebuah *string* bit X sebagai input, dan menghasilkan *output* berupa jumlah bit dalam X
- mod jika a dan $b > 0$ merupakan integer, maka $a \bmod b$ menotasikan integer unik c sehingga:
- 1) $0 \leq c < b$; dan
 - 2) $a - c$ merupakan kelipatan integer dari b .

BAB I

PENDAHULUAN



BAB I

PENDAHULUAN

Bab ini menguraikan tujuan, ruang lingkup, target audiens, serta sistematika penyusunan dokumen “*Pengenalan Authenticated Encryption (AE)*”. Dokumen ini disusun sebagai acuan awal untuk memahami konsep dan penerapan AE sebagai mekanisme kriptografi yang mampu menjamin tidak hanya konfidensialitas tetapi juga integritas dan autentikasi informasi secara terpadu.

A. Ruang Lingkup

Dalam era digital yang ditandai dengan pertukaran informasi secara cepat dan masif, perlindungan data menjadi aspek yang semakin krusial. Informasi yang ditransmisikan melalui jaringan maupun disimpan dalam sistem tidak hanya menghadapi ancaman terhadap konfidensialitas tetapi juga integritas dan autentikasi. Ancaman seperti penyadapan, modifikasi pesan, hingga pemalsuan identitas menuntut adanya solusi kriptografi yang tidak hanya berfokus pada jaminan konfidensialitas semata melalui enkripsi, melainkan juga mampu memberikan jaminan autentikasi dan integritas.

Salah satu pendekatan kriptografi yang lahir untuk menjawab kebutuhan tersebut adalah AE. Berbeda dengan mekanisme kriptografi klasik yang memisahkan fungsi enkripsi dan autentikasi, AE menggabungkan keduanya dalam satu kerangka yang terintegrasi. Dengan cara ini, informasi yang dikirim tidak hanya terlindungi dari upaya pengungkapan oleh pihak yang tidak berhak, tetapi juga memiliki jaminan bahwa pesan tersebut tidak dimodifikasi selama transmisi dan benar-benar berasal dari pengirim yang sah.

Sejarah pengembangan AE menunjukkan adanya kebutuhan praktis yang mendesak. Sebelum AE diperkenalkan, pendekatan umum yang digunakan adalah mengombinasikan *ciphertext* dengan *Message Authentication Code* (MAC). Namun cara penyusunan yang kurang tepat dapat membuka celah keamanan. Penelitian oleh Bellare dan

Namprempre [1] menegaskan bahwa paradigma *Encrypt-then-MAC* adalah yang paling aman dan menjadi dasar bagi desain AE modern. Gagasan ini diperluas oleh Bellare, Rogaway, dan Wagner [2] melalui konsep *authenticated encryption with associated data* (AEAD), yang memungkinkan bagian tertentu dari data (seperti *header* protokol) tetap terbuka namun terproteksi integritasnya.

Kini, AE maupun AEAD telah menjadi fondasi dalam berbagai protokol dan aplikasi keamanan modern. Berbagai standar internasional seperti ISO/IEC, RFC dan NIST menjelaskan mekanisme AE yang kemudian digunakan secara luas dalam berbagai protokol keamanan, seperti Kerberos, TLS, IPsec, SSH, dan lain-lain. Sementara itu, sektor lain seperti *embedded systems*, SCADA, hingga transaksi elektronik juga menjadikan AE sebagai komponen kriptografi yang tak terpisahkan.

Dengan cakupan yang semakin luas, pemahaman mengenai konsep, mekanisme, dan pemanfaatan AE menjadi penting, tidak hanya bagi akademisi, tetapi juga praktisi, dan pengembang sistem keamanan siber. Dokumen ini hadir untuk memberikan pengenalan terhadap AE, mulai dari konsep dasar, mekanisme, hingga contoh penerapannya dalam berbagai bidang.

Dokumen ini memberikan penjelasan tentang:

1. Pengantar dan Konsep Dasar

Sebuah pengantar singkat tentang bagaimana gambaran dan persyaratan umum mekanisme AE.

2. Mekanisme AE

Mekanisme AE yang dijelaskan pada dokumen ini terdiri atas beberapa mekanisme, yaitu *key wrap*, *Counter with Cipher Block Chaining-MAC* (CCM), EAX, *Encrypt then MAC* (EtM), *Galois/ Counter Mode* (GCM), AEAD_ChaCha20-Poly1305, dan EAX Prime (EAX').

3. Pemanfaatan AE

AE dapat diterapkan dalam berbagai aplikasi maupun sistem. Contoh penggunaan AE dijelaskan dalam pemanfaatannya dengan

mekanisme *key wrap*, CCM, EAX, EtM, GCM, ChaCha20-Poly1305, dan EAX’.

B. Target Audiens

Dokumen ini ditujukan bagi para penyelenggara sistem elektronik, baik publik maupun privat, yang meliputi:

1. Para pengambil kebijakan keamanan siber dan sandi yang memiliki pemahaman dasar tentang konsep kriptografi dan ingin memastikan keamanan informasi dalam organisasi berjalan sesuai standar keamanan nasional dan internasional;
2. Perumus kebijakan keamanan siber dan sandi dengan latar belakang teknis atau bekerja sama dengan pakar teknis dalam penyusunan kebijakan yang selaras dengan perkembangan AE;
3. Profesional/praktisi keamanan siber dan sandi yang memiliki pemahaman tentang kriptografi, dasar matematika (teori bilangan, aljabar abstrak, dan probabilitas), serta skema dan protokol kriptografi dasar untuk mengimplementasikan AE dalam sistem elektroniknya untuk memastikan sistem elektronik yang dikelola memenuhi standar internasional;
4. Penyedia layanan/produk keamanan teknologi informasi yang memiliki kapabilitas teknis dalam mengembangkan dan mengadaptasi AE ke dalam layanan atau produk keamanan sesuai standar internasional.

C. Sistematika Penyusunan Dokumen

Dokumen ini terdiri dari beberapa bab yang menguraikan berbagai aspek AE. Untuk selanjutnya, dokumen ini disusun sebagai berikut:

BAB II. PENGANTAR DAN KONSEP DASAR

Bab ini berisi pengantar singkat tentang gambaran dan persyaratan umum mekanisme AE.

BAB III. MEKANISME *AUTHENTICATED ENCRYPTION*

Bab ini berisi tentang mekanisme AE yang terdiri atas beberapa mekanisme, yaitu *key wrap*, CCM, EAX, EtM, GCM, ChaCha20-Poly1305, dan EAX’.

BAB IV. PEMANFAATAN *AUTHENTICATED ENCRYPTION*

Bab ini berisi tentang pemanfaatan AE dengan mekanisme yang sudah dijelaskan pada Bab III.

BAB IV. PENUTUP

Bab ini berisi tentang kesimpulan dari seluruh isi dokumen.



BAB II

PENGANTAR DAN

KONSEP DASAR



BAB II

PENGANTAR DAN KONSEP DASAR

Bab ini memberikan gambaran umum mengenai konsep dasar AE, termasuk prinsip kerja, tujuan keamanan yang dicapai, serta perkembangan menuju mekanisme AEAD. Selain itu, bab ini menjelaskan persyaratan dan referensi standar yang digunakan dalam penerapan berbagai mekanisme AE, sebagaimana diatur dalam SNI ISO/IEC 19772:2020, guna memastikan penerapan yang aman, efisien, dan selaras dengan ketentuan kriptografi internasional.

A. Gambaran Umum AE

AE adalah transformasi data menggunakan algoritma kriptografi untuk menghasilkan *ciphertext* yang tidak dapat diubah oleh entitas yang tidak berwenang tanpa terdeteksi, sehingga menjamin tujuan keamanan sebagai berikut [3].

1. **Konfidensialitas**, yaitu perlindungan terhadap pengungkapan data oleh pihak yang tidak berwenang;
2. **Integritas data** yaitu perlindungan yang memungkinkan penerima data (*recipient*) untuk memverifikasi bahwa data tidak mengalami perubahan; dan
3. **Autentikasi asal data** yaitu perlindungan yang memungkinkan *recipient* untuk memverifikasi identitas pengirim (*originator*) atau asal data.

Umumnya enkripsi dan MAC digunakan bersama dengan algoritma kriptografi dan kunci yang terpisah. Namun, perkembangan terbaru memungkinkan kedua layanan tersebut disediakan bersamaan melalui satu algoritma kriptografi, yaitu AEAD, yang menggantikan penggunaan enkripsi dan MAC secara terpisah. AEAD tidak hanya menggabungkan enkripsi dan MAC ke dalam satu kerangka kerja, tetapi juga mendukung pemrosesan *associated data*, yaitu data tambahan yang tidak dienkripsi tetapi tetap harus dijamin integritas dan autentikasinya (misalnya *header* protokol jaringan). Dengan AEAD, diperoleh manfaat sebagai berikut [2]:

1. Efisiensi: enkripsi dan autentikasi dilakukan dalam satu algoritma, sehingga lebih cepat dan hemat sumber daya.
2. Kesederhanaan: mengurangi risiko kesalahan implementasi karena tidak perlu mengelola dua mekanisme kriptografi yang berbeda.
3. Keamanan yang lebih kuat: memastikan *ciphertext* tidak dapat dipalsukan maupun diubah tanpa terdeteksi.
4. Dukungan untuk *associated data*: memungkinkan perlindungan integritas data tambahan yang penting dalam protokol komunikasi (misalnya *header* TLS atau IPsec)

Contoh AEAD yang banyak digunakan saat ini antara lain: AES-GCM, AES-CCM, dan ChaCha20-Poly1305, yang menjadi standar dalam protokol keamanan modern seperti TLS 1.3, IPsec, IKE, dan SSH.

B. Persyaratan dan Dokumen dalam Penggunaan Mekanisme AE

Penggunaan setiap mekanisme AE ataupun AEAD memerlukan pemilihan parameter spesifik. Rekomendasi terkait persyaratan parameter yang berlaku untuk mekanisme *key wrap*, CCM, EAX, EtM, dan GCM diberikan dalam Lampiran A SNI ISO/IEC 19772:2020 [3]. Dokumen SNI ISO/IEC 19772:2020 bertujuan untuk memandu pengembang dan praktisi keamanan informasi dalam memilih atau merancang skema AE berbasis *block cipher* yang aman dan efisien. Di dalamnya terdapat sejumlah mekanisme AE yang telah diakui secara internasional karena efektivitas dan keamanannya yang sesuai dengan prinsip dan persyaratan standar.

SNI ISO/IEC 19772:2020 mencakup beberapa mekanisme yang masing-masing memiliki karakteristik desain, kelebihan, dan kasus penggunaan yang berbeda, yaitu *key wrap*, CCM, EAX, EtM, dan GCM. Kelima mekanisme ini telah dikaji secara kriptografis dan telah digunakan dalam berbagai sistem dan protokol dunia nyata. Pemahaman yang baik terhadap masing-masing mekanisme sangat penting untuk memastikan penerapan AE yang sesuai dengan kebutuhan keamanan dan performa suatu sistem.

Semua mekanisme dalam SNI ISO/IEC 19772:2020 memerlukan algoritma *block cipher*. Algoritma *block cipher* yang dapat digunakan terdapat pada SNI ISO/IEC 18033-3 [4] dan daftar Algoritma Kriptografi Indonesia [5]. Panjang blok (*block length*) n dari *block cipher* harus minimal 64 bit, dan bila memungkinkan, disarankan menggunakan *block cipher* dengan $n = 128$ bit. Penggunaan *block cipher* dengan $n = 128$ bit diwajibkan untuk mekanisme *key wrap*, CCM, dan GCM [3].

Kelima mekanisme tersebut juga mensyaratkan agar *originator* dan *recipient* berbagi sebuah kunci rahasia K . Kunci ini hanya boleh diketahui oleh kedua pihak tersebut dan, bila diperlukan, diketahui pula oleh pihak ketiga yang dipercaya. Terdapat berbagai cara untuk membentuk kunci ini. Namun, disarankan menggunakan mekanisme pembentukan kunci (*key establishment mechanism*) yang ditentukan dalam ISO/IEC 11770-2 atau ISO/IEC 11770-3.

Kelima mekanisme juga memerlukan pemilihan panjang *tag*. *Tag* adalah nilai autentikasi yang dihasilkan untuk menjamin autentikasi asal data dan integritas. Pemilihan parameter ini memengaruhi tingkat jaminan yang diberikan kepada penerima terkait integritas dan asal pesan yang dilindungi.

Semua mekanisme yang ditetapkan diyakini memberikan tingkat keamanan yang tinggi. Namun, beberapa mekanisme lebih sesuai daripada yang lain untuk aplikasi tertentu. Saat memilih mekanisme untuk digunakan, informasi yang diberikan dalam Tabel 2.1 di bawah ini dapat dijadikan pertimbangan.

Tabel 2.1 Properti dari mekanisme AE
tabel dibuat ulang dari [3].

Mekanisme	<i>Key Wrap</i>	CCM	EAX	<i>Encrypt-then-MAC</i>	GCM
<i>Approximate number of block cipher operations required to encrypt a q-bit</i>	$12 \lceil q/n \rceil$	$2q/n$	$2q/n$	<i>Depends on encryption and MAC methods used</i>	q/n

Mekanisme	Key Wrap	CCM	EAX	Encrypt-then-MAC	GCM
<i>Lisence possibly required</i>	No	No	No	<i>Depends on encryption and MAC methods used</i>	No
<i>Specifically designed for use with short messages</i>	Yes	No	No	No	No
<i>Message lengths is to be known prior to starting encryption</i>	No	Yes	No	No	No
<i>Starting value required</i>	No	Yes	Yes	Yes	Yes
<i>Previously standardized</i>	Yes	Yes	No	No	Yes

Persyaratan dasar yang harus dipenuhi oleh setiap skema AE sesuai dengan SNI ISO/IEC 19772:2020 mencakup parameter input seperti kunci simetrik, *nonce*, *plaintext*, dan *associated data*; serta output berupa *ciphertext* dan tag autentikasi. Berikut persyaratan dari tiap mekanisme AE dalam SNI ISO/IEC 19772:2020 dijelaskan sebagai berikut.

1. Mekanisme *Key Wrap*

Mekanisme *key wrap* dirancang khusus untuk mengamankan kunci kriptografi atau data sensitif lainnya dalam proses distribusi atau penyimpanan. Mekanisme ini memberikan jaminan konfidensialitas, integritas, dan autentikasi asal data atas kunci maupun data penting yang dibungkus (*wrapped*). Mekanisme ini juga mengombinasikan proses:

- key encryption* (enkripsi kunci);
- data authentication* (autentikasi data).

Mekanisme ini umumnya digunakan dalam standar pengamanan seperti *Cryptographic Message Syntax* (CMS) dan *Key Management Interoperability Protocol* (KMIP), dengan kelebihan antara lain: didesain khusus untuk *key management* (seperti dalam distribusi kunci antar modul HSM), melindungi konfidensialitas, keautentikan, dan integritas kunci, serta cocok untuk *wrapping* kunci dan *wrapping* data sensitif. Sedangkan kekurangannya adalah bukan untuk komunikasi data pada umumnya, kemudian perlu manajemen kunci yang kuat, serta kompatibilitas terbatas di luar ekosistem HSM/KMIP.

Mekanisme *keywrap*, hanya menggunakan algoritma *block cipher* dengan ukuran blok 128 bit ($n = 128$). Algoritma *block cipher* yang dapat digunakan terdapat pada SNI ISO/IEC 18033-3 [4] dan daftar Algoritma Kriptografi Indonesia [5].

2. Mekanisme *Counter with CBC-MAC* (CCM)

Mekanisme CCM merupakan gabungan antara *CTR mode* untuk enkripsi dan CBC-MAC untuk autentikasi yang umumnya digunakan untuk sistem *embedded* atau perangkat nirkabel, yang komponennya terdiri atas *block cipher* (umumnya menggunakan AES), dan *nonce* yang unik untuk setiap pesan. Kelebihan mekanisme ini adalah penggunaannya yang luas, misalnya pada *Wi-Fi Protected Access 2* (WPA2). Selain dijelaskan dalam SNI ISO/IEC 19772:2020, mekanisme ini juga distandardisasi dalam NIST SP 800-38C [6]. Kekurangan pada mekanisme ini adalah proses enkripsi dan autentikasi tidak dapat dilakukan secara paralel.

Dalam mekanisme ini, hanya menggunakan algoritma *block cipher* dengan ukuran blok 128 bit ($n = 128$). Algoritma *block cipher* yang dapat digunakan terdapat pada SNI ISO/IEC 18033-3 dan daftar Algoritma Kriptografi Indonesia. Selain itu diperlukan pemilihan panjang *tag t* (dari himpunan {32, 48, 64, 80, 96, 112, 128}). Pemilihan panjang *tag t* bergantung pada lingkungan penggunaan. Semakin panjang nilai *t*, semakin tinggi tingkat keamanan terhadap serangan pemalsuan (*forgery attacks*) *tag* [3]. Namun demikian, kecuali terdapat

alasan kuat untuk memilih nilai yang berbeda, disarankan untuk menggunakan $t \geq 64$.

Mekanisme CCM memerlukan pemilihan panjang w (dalam satuan oktet) dari *field* panjang pesan dengan nilai dari himpunan $\{2, 3, 4, 5, 6, 7, 8\}$. Pemilihan w juga bergantung pada lingkungan penggunaan tetapi tidak memengaruhi tingkat keamanan. Nilai w yang lebih besar memungkinkan pesan yang lebih panjang, tetapi mengurangi panjang sisa dari variabel awal. Bahkan dengan $w = 8$, masih tersedia 56 bit variabel awal yang dapat dipilih untuk memastikan bahwa variabel awal yang berbeda digunakan untuk setiap pesan, yang memadai untuk sebagian besar aplikasi. Untuk mayoritas aplikasi, nilai $w = 4$ sudah memadai karena mendukung panjang pesan maksimum sebesar $2^{32} \approx 4 \times 10^9$ oktet.

3. Mekanisme EAX

Mekanisme EAX Adalah mode AE yang dibangun dari *cipher block chaining* (CBC) dan *counter mode*, yang komponennya terdiri atas penggunaan CMAC untuk autentikasi, serta penggunaan CTR untuk enkripsi. Fitur pada mekanisme EAX sepenuhnya dapat diparalelkan untuk enkripsi dan autentikasi. Kelebihan mekanisme ini adalah fleksibel dan efisien, dan biasanya digunakan untuk konstruksi perangkat lunak dan aplikasi umum.

Mekanisme ini memerlukan pemilihan parameter panjang *tag* t ($t \leq n$). Pemilihan panjang *tag* t bergantung pada lingkungan tempat mekanisme ini digunakan. Namun demikian, kecuali terdapat alasan kuat untuk memilih nilai yang berbeda, disarankan penggunaan $t \geq 64$.

4. Mekanisme *Encrypt-then-MAC* (EtM)

Mekanisme ini mengharuskan adanya pemilihan mode operasi serta metode untuk perhitungan MAC. Keamanan yang diberikan oleh EtM sangat bergantung pada keamanan dari kedua primitif yang mendasarinya, yaitu *block cipher* dan MAC.

Untuk mode operasi enkripsi, ketentuan keamanan yang tercantum dalam SNI ISO/IEC 10116 [7] harus dipatuhi. Secara khusus, apabila mode CBC digunakan untuk mengenkripsi beberapa *plaintext* dengan kunci yang sama, maka salah satu ketentuan berikut harus dipenuhi:

- a. setiap proses enkripsi *plaintext* harus menggunakan variabel awal (*initialization vector*) yang bersifat acak; atau
- b. blok *plaintext* pertama harus ditetapkan sesuai dengan urutannya pada suatu nilai yang unik untuk setiap *plaintext* (misalnya berupa *counter*).

Pemilihan algoritma MAC harus disesuaikan dengan konteks penggunaan skema enkripsi terautentikasi, dan dokumen yang diberikan dalam ISO/IEC 9797 (seluruh bagian) harus diperhatikan secara cermat. Secara khusus, apabila dipilih MAC berbasis *block cipher* sebagaimana didefinisikan dalam SNI ISO/IEC 9797-1 [8], maka:

- a. algoritma MAC 1 hanya boleh digunakan apabila panjang pesan bersifat tetap; dan
- b. metode *padding* 1 hanya boleh digunakan apabila panjang pesan bersifat tetap.

Penjelasan tentang algoritma MAC 1 dan metode *padding* 1 dijelaskan pada Bab III bagian D.

5. Mekanisme *Galois/Counter Mode* (GCM)

Dalam mekanisme ini, hanya menggunakan algoritma *block cipher* dengan ukuran blok 128 bit ($n = 128$). Algoritma *block cipher* yang dapat digunakan terdapat pada SNI ISO/IEC 18033-3 [4] dan daftar Algoritma Kriptografi Indonesia [5].

Variabel awal dengan panjang variabel, S , harus dipilih sedemikian sehingga memenuhi ketentuan $1 \leq \text{len}(S) \leq 2^{64}$. Persyaratan bahwa variabel awal tidak pernah digunakan kembali

selama masa pakai suatu kunci merupakan hal yang kritis bagi keamanan mekanisme ini.

Panjang *tag*, t , harus dipilih sedemikian rupa sehingga t merupakan kelipatan 8 dengan ketentuan $96 \leq t \leq 128$. (Nilai $t = 32$ atau $t = 64$ juga diperbolehkan untuk aplikasi khusus, meskipun opsi ini hanya boleh digunakan dengan sangat hati-hati. Panduan lebih rinci mengenai penggunaan panjang *tag* tersebut tersedia pada Lampiran C Referensi [9]).

String data, D , yang akan menjadi objek mekanisme enkripsi terautentikasi harus memenuhi syarat $\text{len}(D) \leq (2^{39} - 256)$. *String additional authenticated data* (AAD), A , harus memenuhi syarat $\text{len}(A) \leq 2^{64}$.

Jumlah total blok data dan blok data tambahan yang dapat diproses oleh GCM untuk satu kunci tetap K paling banyak 2^{64} . Selain itu, jumlah total pemanggilan prosedur enkripsi untuk setiap kunci tertentu paling banyak 2, kecuali jika $\text{len}(S) = 96$ untuk setiap penggunaan kunci tersebut.

Mode AE yang menggunakan *counter mode* dan *Galois field multiplication* untuk autentikasi biasanya digunakan pada jaringan berkecepatan tinggi, aplikasi *real-time*, dengan komponen terdiri atas enkripsi menggunakan CTR, serta autentikasi menggunakan operasi dalam aritmetik *Galois Field*. Kelebihan mekanisme ini adalah memiliki kecepatan tinggi, dan sangat efisien untuk perangkat keras, serta didukung secara luas (contohnya TLS dan IPsec). Sedangkan kekurangannya adalah sangat sensitif terhadap penggunaan ulang *nonce*.

BAB III

MEKANISME *AUTHENTICATED* *ENCRYPTION*



BAB III

MEKANISME AUTHENTICATED ENCRYPTION

Berdasarkan SNI ISO/IEC 19772:2020, terdapat lima mekanisme AE. Kelima mekanisme yang ditetapkan dalam SNI ISO/IEC 19772:2020 [3] didasarkan pada algoritma *block cipher*, dan mengharuskan *originator* dan *recipient* berbagi kunci rahasia untuk *block cipher* tersebut. Kelima mekanisme yang ditetapkan dalam SNI ISO/IEC 19772:2020 adalah sebagai berikut:

- a. *Authenticated encryption mechanism 2 (key wrap)*
- b. *Authenticated encryption mechanism 3 (CCM)*
- c. *Authenticated encryption mechanism 4 (EAX)*
- d. *Authenticated encryption mechanism 5 (encrypt-then-MAC)*
- e. *Authenticated encryption mechanism 6 (GCM).*

Originator dan *recipient* yang akan menerapkan mekanisme AE harus memenuhi persyaratan:

- a. menyepakati mekanisme yang akan digunakan dari mekanisme yang ditetapkan dalam SNI ISO/IEC 19772:2020;
- b. menyepakati penggunaan *block cipher* yang akan digunakan bersama mekanisme tersebut (harus menggunakan salah satu *block cipher* yang distandarkan dalam SNI ISO/IEC 18033-3 [4]);
- c. berbagi kunci rahasia K . Pada semua mekanisme, kecuali mekanisme AE 5, K harus berupa kunci untuk *block cipher* yang dipilih, sedangkan pada mekanisme AE 5, K digunakan sebagai input untuk prosedur pembangkitan kunci.

Selain lima mekanisme AE di atas yang didasarkan pada algoritma *block cipher*, terdapat juga mekanisme AE yang didasarkan pada *stream cipher*, yaitu ChaCha20-Poly1305 yang merupakan AEAD.

Manajemen kunci berada di luar cakupan dalam dokumen ini. Teknik manajemen kunci didefinisikan dalam ISO/IEC 11770 (semua bagian). Selain itu, setiap mekanisme memiliki persyaratan khusus yang dijelaskan tepat sebelum deskripsi masing-masing mekanisme.

A. *Authenticated encryption mechanism 2 (key wrap)*

Skema *Authenticated encryption Mechanism 2 (key wrap)* pada awalnya didesain untuk AE dari kunci dan informasi yang terasosiasi. Dengan kata lain, skema ini dibuat khusus untuk mengenkripsi *string* data pendek. Namun, skema ini juga dapat digunakan untuk mengenkripsi *string* data dengan panjang *arbitrary*/sebarang (sampai maksimal sekitar 2^{67} bit). Meskipun demikian, skema ini tidak efisien untuk melindungi pesan yang panjang.

Key wrap dengan AE adalah metode yang digunakan untuk melindungi kunci kriptografi atau data sensitif lainnya dengan menjamin konfidensialitas, keautentikan dan integritas secara bersamaan. Prosesnya terdiri dari 2 (dua) fungsi, yaitu [10]:

1. Fungsi AE, yaitu fungsi yang mengenkripsi *plaintext* menjadi *ciphertext* dan menyediakan cara bagi fungsi *Authenticated-Decryption* yang terasosiasi untuk memverifikasi keautentikan dan integritas data.
2. Fungsi *Authenticated-Decryption*, yaitu fungsi yang mendekripsi *ciphertext* yang diklaim menjadi *plaintext* yang sesuai serta memverifikasi keautentikan dan integritas data. *Output* fungsi ini bisa berupa *plaintext* yang valid atau sebuah indikasi bahwa *plaintext* tersebut tidak autentik.

Berikut ini mekanisme enkripsi terautentikasi mekanisme 2 (*key wrap*) berdasarkan SNI ISO/IEC 19772:2020.

1. Notasi

Untuk tujuan spesifikasi mekanisme ini, berlaku simbol dan notasi berikut:

$C_0, C_1, \dots, C_m$	sekuens dari $(m + 1)$ blok 64-bit yang didapatkan sebagai output dari proses AE
$D_1, D_2, \dots, D_m$	sekuens dari m blok 64-bit yang didapatkan dengan mempartisi D , yaitu $64m = \text{len}(D)$
$R_1, R_2, \dots, R_m$	sekuens dari m blok 64-bit dihitung selama proses enkripsi dan dekripsi
Y	blok 64-bit digunakan selama proses enkripsi dan dekripsi
Z	blok 128-bit dihitung selama proses enkripsi dan dekripsi

2. Persyaratan

Block cipher yang akan digunakan dalam mekanisme ini harus berupa *block cipher* 128-bit, yaitu harus memiliki panjang blok, $n = 128$. *String* data D yang akan diproteksi menggunakan mekanisme ini harus berisi sedikitnya 128 bit dan kelipatan 64 bit (yaitu panjang bit D harus $64m$ untuk integer $m > 1$).

3. Prosedur Enkripsi

Originator harus melakukan langkah-langkah berikut untuk melindungi *string* data D .

- Partition D into a sequence of m 64-bit blocks $D_1, D_2, \dots, D_m$, so that D_1 contains the first 64 bits of D , D_2 the next 64 bits, and so on.
- Let Y be the 64-bits block having hexadecimal representation A6A6A6A6A6A6A6A6, i.e in binary it equals (10100110 10100110 ... 10100110).
- For $i = 1, 2, \dots, m$:
Let $R_i = D_i$.
- For $i = 1, 2, \dots, m$, perform the following four steps:
 - Let $Z = e_K(Y || R_i)$;

2) Let $Y = Z|_{64} \oplus \#_{64}(i)$;

3) For $j = 1, 2, \dots, m - 1$:
 Let $R_j = R_{j+1}$

4) Let $R_m = Z|_{64}$.

e) Let $C_0 = Y$

f) For $i = 1, 2, \dots, m$:
 Let $C_i = R_i$.

The output of the above process, i.e. the authenticated-encrypted version of D , shall be the bit-string:

$$C = C_0 || C_1 || \dots || C_m$$

That is, a string of $64(m+1)$ bits, that is C contains precisely 64 bits more than D .

4. Prosedur Dekripsi

Recipient harus melakukan langkah-langkah berikut untuk mendekripsi dan memverifikasi *string* C yang telah dienkripsi dan diautentikasi.

- a) If $\text{len}(C)$ is not a multiple of 64 or is less than 192, then halt and output INVALID.
- b) Partition C into a sequence of $m+1$ 64-bit blocks $C_0, C_1, \dots, C_m$ so that C_0 can contain the first 64 bits of C , C_1 the next 64 bits, and so on.
- c) Let $Y = C_0$.
- d) For $i = 1, 2, \dots, m$:
 let $R_i = C_i$.
- e) For $i = 6m, 6m - 1$, down to 1, perform the following four steps:
 - 1) Let $Z = d_K([Y \oplus \#_{64}(i)] || R_m)$;
 - 2) Let $Y = Z|_{64}$;
 - 3) For $j = m, m - 1, \dots, 2$:

$\text{let } R_j = R_{j-1};$

4) Let $R_1 = Z|^{64}$

f) If $Y = (10100110 \ 10100110 \ \dots \ 10100110)$, then output
 $D = R_1 || R_2 || \dots || R_m$. Otherwise, output INVALID.

Selain SNI ISO/IEC 19772:2020, NIST juga menerbitkan rekomendasi untuk *key wrap*, yaitu NIST SP 800-38F *Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping*. NIST SP 800-38F [10] merupakan bagian keenam dari seri rekomendasi mengenai mode operasi *block cipher*. Sama seperti SNI ISO/IEC 19772:2020, tujuan dari NIST SP 800-38F adalah menyediakan rekomendasi untuk metode *key wrapping*, yaitu proteksi kunci kriptografi.

Apabila SNI ISO/IEC 19772:2020 hanya membahas metode *key wrapping* berdasarkan *block cipher* dengan panjang blok, $n=128$, maka NIST SP 800-38F membahas lebih detail dengan membagi metode *key wrapping* menjadi 3 (tiga) jenis, yaitu:

1. Enkripsi terautentikasi deterministik mode operasi *block cipher* *Advanced Encryption Standard (AES)*, atau disebut *AES Key Wrap (KW)*;
2. *Key Wrap with Padding (KWP)* yaitu KW dengan skema *padding* internal yang meningkatkan interoperabilitas; dan
3. *Triple Data Encryption Algorithm Key Wrap (TKW)* yaitu KW analog dengan *Triple Data Encryption Algorithm (TDEA)* sebagai *block cipher* dasar, yang digunakan untuk mendukung aplikasi lama (*legacy application*).

KW, KWP dan TKW didesain untuk proteksi terhadap konfidensialitas dan autentisitas/integritas dari kunci kriptografi. Masing-masing menyediakan opsi untuk memproteksi kunci dengan cara yang berbeda dari metode yang memproteksi data umum.

Untuk KW dan KWP, *block cipher* yang digunakan harus disetujui, dan ukuran bloknnya harus 128 bit. Saat ini, satu-satunya *block cipher* yang

memenuhi kriteria ini adalah AES, dengan panjang blok 128, 192, atau 256 bit. KWP digunakan ketika kunci yang akan dibungkus memiliki panjang yang bukan kelipatan ukuran *block cipher* (8 *byte*/64 bit) sehingga membutuhkan tambahan *padding*. Sedangkan, untuk TKW, *block cipher* yang digunakan ditetapkan sebagai TDEA, sehingga panjang bloknnya adalah 64 bit. Tabel 3.1 menjelaskan secara umum perbedaan KW, KWP dan TKW

Tabel 3.1 Perbedaan KW, KWP dan TKW [10]

Fitur	KW	KWP	TKW
Algoritma Dasar	AES (128-bit blok)	AES (128-bit blok) + <i>padding</i>	TDEA (64-bit)
<i>Padding</i>	Tidak diperlukan (<i>plaintext</i> harus kelipatan ukuran blok)	Diperlukan (agar bisa menangani panjang <i>plaintext</i> yang tidak sesuai ukuran blok)	Tidak diperlukan
<i>Integrity Check Value</i> (ICV)	64-bit (0xA6A6A6A6A6A6A6A6)	32-bit (0xA65959A6)	32-bit (0xA6A6A6A6)
Kompatibilitas	AES-128, AES-192, AES-256	AES-128, AES-192, AES-256	TDEA

KW, KWP, dan TKW dirancang agar tangguh, artinya setiap bit *output* akan sangat bergantung pada setiap bit input dengan cara yang kompleks, bahkan jika data yang diamankan lebih panjang dari satu blok. Namun, kemampuan ini membuat kecepatan pemrosesan data jadi jauh lebih lambat dibandingkan mode AE lainnya. Meskipun begitu, *trade-off* ini bisa saja tepat untuk aplikasi manajemen kunci tertentu. Misalnya, jika panjang kunci yang ingin diproteksi lebih besar daripada ukuran blok dari *block cipher* yang digunakan, atau jika nilai dari data yang diproteksi sangat penting.

Dalam fungsi AE, *wrapping function* diterapkan pada *string plaintext* yang diperbesar untuk menghasilkan *ciphertext*. Setiap varian *key wrap* memperbesar *plaintext* dengan menambahkan *string* tetap yang disebut nilai pemeriksa integritas (*integrity check value/ICV*) di bagian awal. Pada KWP, *plaintext* yang diperbesar juga mencakup pengkodean 32-bit dari panjang *plaintext* dalam satuan oktet serta kemungkinan beberapa oktet “nol” sebagai *padding*.

Pada setiap varian *key wrap*, fungsi *Authenticated-Decryption* menerapkan *unwrapping function* pada *ciphertext* yang diklaim, lalu memeriksa apakah *string output* merupakan hasil pembesaran *plaintext* yang sesuai. Satuan panjang yang berguna untuk menggambarkan fungsi-fungsi ini adalah setengah ukuran blok, yaitu 64 bit untuk KW dan KWP, dan 32 bit untuk TKW. *String* bit dengan panjang ini disebut semiblok, dan untuk integer tak negatif n , istilah “ n semiblok” berarti *string* bit yang dapat direpresentasikan sebagai gabungan dari n semiblok, atau secara alternatif, *string* semiblok dengan panjang n .

Untuk setiap varian *key wrap*, *wrapping function* dan *unwrapping function* dijalankan pada tiga semiblok atau lebih, meskipun secara prinsip definisinya dapat diperluas hingga input dua semiblok. Panjang *output* sama dengan panjang input untuk kedua fungsi tersebut. KW dan TKW didefinisikan untuk dua semiblok atau lebih. Untuk KWP, domain input di-*padding* hingga *string* oktet tidak kosong (*nonempty octet strings*).

Batasan *mandatory* untuk panjang *plaintext* pada setiap varian *key wrap* serta batas panjang *ciphertext* yang bersesuaian merupakan ketentuan fundamental dalam NIST SP 800-38F. Ketentuan ini tidak hanya menetapkan ukuran maksimum *plaintext* yang dapat diproses oleh masing-masing varian, tetapi juga memastikan bahwa panjang *ciphertext* yang dihasilkan tetap berada dalam rentang yang ditentukan. Pada mekanisme KWP, *plaintext* yang diproses harus berbentuk *string* oktet. NIST SP 800-38F juga menetapkan bahwa algoritma KWP hanya berlaku ketika panjang *plaintext* berjumlah kurang dari 2^{32} oktet.

Berbeda dengan KWP, KW memiliki kapasitas untuk menerima input yang lebih panjang, meskipun tetap dibatasi sehingga *plaintext* untuk KW tidak boleh melebihi 2^{54} semiblok. Sementara itu, pada TKW batasan yang ditetapkan untuk panjang *plaintext* tidak boleh melebihi 2^{28} semiblok. Konsekuensi dari pembatasan tersebut adalah *ciphertext* yang dihasilkan untuk KW maksimal berjumlah 2^{54} semiblok, sedangkan untuk TKW maksimal 2^{28} semiblok. Kepatuhan terhadap persyaratan ini merupakan tanggung jawab pihak-pihak yang menggunakan, mengimplementasikan, memasang, atau mengonfigurasi aplikasi yang mengadopsi NIST SP 800-38F. Adapun batasan *mandatory* untuk KW, KWP, dan TKW dijelaskan dalam Tabel 3.2.

Tabel 3. 2 Batasan Mandatory *Plaintext* dan *Ciphertext* pada KW, KWP dan TKW

Algoritma	<i>Plaintext</i>	<i>Ciphertext</i>
KW	2 hingga $2^{54} - 1$ semiblok	3 hingga $2^{54} - 1$ semiblok
KWP	1 hingga $2^{32} - 1$ oktet	2 hingga 2^{29} semiblok
TKW	2 hingga $2^{28} - 1$ oktet semiblok	3 hingga 2^{28} semiblok

Dalam kriptografi, batasan jumlah panggilan (*invocation*) merujuk pada berapa kali suatu algoritma enkripsi dijalankan dengan kunci yang sama untuk *wrapping* kunci lain (*Key Encryption Key/KEK*). Tidak ada ketentuan untuk membatasi berapa kali algoritma KW atau KWP dijalankan, ataupun untuk TKW yang menggunakan TDEA sebagai *block cipher* dasarnya. Namun, ada ketentuan dalam referensi [11] yang menyebutkan bahwa, agar status “terbatas” tetap berlaku untuk TKW yang menggunakan dua kunci TDEA, jumlah penggunaan algoritma TKW dengan satu kunci *wrapping* (KEK) yang sama tidak boleh melebihi 2^{20} kali. Penjelasan lebih detail tentang status terbatas ini terdapat dalam publikasi tersebut.

B. *Authenticated encryption mechanism 3 (CCM)*

Counter with Cipher Block Chaining - Message Authentication Code (CCM) merupakan algoritma kriptografi yang dapat memberikan jaminan atas konfidensialitas dan keautentikan data. CCM didasarkan pada algoritma *block cipher*, dengan ukuran blok sebesar 128 bit, seperti algoritma AES yang distandarkan dalam *Federal Information Processing Standard Publication* (FIPS Pub) 197. CCM dapat dianggap sebagai salah satu mode operasi dari algoritma *block cipher*. Seperti halnya mode operasi lainnya, kunci tunggal untuk *block cipher* harus ditetapkan terlebih dahulu di antara pihak-pihak yang akan bertukar data. Oleh karena itu, CCM harus diimplementasikan dalam struktur manajemen kunci yang dirancang dengan baik. CCM ditujukan untuk penggunaan dalam lingkungan paket (*packet environment*), yaitu, saat semua data tersedia dalam penyimpanan sebelum CCM diterapkan. CCM tidak dirancang untuk mendukung pemrosesan parsial atau pemrosesan aliran.

Input ke CCM mencakup 3 (tiga) elemen, yaitu [6]:

- data yang akan diautentikasi sekaligus dienkripsi, disebut *payload*;
- *associated data*, misalnya *header*, yang akan diautentikasi namun tidak dienkripsi;
- nilai unik, umumnya disebut *nonce* yang ditetapkan untuk *payload* dan data terkait.

Berdasarkan NIST SP 800-38C, CCM terdiri dari dua proses yang saling berkaitan:

1. Pembangkitan dan enkripsi (*generation-encryption*)

Dalam proses ini, *cipher block chaining* (CBC) diterapkan pada *payload*, *associated data*, dan *nonce* untuk membangkitkan sebuah kode autentikasi pesan (MAC) kemudian enkripsi mode *counter* (CTR) diterapkan pada MAC dan *payload* untuk mengubahnya menjadi bentuk *ciphertext*.

Dengan demikian, proses *generation-encryption* pada CCM akan menambahkan ukuran *payload* sebesar ukuran MAC.

2. Dekripsi dan verifikasi (*decryption-verification*)

Pada proses ini, dekripsi mode CTR diterapkan pada *ciphertext* untuk mendapatkan kembali MAC dan *payload* yang bersesuaian; kemudian, CBC diterapkan pada *payload*, data terkait, dan *nonce* yang diterima untuk memverifikasi kode MAC.

Verifikasi yang berhasil dapat memberikan jaminan bahwa *payload* dan data terkait berasal dari sumber sah yang memiliki akses ke kunci.

Berdasarkan SNI ISO/IEC 19772:2020, CCM terdiri atas prosedur enkripsi dan prosedur dekripsi. Berikut adalah penjelasan mekanisme enkripsi terautentikasi mekanisme 3 (CCM) berdasarkan SNI ISO/IEC 19772:2020.

1. Keterangan Notasi

Berikut ini simbol-simbol dan notasi yang digunakan:

B	blok bit yang digunakan dalam perhitungan nilai <i>tag</i>
$B_1, B_2, \dots, B_v$	urutan blok bit (masing-masing terdiri atas n bit) yang digunakan dalam perhitungan nilai <i>tag</i>
$C_1, C_2, \dots, C_m$	urutan m blok 128-bit yang diperoleh sebagai bagian dari <i>output</i> proses enkripsi-terotentikasi
$D_1, D_2, \dots, D_m$	urutan m blok 128-bit yang diperoleh dari pembagian versi D yang telah di- <i>padding</i>
F	oktet <i>flag</i> (penanda)
L	panjang D (dalam oktet), tidak termasuk <i>padding</i> dan blok panjang D_0
r	jumlah oktet dari D yang berada dalam blok D_m
S	variabel awal (sepanjang 120–8w bit)

T	nilai <i>tag</i> dalam bentuk <i>plaintext</i> (sepanjang t bit)
T'	nilai <i>tag</i> yang dihitung ulang, dihasilkan selama proses dekripsi
U	nilai <i>tag</i> terenkripsi (sepanjang t bit)
v	variabel yang digunakan dalam perhitungan nilai <i>tag</i>
w	panjang dari field panjang pesan (<i>message length field</i>) dalam octet
X	blok 128 bit yang dihitung selama proses enkripsi dan dekripsi
Y	blok 128 bit yang dihitung selama proses enkripsi dan dekripsi

2. Persyaratan Spesifik

Sebelum menjalankan mekanisme ini, *originator* dan *recipient* data harus menyepakati nilai:

- t , yaitu panjang bit dari *tag*; nilai t harus dipilih dari himpunan: $\{32, 48, 64, 80, 96, 112, 128\}$.
- w , yaitu panjang oktet dari field panjang pesan; w harus dipilih dari himpunan: $\{2, 3, 4, 5, 6, 7, 8\}$.

Catatan:

- Pilihan nilai w mempengaruhi panjang maksimum pesan yang bisa diproteksi.
- Panjang maksimum pesan adalah 2^{8w+3} bit, atau 2^{8w} oktet.

Algoritma *block cipher* yang digunakan harus *block cipher* 128 bit.

Data *string* D dan data autentikasi tambahan A harus terdiri atas sejumlah penuh oktet (*whole number of octets*), yaitu panjang data harus kelipatan dari 8 bit. Dengan kata lain, $len(D)$ dan $len(A)$ harus kelipatan dari 8.

3. Prosedur Enkripsi

Originator melakukan langkah berikut untuk melindungi *string* data *D*.

Let $L = \frac{\text{len}(D)}{8}$, L is the number of octets in D .

a) A starting variable S containing 15-w octets (i.e. 120-8w bits) shall be selected. This variable shall be distinct for every message to be protected, and shall be made available to the recipient of the message. However, it is not necessary that this value is unpredictable or secret.

NOTE 1 The value S can, for example, be generated using a counter maintained by the originator, and sent in cleartext along with the protected message.

b) Right pad the data string D with 16-r zero octets (i.e. between 0 and 120 zero bits) so that the padded version of D contains a multiple of 128 bits. Then, partition the padded version of D into a sequence of m 128-bit blocks $D_1, D_2, \dots, D_m$, so that D_1 contains the first 128 bits of D , D_2 the next 128 bits, and so on.

NOTE 2 The value m needs to satisfy $16(m-1) < L \leq 16m$.

c) If $\text{len}(A) = 0$, then let the flag octet $F = 0^2 \parallel \#_3 \left(\frac{(t-16)}{16} \right) \parallel \#_3(w-1)$.

d) If $\text{len}(A) > 0$, then let the flag octet $F = 0 \parallel 1 \parallel \#_3 \left(\frac{(t-16)}{16} \right) \parallel \#_3(w-1)$

NOTE 3 The most significant (left-most) bit of F is a "reserved" bit, i.e. it is set to zero for the version of the mechanism specified here but can be used in the future in other (as yet unspecified) versions of the mechanism. The next to the most significant bit of F is

set to zero to indicate that all the data being protected by the mechanism is encrypted.

e) Let $X = e_K(F \parallel S \parallel \#_{8w}(L))$.

f) If $\text{len}(A) > 0$, then perform the following six steps:

1) if $0 < \text{len}(A) < 65\,280$, then let $B = \#_{16}\left(\frac{\text{len}(A)}{8}\right) \parallel A$;

2) if $65\,280 \leq \text{len}(A) < 2^{32}$, then let $B = 1^{15} \parallel 0 \parallel \#_{32}\left(\frac{\text{len}(A)}{8}\right) \parallel A$;

3) if $2^{32} \leq \text{len}(A) < 2^{64}$, then let $B = 1^{16} \parallel \#_{64}\left(\frac{\text{len}(A)}{8}\right) \parallel A$;

4) partition B into a sequence of blocks: $B_1, B_2, \dots, B_v$ as follows: let B_1 contain the first n bits of B , B_2 the next n bits, and so on, until B_v contains the final k bits, where $0 < k \leq n$. Thus, $\text{len}(B) = (v-1)n + k$;

5) right pad B_v with $n-k$ zeros, i.e. let $B_v = B_v \parallel 0^{n-k}$;

6) for $i = 1, 2, \dots, v$:

let $X = e_K(X \oplus B_i)$.

g) For $i = 1, 2, \dots, m$:

let $X = e_K(X \oplus D_i)$.

h) Let $T = X|_t$.

NOTE 4 The plaintext tag T is equal to a MAC computed on the data string $B_1, B_2, \dots, B_v, D_1, D_2, \dots, D_m$ using a slight modification of MAC algorithm 1 specified in ISO/IEC 9797-1.

i) Let the flag octet $F = 0^5 \parallel \#_3(w-1)$, and let $Y = F \parallel S \parallel 0^{8w}$.

NOTE 5 The two most significant (left-most) bits of F are "reserved" bits, i.e. they are set to zero for the version of the mechanism specified here but can be used in the future in other (as yet unspecified) versions of the mechanism. The next three most significant bits of F are set to zero to ensure that this octet is distinct from the flag octet used in step c) above.

j) Let $U = T \oplus [e_K(Y)]|_t$.

k) For $i = 1, 2, \dots, m-1$, perform the following two steps:

1) let $Y = F \parallel S \parallel \#_{8w}(i)$;

2) let $C_i = D_i \oplus e_K(Y)$.

1) Let $Y = F \parallel S \parallel \#_{8w}(m)$, and let $C_m = [D_m \oplus e_K(Y)]_{8r}$.

The output of the above process, i.e. the authenticated-encrypted version of D , shall be the bit-string:

$$C = C_1 \parallel C_2 \parallel \dots \parallel C_{m-1} \parallel C_m \parallel U$$

That is, a string of $8L+t$ bits, that is C contains precisely t bits more than the original data string D [although it is also necessary to convey the $(120-8w)$ -bit starting variable S and the variable length additional authenticated data A to the recipient].

4. Prosedur Dekripsi

Berikut ini merupakan langkah yang dilakukan pihak penerima ketika memverifikasi dan mendekripsi *ciphertext* C untuk mendapatkan kembali data D dan mengecek keautentikannya.

a) If C does not contain a whole number of octets, then halt and output INVALID.

b) If the length of C is less than $(t+8)$ bits, then halt and output INVALID.

c) Let m and r be the unique integers such that C contains a total of $128(m-1)+8r+t$ bits, where $0 < r \leq 16$. Partition C into a sequence of blocks: $C_1, C_2, \dots, C_m, U$ as follows. Let C_1 contain the first 128 bits of C , C_2 the next 128 bits of C , and so on, until C_m contains the next $8r$ bits of C . Finally, let U be the final t bits of C .

d) Let the flag octet $F = (0^5 \parallel \#_3(w-1))$, and let $Y = (F \parallel S \parallel 0^{8w})$.

e) Let $T = U \oplus [e_K(Y)]_t$.

f) For $i = 1, 2, \dots, m-1$, perform the following two steps:

- 1) Let $Y = (F \parallel S \parallel \#_{8w}(i));$
- 2) Let $D_i = C_i \oplus e_K(Y).$
- g) Let $Y = (F \parallel S \parallel \#_{8w}(m)),$ and let $D_m = C_m \oplus [e_K(Y)]_{8r}.$
- h) Let $D = D_1 \parallel D_2 \parallel \dots \parallel D_m,$ and let $L = 16m - 16 + r.$
- i) Right pad D_m with $128-8r$ zeros, i.e. let $D_m = D_m \parallel 0^{128-8r}.$
- j) If $\text{len}(A) = 0,$ then let the flag octet $F = 0^2 \parallel \#_3((t - 16)/16) \parallel \#_3(w - 1).$
- k) If $\text{len}(A) > 0,$ then let the flag octet $F = 0 \parallel 1 \parallel \#_3((t - 16)/16) \parallel \#_3(w - 1).$
- l) Let $X = e_K(F \parallel S \parallel \#_{8w}(L)).$
- m) If $\text{len}(A) > 0,$ then perform the following six steps:
 - 1) if $0 < \text{len}(A) < 65\,280$ then let $B = \#_{16}(\text{len}(A)/8) \parallel A;$
 - 2) if $65\,280 \leq \text{len}(A) < 2^{32},$ then let $B = 1^{15} \parallel 0 \parallel \#_{32}(\text{len}(A)/8) \parallel A;$
 - 3) if $2^{32} \leq \text{len}(A) < 2^{64},$ then let $B = 1^{16} \parallel \#_{64}(\text{len}(A)/8) \parallel A;$
 - 4) partition B into a sequence of blocks: $B_1, B_2, \dots, B_v$ as follows: let B_1 contain the first n bits of $B,$ B_2 the next n bits, and so on, until B_v contains the final k bits, where $0 < k \leq n.$ Thus, $\text{len}(B) = (v - 1)n + k;$
 - 5) right pad B_v with $n-k$ zeros, i.e. let $B_v = B_v \parallel 0^{n-k};$
 - 6) for $i = 1, 2, \dots, v:$

$$\text{let } X = e_K(X \oplus B_i).$$
- n) For $i = 1, 2, \dots, m:$

$$\text{let } X = e_K(X \oplus D_i).$$
- o) Let $T' = X|_t.$
- p) If $T = T',$ then output D as computed in step h) and $A.$ Otherwise, output INVALID.

Terdapat serangan prekomputasi yang cukup umum terhadap semua mode *block cipher* yang memungkinkan serangan *meet-in-the-middle* pada kunci yang digunakan. Pengguna harus mengambil tindakan pencegahan untuk menyulitkan serangan prekomputasi. Penggunaan ulang nilai

nonce yang sama (meskipun dengan kunci yang berbeda) sebaiknya dihindari. Salah satu solusinya adalah memasukkan nilai acak ke dalam *nonce*. Tentu saja, *packet counter* juga diperlukan dalam *nonce*. Karena ukuran *nonce* terbatas, nilai acak di dalamnya hanya memberikan peningkatan keamanan dalam jumlah terbatas [12].

Untuk menjaga keamanan, jumlah total operasi enkripsi algoritma *block cipher* dalam CBC-MAC selama masa pakai satu kunci harus dibatasi hingga 2^{61} . Jakob Jonsson telah mengembangkan bukti keamanan CCM [12]. Buktinya menunjukkan bahwa CCM memberikan tingkat kerahasiaan dan keaslian yang sejalan dengan mode enkripsi terautentikasi lain, seperti mode *offset codebook* (OCB) yang diusulkan Phillip Rogaway, dkk [13].

C. **Authenticated encryption mechanism 4 (EAX)**

Sebagai respons terhadap keterbatasan CCM, EAX dirancang untuk menjadi lebih sederhana, lebih fleksibel, dan lebih efisien [2] EAX adalah mode AEAD berbasis *nonce* yang menggunakan *block cipher* apa pun sebagai dasarnya, meskipun dalam banyak implementasi AES sering digunakan sebagai *block cipher* yang mendasari.

Skema EAX adalah mode operasi *block cipher* yang dirancang untuk AEAD. Skema ini menggunakan kombinasi dua teknik kriptografi utama, yaitu *Counter Mode* (CTR) untuk enkripsi dan *One-Key CBC-MAC* (OMAC) untuk autentikasi [2].

Berikut ini mekanisme enkripsi terautentikasi mekanisme 4 (EAX) berdasarkan SNI ISO/IEC 19772:2020 [3].

1. Notasi

Dalam rangka spesifikasi mekanisme ini, berlaku simbol dan notasi sebagai berikut:

$C_1, C_2, \dots, C_m$ sekuens blok-bit (masing-masing terdiri dari n bit, dengan kemungkinan pengecualian pada C_m) yang

diperoleh sebagai bagian dari *output* proses enkripsi yang diautentikasi

$D_1, D_2, \dots, D_m$ urutan blok-bit (masing-masing terdiri dari n bit, dengan kemungkinan pengecualian pada D_m) yang diperoleh dari pemisahan D

E_0, E_1, E_2 blok-blok sepanjang n -bit yang dihitung selama proses enkripsi dan dekripsi

M fungsi yang digunakan dalam proses enkripsi dan dekripsi

S variabel awal (sepanjang n bit)

T *tag* (sepanjang t bits), yang dilekatkan pada pesan terenkripsi untuk memberikan perlindungan integritas

T' nilai *tag* yang dihitung ulang, yang dihasilkan selama proses dekripsi

W blok sepanjang n -bit yang dihitung selama proses enkripsi dan dekripsi

2. Persyaratan spesifik

Sebelum mekanisme ini digunakan, *originator* dan *recipient* data yang akan dikenai mekanisme AE harus menyepakati nilai t , yaitu panjang *tag* dalam bit, dengan ketentuan $0 < t \leq n$.

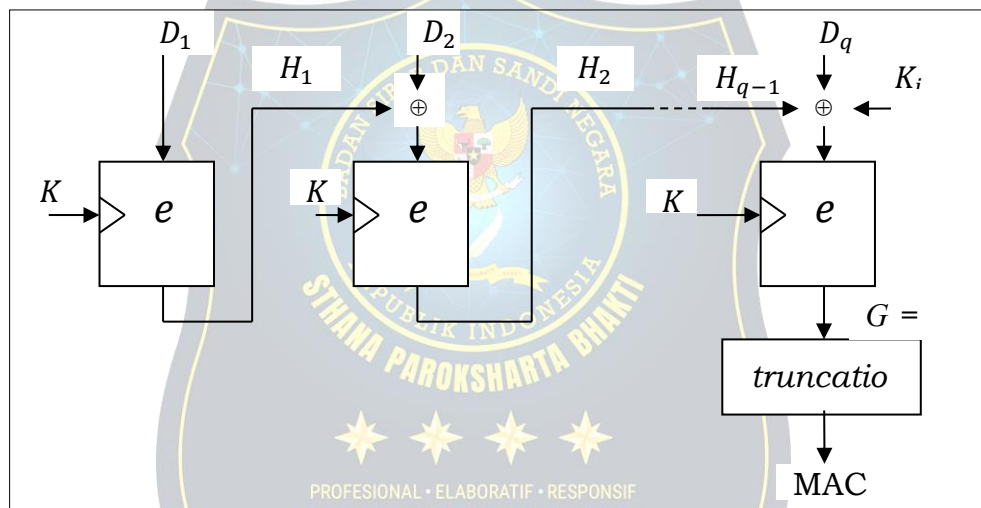
3. Definisi fungsi M

Prosedur enkripsi dan dekripsi memerlukan definisi fungsi M yang menerima input berupa *string* bit dengan panjang sebarang dan sebuah kunci *block cipher*, serta menghasilkan sebuah blok sepanjang n bit sebagai *output*. Definisi fungsi ini adalah sebagai berikut:

Jika X adalah sebuah *string* bit, dan K adalah kunci untuk algoritma *block cipher* yang dipilih, maka $M_K(X)$ harus sama dengan MAC yang tidak dipotong (*untruncated*), yang dihitung atas *string* X menggunakan kunci K dengan algoritma MAC 5 dari ISO/IEC 9797-1:2011 [8], dengan algoritma *block cipher* yang digunakan dalam

algoritma MAC tersebut harus sama dengan algoritma *block cipher* yang dipilih untuk proses AE.

Algoritma MAC 5 menggunakan Metode Derivasi Kunci 2, Iterasi Akhir 3, dan Transformasi *Output* 1. Algoritma MAC 5 hanya boleh digunakan dengan Metode *Padding* 4. Kunci masking K_1 dan K_2 yang digunakan dalam Iterasi Akhir 3 diturunkan dari kunci algoritma MAC K menggunakan Metode Derivasi Kunci 2. Algoritma MAC 5 umumnya dikenal sebagai OMAC1 [14] atau CMAC [15]. Kunci algoritma MAC terdiri atas satu buah kunci *block cipher* K . Algoritma MAC 5 diilustrasikan pada Gambar 3.1 ,dengan $K_i = K_1$ atau K_2 .



Gambar 3.1 Algoritma MAC 5
Digambar ulang dari [8].

Jika Algoritma MAC 5 digunakan bersamaan dengan algoritma yang menghitung pengenalan kunci (*key identifier*) (yang bersifat publik) sebagai $S = e_K(0^n)$, seperti dalam X9.24 [16], maka Algoritma MAC 5 rentan terhadap serangan pemalsuan XOR (*XOR forgery attacks*) (Lampiran C SNI ISO/IEC 9797-1:2011 [8]). Dalam kasus seperti ini, algoritma hanya boleh digunakan jika serangan tersebut tidak mungkin dilakukan, misalnya karena panjang pesan bersifat tetap.

a. Metode derivasi kunci 2

Metode ini menghitung dua kunci *masking* K_1 dan K_2 yang masing-masing memiliki panjang n bit dari sebuah kunci *block cipher*. Metode ini terdiri atas langkah-langkah berikut:

- Pertama, *string* rahasia S dengan panjang n bit dihitung sebagai berikut: $S := e_K(0^n)$.
- Selanjutnya, kunci masking K_1 diperoleh dari $S : K_1 := \text{multx}(S)$
- Terakhir, kunci masking K_2 diturunkan dari $K_1 : K_2 := \text{multx}(K_1)$.

b. Iterasi Akhir 3

Transformasi ini menggunakan kunci *block cipher* K yang sama seperti pada proses iterasi, serta dua kunci *masking* K_1 dan K_2 yang masing-masing dengan panjang n bit. Blok H_q dihitung dengan melakukan operasi XOR antara input dan kunci K_1 atau K_2 (bergantung pada hasil operasi *padding*), kemudian dienkripsi menggunakan kunci *block cipher* K .

Sesuai dengan Metode *Padding* 4, jika panjang *string* data yang akan dimasukkan ke dalam algoritma MAC (dalam bit) merupakan kelipatan positif dari n , maka:

$$H_q := e_K(D_q \oplus H_{q-1} \oplus K_1)$$

Jika tidak, maka:

$$H_q := e_K(D_q \oplus H_{q-1} \oplus K_2)$$

Transformasi *output* 1 merupakan fungsi identitas, yaitu $G := H_q$

c. Metode *Padding* 4

Jika *string* data D yang akan dimasukkan ke dalam algoritma MAC memiliki panjang (dalam bit) yang merupakan kelipatan integer positif dari n , maka *padding* tidak perlu diterapkan. Jika tidak, maka *string* data D harus ditambahkan *padding* di sisi kanan dengan satu buah bit '1'.

Setelah itu, *string* hasilnya harus ditambahkan lagi *padding* di sisi kanan dengan sesedikit mungkin (atau tidak sama sekali) bit '0' untuk menghasilkan *string* data dengan panjang (dalam bit) yang merupakan kelipatan integer positif dari n .

Catatan:

Jika *string* data kosong, maka Metode *Padding* 4 menentukan bahwa *string* tersebut harus di-*padding* di sisi kanan dengan satu buah bit '1' diikuti oleh $n - 1$ buah bit '0'.

4. Prosedur enkripsi

Originator harus melakukan langkah-langkah berikut untuk melindungi sebuah *string* data D .

- a) A starting variable S containing n bits shall be selected. This variable shall be distinct for every message to be protected, and shall be made available to the recipient of the message. However, it is not necessary that this value is unpredictable or secret.
- b) Let $E_0 = M_K(0^n || S)$.
- c) Let $E_1 = M_K(0^{n-1} || 1 || A)$.
- d) Let $W = E_0$.
- e) Partition D into a sequence of blocks: $D_1, D_2, \dots, D_m$, as follows. Let D_1 contain the first n bits of D , D_2 the next n bits, and so on, until D_m contains the final r bits, where $0 < r \leq n$. Thus $\text{len}(D) = (m-1)n + r$.
- f) For $i = 1, 2, \dots, m-1$, perform the following two steps:
 - 1) let $C_i = D_i \oplus e_K(W)$;
 - 2) let $W = \#_n(\#^{-1}(W) + 1 \bmod 2^n)$.
- g) Let $C_m = D_m \oplus [e_K(W)]_r$.

h) Let $E_2 = M_K(0^{n-2}||1||0||C_1||C_2|| \dots ||C_m)$.

i) Let $T = [E_0 \oplus E_1 \oplus E_2]_t$.

The output of the above process, i.e. the authenticated-encrypted version of D , shall be the bit-string:

$$C = C_1 || C_2 || \dots || C_m || T$$

That is, a string of $(m-1)n+r+t$ bits, that is C contains precisely t bits more than D (although it is also necessary to convey the n -bit starting variable S and the variable length additional authenticated data A to the recipient).

5. Prosedur Dekripsi

Recipient harus melakukan langkah-langkah berikut untuk mendekripsi dan memverifikasi sebuah *string* C yang telah dienkripsi dan diautentikasi.

a) If the length of C is less than t , then halt and output INVALID.

b) Let m and r be the unique integers defined so that C contains a total of $(m-1)n+r+t$ bits, where $0 < r \leq n$. Partition C into a sequence of blocks: $C_1, C_2, \dots, C_m, T$ as follows. Let C_1 contain the first n bits of C , C_2 the next n bits of C , and so on, until C_m contains the next r bits of C . Finally, let T be the final t bits of C .

c) Let $E_0 = M_K(0^n||S)$.

d) Let $E_1 = M_K(0^{n-1}||1||A)$.

e) Let $E_2 = M_K(0^{n-2}||1||0||C_1||C_2|| \dots ||C_m)$.

f) Let $T' = [E_0 \oplus E_1 \oplus E_2]_t$.

- g) If $T \neq T'$, then halt and output INVALID.
- h) Let $W = E_0$.
- i) For $i = 1, 2, \dots, m-1$, perform the following two steps:
 - 1) let $D_i = C_i \oplus e_K(W)$;
 - 2) let $W = \#_n(\#^{-1}(W) + 1 \bmod 2^n)$.
- j) Let $D_m = C_m \oplus [e_K(W)]_r$.
- k) Output D and A .

Skema EAX memiliki beberapa keunggulan yang merupakan perbaikan dari keterbatasan pada skema CCM, keunggulan tersebut antara lain sebagai berikut [2]:

- a. Skema EAX mendukung AEAD. Data tambahan seperti *header* tetap dapat diautentikasi tanpa dienkripsi.
- b. Dalam skema EAX, tidak memerlukan *padding*. Tidak ada aturan *padding* rumit seperti di CCM.
- c. Skema EAX dilakukan secara *online processing* sehingga dapat mengenkripsi data tanpa harus mengetahui panjang pesan terlebih dahulu.
- d. Skema EAX mendukung *nonce* dengan panjang fleksibel. Tidak seperti CCM yang memiliki batasan ketat pada panjang *nonce*.

Salah satu kontribusi utama adalah pembuktian keamanan EAX berdasarkan model kriptografi yang kuat. Keamanan EAX bergantung pada asumsi bahwa *block cipher* yang digunakan adalah *pseudorandom permutation* (PRP) yang baik. Hal berarti bahwa selama *block cipher* memiliki properti acak yang cukup kuat, maka *ciphertext* yang dihasilkan oleh EAX akan tidak dapat dibedakan dari data acak, sehingga menjamin *confidentialitas* data.

Selain itu, EAX juga memastikan bahwa pesan tidak dapat dimodifikasi atau dipalsukan tanpa terdeteksi, karena setiap perubahan dalam *ciphertext* akan menyebabkan kegagalan dalam proses autentikasi. Dengan demikian, EAX menawarkan perlindungan terhadap serangan pemalsuan (*forgery*) dan serangan *chosen-ciphertext* [2].

Selain EAX, terdapat pula varian EAX' yang diadopsi dalam standar ANSI C12.22 yang ditujukan untuk efisiensi implementasi tanpa mengurangi jaminan keamanan dari mode aslinya. Penjelasan tentang algoritma EAX' dijelaskan pada Bab III bagian G.

D. *Authenticated encryption mechanism 5 (encrypt-then-MAC)*

Authenticated encryption mechanism 5 (encrypt-then-MAC) terdiri atas kombinasi mekanisme enkripsi dan skema MAC apapun. Mekanisme dasarnya melibatkan proses mengenkripsi terlebih dahulu data yang akan dilindungi, kemudian menghitung MAC dari data terenkripsi yang dihasilkan. Mekanisme varian *additional authenticated data* (AAD) memungkinkan AAD untuk diautentikasi tetapi tidak dienkripsi.

Pendekatan *encrypt-then-MAC* (tanpa AAD) telah dianalisis oleh Bellare dan Namprempre, yang memberikan pembuktian keamanan dengan asumsi bahwa metode enkripsi dan teknik MAC memenuhi sifat-sifat keamanan tertentu. Meskipun analisis Bellare dan Namprempre tidak mempertimbangkan perluasan teknik dengan menyertakan variabel awal atau AAD, dapat dibuktikan bahwa keamanan tetap terjaga dalam kasus tersebut.

Berikut ini mekanisme enkripsi terautentikasi mekanisme 5 (EtM) berdasarkan SNI ISO/IEC 19772:2020.

1. Notasi

Untuk keperluan spesifikasi mekanisme ini, simbol dan notasi berikut berlaku:

C' *string* bit yang diperoleh dengan mengenkripsi *string* data D

- δ fungsi dekripsi, yaitu suatu fungsi yang menerima sebagai input kunci *block cipher* K_1 , variabel awal S , dan *string* data terenkripsi C' , dan, mode operasi yang dipilih, menghasilkan *string* data yang didekripsi, *outputnya* ditulis $\delta_{K_1,S}(C')$
- ε fungsi enkripsi, yaitu suatu fungsi yang menerima sebagai input kunci *block cipher* K_1 , variabel awal S , dan *string* data D , dan dengan menggunakan mode operasi yang dipilih, menghasilkan *string* data yang terenkripsi, *outputnya* ditulis $\varepsilon_{K_1,S}(D)$
- f fungsi MAC
Jika X adalah *string* input, dan K_2 adalah kunci MAC, maka *output* MAC ditulis $f_{K_2}(X)$
- K_1 kunci rahasia untuk *block cipher*
- K_2 kunci rahasia untuk fungsi MAC
- S variabel awal (n bit)
- T *tag* (t bit), yang dilekatkan pada pesan terenkripsi untuk menyediakan proteksi integritas
- T' nilai *tag* yang dihitung ulang, dihasilkan selama proses dekripsi

2. Persyaratan khusus

Sebelum menggunakan mekanisme ini, *originator* dan *recipient* data yang akan menerapkan mekanisme AE harus menyepakati hal-hal berikut ini:

- a) Mode operasi *block cipher* yang terdapat dalam SNI ISO/IEC 10116 [7], yaitu CBC, *Cipher Feedback* (CFB), *Output Feedback* (OFB), dan CTR. Adapun mode *Electronic Codebook* (ECB) tidak boleh digunakan.
- b) Metode untuk komputasi MAC, yang harus dipilih dari teknik-teknik yang terdapat dalam ISO/IEC 9797(semua bagian) (diasumsikan bahwa metode yang dipilih menghasilkan *tag* sepanjang t bit).

- c) Metode untuk memperoleh pasangan kunci rahasia (K_1 , K_2) dari kunci rahasia K , dengan K_1 adalah kunci untuk *block cipher* yang dipilih dan K_2 adalah kunci untuk metode komputasi MAC yang dipilih.

K dipilih sedemikian sehingga jumlah nilai yang mungkin untuk K setidaknya sebesar jumlah nilai yang mungkin untuk kunci *block cipher*, dan juga setidaknya sebesar jumlah nilai yang mungkin untuk kunci MAC. (K_1 , K_2) dapat diperoleh dari kunci rahasia K dengan mengambil *string* bit (yang tidak tumpang tindih) dari K atau dari $h(K)$, dengan h adalah fungsi hash dalam ISO/IEC 10118 (semua bagian). Secara lebih umum, (K_1 , K_2) dapat diperoleh dari kunci rahasia K menggunakan fungsi derivasi yang ditentukan dalam ISO/IEC 11770-6 [17].

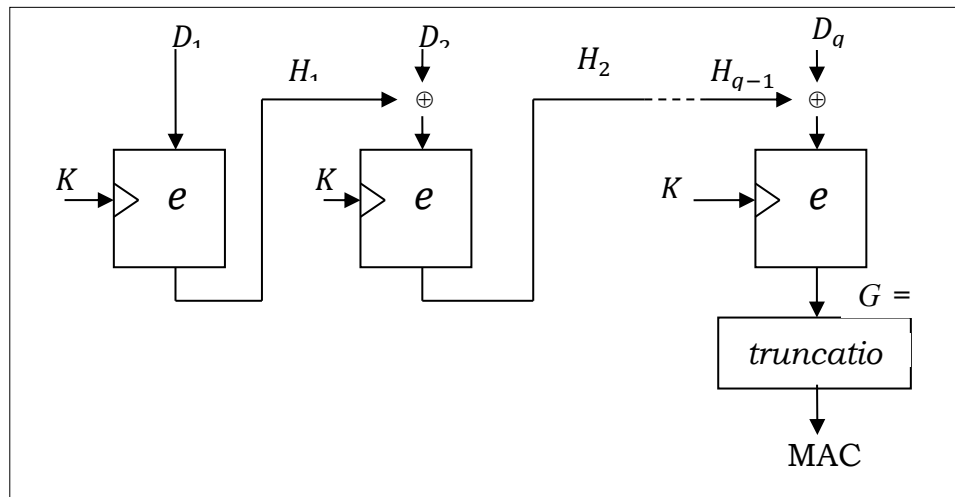
- d) Selanjutnya dapat menggunakan mekanisme dasar (yang tidak mendukung AAD) atau menggunakan mekanisme varian AAD. Jika menggunakan mekanisme varian AAD, maka *string* data terotentikasi tambahan A harus mengandung jumlah oktet yang bulat (kemungkinan nol), yaitu $len(A)$ harus merupakan kelipatan 8, tetapi harus mengandung kurang dari 2^{64} oktet (atau bahkan lebih sedikit bergantung pada persyaratan dari skema MAC yang digunakan). Satu kunci K hanya boleh digunakan dengan satu varian, yaitu hanya dengan varian dasar atau hanya dengan varian AAD.

3. Persyaratan tambahan

- a) Saran keamanan dalam SNI ISO/IEC 10116 [7] harus diikuti. Jika mode CBC digunakan untuk mengenkripsi beberapa *plaintext* dengan kunci yang sama, maka salah satu dari ketentuan berikut harus diterapkan:
- 1) variabel awal yang acak harus digunakan untuk setiap proses enkripsi *plaintext*; atau

- 2) blok *plaintext* pertama harus diatur sesuai dengan urutannya pada suatu nilai yang unik untuk setiap *plaintext* (misalnya, menggunakan *counter*).
- e) Pemilihan teknik MAC harus mempertimbangkan konteks penggunaan dari teknik AE tersebut, dan saran yang diberikan dalam ISO/IEC 9797 (semua bagian) harus diikuti dengan cermat. Secara khusus, jika MAC berbasis *block cipher* dari ISO/IEC 9797-1:2011 – *Message Authentication Codes (MACs) – Part 1: Mechanism using a block cipher* [8] yang dipilih, maka:
 - 1) algoritma MAC 1 hanya boleh digunakan jika panjang pesan bersifat tetap; dan
 - 2) metode *padding* 1 hanya boleh digunakan jika panjang pesan bersifat tetap.

Algoritma MAC 1 [8] menggunakan *Final Iteration* 1 dan *Output Transformation* 1. *Final iteration* F diterapkan pada blok terakhir D_q dari *string* data yang telah di-*padding* untuk menghasilkan blok H_q . *Final iteration* 1 menggunakan kunci *block cipher* K yang sama seperti pada proses iterasi sebelumnya. Blok H_q dihitung dengan menerapkan *block cipher* menggunakan kunci K sebagai berikut: $H_q := e_K(D_q \oplus H_{q-1})$. *Output transformation* g diterapkan pada nilai H_q , yang diperoleh sebagai hasil dari *final iteration*. *Output Transformation* 1 merupakan fungsi identitas, yaitu: $G := H_q$. Kunci Algoritma MAC terdiri atas kunci *block cipher* K . Algoritma MAC 1 diilustrasikan dalam Gambar 3.2 Algoritma MAC 1 rentan terhadap XOR *forgery attack*. Oleh karena itu, algoritma ini hanya boleh digunakan dalam situasi saat serangan tersebut tidak mungkin dilakukan, misalnya karena panjang pesan bersifat tetap. Algoritma MAC 1 diilustrasikan pada Gambar 3.2



Gambar 3.2 Algoritma MAC 1

Digambar ulang dari [8].

Algoritma MAC 1 dapat digunakan dengan Metode *Padding* 1, 2 atau 3. Pemilihan metode *padding* memengaruhi keamanan algoritma MAC. *Padding* merupakan tahap kedua dari tujuh tahapan dalam pengaplikasian algoritma MAC. Langkah ini melibatkan penambahan bit-bit tambahan (*padding*) di awal dan/atau akhir *string* data D , sehingga versi data yang sudah di-*padding* memiliki panjang yang selalu merupakan kelipatan dari n bit. Bit-bit *padding* yang ditambahkan ke *string* data asli, sesuai dengan metode *padding* yang dipilih, hanya digunakan untuk menghitung MAC. Oleh karena itu, bit-bit *padding* tersebut (jika ada) tidak perlu disimpan atau dikirimkan bersama dengan data. Pihak verifikator harus mengetahui apakah bit-bit *padding* disimpan atau dikirimkan, serta metode *padding* mana yang digunakan. Berikut ini penjelasan dari setiap metode *padding* yang dapat digunakan dalam Algoritma MAC 1:

1) Metode *Padding* 1

String data D yang akan dimasukkan ke algoritma MAC harus di-*padding* di sebelah kanan dengan sedikit mungkin (atau tanpa) bit '0' hingga panjangnya (dalam bit) menjadi kelipatan integer positif dari n . Algoritma MAC yang menggunakan Metode *Padding* 1 dapat menjadi rentan terhadap serangan pemalsuan

yang sepele (*trivial forgery attack*). Jika *string* data kosong, Metode *Padding* 1 menetapkan bahwa *string* tersebut di-*padding* di kanan dengan n buah bit '0'.

2) Metode *Padding* 2

String data D yang akan dimasukkan ke algoritma MAC harus di-*padding* di sebelah kanan dengan satu buah bit '1'. Setelah itu, *string* hasilnya di-*padding* di kanan dengan sedikit mungkin (atau tanpa) bit '0' hingga panjangnya (dalam bit) menjadi kelipatan integer positif dari n . Jika *string* data kosong, Metode *Padding* 2 menetapkan bahwa *string* tersebut di-*padding* di kanan dengan satu buah bit '1' diikuti oleh $n - 1$ buah bit '0'.

3) Metode *Padding* 3

String data D yang akan dimasukkan ke algoritma MAC harus di-*padding* di sebelah kanan dengan sedikit mungkin (atau tanpa) bit '0' hingga panjangnya (dalam bit) menjadi kelipatan integer positif dari n . *String* hasilnya kemudian di-*padding* di sebelah kiri dengan sebuah blok L . Blok L terdiri atas representasi biner dari panjang (dalam bit) L_D dari *string* data D yang belum di-*padding*, yang kemudian di-*padding* di kiri dengan sedikit mungkin (atau tanpa) bit '0' hingga membentuk blok berukuran n bit. Bit paling kanan dari blok L mewakili bit paling tidak signifikan (*least significant bit*) dari representasi biner L_D .

Metode *Padding* 3 tidak cocok digunakan dalam situasi di mana panjang dari *string* data tidak diketahui sebelum proses perhitungan MAC dimulai. Jika *string* data kosong, Metode *Padding* 3 menetapkan bahwa *string* tersebut di-*padding* di kanan dengan n buah bit '0' dan di-*padding* di kiri dengan blok L yang terdiri dari n buah bit '0'.

4. Prosedur enkripsi

Originator harus melakukan langkah-langkah berikut untuk melindungi *string* data D dan, jika menggunakan mekanisme varian AAD, untuk memastikan integritas dari *string additional authenticated data* A .

a) A starting variable S appropriate for use with the selected block cipher mode of operation shall be selected. Security requirements for S are as described in the appropriate clauses of ISO/IEC 10116, and further guidance is given in A.6.

b) Let $C' = \varepsilon_{K_1, S}(D)$.

If not using the AAD variant:

c) Let $T = f_{K_2}(S || C')$.

If using the AAD variant:

c) If $\text{len}(A)$ is not a multiple of 8 or is $\geq 2^{67}$, then halt and output INVALID.

Let $T = f_{K_2}(\#_{64}(\text{len}(A)/8) || A || S || C')$.

The output of the above process, i.e. the authenticated-encrypted version of D , shall be the bit-string:

$C = C' || T$, together with the starting variable S .

5. Prosedur dekripsi

Recipient harus melakukan langkah-langkah berikut untuk mendekripsi dan memverifikasi *string* terenkripsi-terotentikasi C , dengan variabel awal S yang menyertainya dan jika menggunakan mekanisme varian AAD, untuk memverifikasi integritas dari *additional authenticated data* A .

- a) If the length of C is less than t , then halt and output INVALID.
- b) Let T be the rightmost t bits of C , and let C' be equal to C with the rightmost t bits removed, i.e. $C = C' || T$.
If not using the AAD variant:
- c) Let $T' = f_{K_2}(S || C')$.
If using the AAD variant:
- c) If $\text{len}(A)$ is not a multiple of 8 or is $\geq 2^{67}$, then halt and output INVALID.
Let $T' = f_{K_2}(\#_{64}(\text{len}(A)/8) || A || S || C')$.
- d) If $T \neq T'$, then halt and output INVALID.
- e) Let $D = \delta_{(K_1, S)}(C')$.
- f) Output D .

E. Authenticated encryption mechanism 6 (GCM)

Galois/Counter Mode (GCM) adalah mekanisme AE yang dikembangkan oleh McGrew dan Viega [9]. GCM merupakan mode operasi yang dapat memberikan jaminan konfidensialitas dan autentikasi data. Operasi GCM bergantung pada pemilihan algoritma *block cipher* yang digunakan dengan ukuran blok minimal 128 bit, misalnya algoritma AES. Kunci GCM identik dengan kunci *block cipher* yang digunakan.

Untuk setiap kunci tertentu, *block cipher* yang mendasari mode ini terdiri atas dua fungsi yang merupakan invers satu sama lain. Pemilihan *block cipher* juga mencakup penetapan salah satu dari dua fungsi tersebut sebagai *forward cipher function*, sebagaimana ditentukan dalam spesifikasi algoritma *block cipher*. GCM tidak menggunakan fungsi *cipher* invers (dekripsi). *Forward cipher function* adalah permutasi terhadap *string* bit berukuran tetap (disebut blok), dengan panjang blok

disebut sebagai ukuran blok. Fungsi ini bergantung pada kunci K dan dilambangkan sebagai $CIPH_K$.

Block cipher yang digunakan dalam GCM harus memenuhi kriteria sebagai berikut:

- 1) Merupakan algoritma *block cipher* yang terdapat pada SNI ISO/IEC 18033-3 [4] dan daftar Algoritma Kriptografi Indonesia [5];
- 2) Memiliki ukuran blok sebesar 128 bit;
- 3) Memiliki ukuran kunci minimal 128 bit.

Kunci harus dibangkitkan secara acak seragam, atau sedekat mungkin dengan distribusi seragam, artinya setiap kemungkinan kunci memiliki peluang yang hampir sama untuk dibangkitkan. Dengan demikian, kunci yang digunakan akan bersifat *fresh*, yaitu sangat mungkin berbeda dari kunci-kunci yang pernah digunakan sebelumnya. Kunci harus dirahasiakan dan digunakan secara eksklusif untuk operasi GCM dengan *block cipher* yang dipilih.

GCM memberikan jaminan terhadap konfidensialitas data melalui penggunaan varian dari mode operasi *Counter* untuk proses enkripsi. Selain itu, GCM juga memberikan jaminan autentikasi terhadap data rahasia (hingga sekitar 64 gigabyte per satu kali pemanggilan) dengan memanfaatkan fungsi *hash* universal yang didefinisikan pada *binary Galois Field*, sebagaimana dijelaskan dalam NIST *Special Publication* 800-38D [9].

GCM juga mampu menyediakan jaminan autentikasi untuk AAD yang tidak dienkripsi, dengan panjang data tidak terbatas dalam satu kali pemrosesan. Apabila input GCM dibatasi hanya pada data yang tidak akan dienkripsi, maka spesialisasi dari GCM ini disebut sebagai GMAC, yang berfungsi murni sebagai mode autentikasi atas data input. Dalam NIST *Special Publication* 800-38D [9], seluruh pernyataan mengenai GCM juga berlaku untuk GMAC.

GCM memberikan jaminan autentikasi yang lebih kuat dibandingkan metode non-kriptografis seperti *checksum* atau kode

deteksi kesalahan (*error detecting code*). Secara khusus, GCM mampu mendeteksi modifikasi data yang tidak disengaja, serta modifikasi yang disengaja dan tidak sah.

GCM memiliki dua fungsi utama, yaitu *authenticated encryption* dan *authenticated decryption*. Kedua fungsi tersebut dirancang agar efisien dan dapat diparalelkan, sehingga memungkinkan implementasi berkecepatan tinggi baik di perangkat keras maupun perangkat lunak.

Salah satu hal penting dalam penggunaan GCM adalah bahwa pelanggaran terhadap persyaratan keunikan *string* inisialisasi dapat secara signifikan melemahkan jaminan keamanan. Oleh karena itu, mode operasi ini tidak boleh digunakan tanpa memastikan bahwa persyaratan keunikan tersebut benar-benar terpenuhi. Beberapa pertimbangan praktis lebih lanjut dibahas dalam NIST *Special Publication* 800-38D.

Dalam referensi [3] dijelaskan bahwa sebelum menggunakan GCM, *originator* dan *recipient* data perlu menyepakati panjang *tag* (t) dalam bit. Nilai t harus merupakan kelipatan 8 dan harus memenuhi syarat $96 \leq t \leq 128$. Ada juga opsi $t = 32$ dan $t = 64$ untuk aplikasi khusus. Variabel awal S harus dipilih dengan panjang bit $1 \leq \text{len}(S) \leq 2^{64}$ dan tidak boleh digunakan kembali selama penggunaan kunci yang sama untuk menjaga keamanan. *String* data D yang akan dienkripsi harus memenuhi $\text{len}(D) \leq 2^{39} - 256$. *String additional authenticated data*, A , harus memenuhi $\text{len}(A) \leq 2^{64}$.

Maksimum jumlah blok data dan blok A yang dapat dienkripsi dengan satu kunci adalah 2^{64} . Proses enkripsi dan dekripsi menggunakan GCM harus mematuhi jumlah total pemanggilan enkripsi maksimum 2^{32} untuk setiap kunci, kecuali jika $\text{len}(S) = 96$ untuk setiap penggunaan kunci tertentu.

Berikut ini mekanisme enkripsi terautentikasi mekanisme 6 (GCM) berdasarkan SNI ISO/IEC 19772:2020.

1. Notasi

Untuk keperluan spesifikasi mekanisme ini, simbol dan notasi berikut digunakan:

$C_1, C_2, \dots, C_m$ Urutan m blok 128 bit (dengan kemungkinan pengecualian pada C_m , yang dapat berisi antara 1 hingga 128 bit) yang diperoleh sebagai bagian dari *output* proses enkripsi terotentikasi.

$D_1, D_2, \dots, D_m$ Urutan m blok 128 bit (dengan kemungkinan pengecualian pada D_m) yang diperoleh dengan mempartisi *string* data D

G Fungsi yang digunakan dalam proses enkripsi dan dekripsi

H Blok 128-bit yang digunakan dalam proses enkripsi dan dekripsi.

inc Fungsi yang menerima sebuah blok 128 bit sebagai input dan menghasilkan blok 128 bit sebagai *output*, yang didefinisikan sebagai berikut

Jika X merupakan blok berukuran 128-bit, maka:

$$inc(X) = (X|_{96}) || \#_{32}(\#^{-1}(X|_{32}) + 1 \bmod 2^{32})$$

r Jumlah bit pada blok terakhir dari pesan yang akan dienkripsi, setelah pesan tersebut dibagi ke dalam blok-blok berukuran n bit. Dengan kata lain, pesan memiliki $(m - 1)n + r$ bit.

R Blok 128 bit yang digunakan dalam perhitungan perkalian GF (2^{128})

S Variabel awal (dengan panjang yang bersifat variabel).

T	tag (t bits), yang digabungkan dengan pesan terenkripsi untuk memberikan perlindungan integritas.
T'	Nilai tag yang dihitung ulang, yang dihasilkan selama proses dekripsi.
U, V, W, Z	Blok 128 bit yang digunakan dalam mendefinisikan perhitungan perkalian $GF(2^{128})$
$X_0, X_1, \dots, X_{K+i+1}$	Blok-blok 128 bit yang digunakan dalam perhitungan fungsi G
$Y_0, Y_1, \dots, Y_m$	Urutan blok 128 bit yang digunakan dalam proses enkripsi dan dekripsi
$\{\}$	<i>String</i> bit dengan panjang nol.
$\bullet$	Operasi perkalian pada <i>field</i> $GF(2^{128})$ Polinomial yang digunakan untuk menentukan representasi $GF(2^{128})$ adalah $1 + \alpha + \alpha^2 + \alpha^7 + \alpha^{128}$.

2. Definisi operasi perkalian • ★ ★ ★ ★

Misalkan U dan V adalah blok 128-bit; maka $W = U \bullet V$ didefinisikan sebagai berikut, dengan W juga merupakan blok 128-bit. Perlu dicatat bahwa dalam deskripsi berikut, v_i menyatakan bit ke- i dari V , yaitu: $V = v_0 || v_1 || \dots || v_{127}$. Dengan cara yang serupa, z_{127} menyatakan bit paling kanan dari Z .

- a. Let $R = 11100001 || 0^{120}$.
- b. Let $W = 0^{128}$.
- c. Let $Z = U$.
- d. For $i = 0, 1, \dots, 127$, perform the following two steps:
 - 1) if $v_i = 1$, then let $W = W \oplus Z$;
 - 2) if $z_{127} = 0$, then let $Z = Z \gg 1$. Otherwise let $Z = (Z \gg 1) \oplus R$.

3. Definisi Fungsi G

Prosedur enkripsi dan dekripsi menggunakan sebuah fungsi G , yang menerima input berupa satu blok 128-bit dan dua *string* bit dengan panjang sebarang, serta menghasilkan sebuah blok 128-bit sebagai *output*. Misalkan H adalah sebuah blok 128-bit, dan W serta Z adalah dua *string* bit dengan panjang sebarang (dapat kosong). Asumsikan terdapat integer unik k dan u sedemikian sehingga $\text{len}(W) = 128(k - 1) + u$ dan $0 < u \leq 128$.

Demikian pula, asumsikan terdapat integer unik l dan v sedemikian sehingga $\text{len}(Z) = 128(l - 1) + v$ dan $0 < v \leq 128$. Tetapkan $W_1, W_2, \dots, W_k$ merupakan urutan blok 128-bit (dengan kemungkinan pengecualian pada W_k , yang berisi u bit terakhir dari W) yang diperoleh dengan mempartisi W . Demikian pula, tetapkan $Z_1, Z_2, \dots, Z_l$ merupakan urutan blok 128-bit (dengan kemungkinan pengecualian pada Z_l yang berisi v bit terakhir dari Z) yang diperoleh dengan mempartisi Z .

Dengan demikian, fungsi $G(H, W, Z)$ didefinisikan sebagai 128 bit nilai X_{k+l+1} , dengan X_i didefinisikan secara rekursif untuk $i = 0, 1, \dots, k + l - 1$, sebagai berikut:

- | | | |
|----|---|--|
| a) | $X_0 = 0^{128}$ | |
| b) | $X_i = (X_{i-1} \oplus W_i) \cdot H$ | $1 \leq i \leq k-1$ (this step is omitted if $k \leq 1$). |
| c) | $X_k = (X_{k-1} \oplus (W_k 0^{128-u})) \cdot H$ | (this step is omitted if $k = 0$). |
| d) | $X_i = (X_{i-1} \oplus Z_{i-k}) \cdot H$ | $k+1 \leq i \leq k+l-1$ (this step is omitted if $l \leq 1$). |
| e) | $X_{k+l} = (X_{k+l-1} \oplus (Z_l 0^{128-v})) \cdot H$ | (this step is omitted if $l = 0$). |
| f) | $X_{k+l+1} = (X_{k+l} \oplus [\#_{64}(\text{len}(W)) \#_{64}(\text{len}(Z))]) \cdot H$ | |

4. Prosedur Enkripsi

Originator harus melakukan langkah-langkah berikut untuk melindungi *string* data D dan memastikan integritas AAD (A):

- a) A variable length starting variable S shall be selected. This value shall be distinct for every message to be protected, and shall be made available to the recipient of the message. However, it is not necessary that this value be unpredictable or secret.
NOTE The value S can, for example, be generated using a counter maintained by the originator, and sent in clear text along with the protected message.
- b) Partition D into a sequence of 128 bit blocks: $D_1, D_2, \dots, D_m$, as follows. Let D_1 contain the first 128 bits of D , D_2 the next 128 bits, and so on, until D_m contains the final r bits, where $0 < r \leq 128$. Thus, D contains a total of $(m-1)n + r$ bits.
- c) Let $H = e_K(0^{128})$.
- d) If $\text{len}(S) = 96$ then let $Y_0 = S || 0^{31} || 1$. Otherwise let $Y_0 = G(H, \{\}, S)$.
- e) For $i = 1, 2, \dots, m-1$, perform the following two steps:
 - 1) Let $Y_i = \text{inc}(Y_{i-1})$;
 - 2) Let $C_i = D_i \oplus e_K(Y_i)$.
- f) Let $Y_m = \text{inc}(Y_{m-1})$.
- g) Let $C_m = D_m \oplus (e_K(Y_m))|_r$.
- h) Let $T = (G(H, A, C_1 || C_2 || \dots || C_m \oplus e_K(Y_0)))|_t$.

The output of the above process, i.e. the authenticated-encrypted version of D , shall be the bit-string:

$$C = C_1 || C_2 || \dots || C_m || T$$

That is, a string of $(m-1)n + r + t$ bits, that is C contains precisely t bits more than D (although it is also necessary to convey the variable length starting variable

S and the variable length additional authenticated data A to the recipient).

5. Prosedur Dekripsi

Recipient harus melakukan langkah-langkah berikut untuk mendekripsi dan memverifikasi *string* terenkripsi-otentikasi C , serta untuk memverifikasi AAD A .

- a) If the length of C is less than t , then halt and output INVALID.
- b) Let m and r be the unique integers defined so that $\text{len}(C) = (m-1)n + r + t$, where $0 < r \leq n$. Partition C into a sequence of blocks: $C_1, C_2, \dots, C_m, T$ as follows. Let C_1 contain the first n bits of C , C_2 the next n bits of C , and so on, until C_m contains the next r bits of C . Finally, let T be the final t bits of C .
- c) Let $H = e_K(0^{128})$
- d) If $\text{len}(S) = 96$ then let $Y_0 = S || 0^{31} || 1$. Otherwise, let $Y_0 = G(H, \{, S)$.
- e) Let $T' = (G(H, A, C_1 || C_2 || \dots || C_m) \oplus e_K(Y_0))|_t$
- f) If $T \neq T'$, then halt and output INVALID
- g) For $i = 1, 2, \dots, m-1$, perform the following two steps:
 - 1) let $Y_1 = \text{inc}(Y_{i-1})$;
 - 2) let $D_i = C_i \oplus e_K(Y_i)$.
- h) Let $Y_m = \text{inc}(Y_{m-1})$.
- i) Let $D_m = C_m \oplus (e_K(Y_m))|_r$.
- j) Output D and the additional authenticated data A .

F. Mekanisme AEAD_CHACHA20_POLY1305

Kriptografi modern didominasi oleh AES karena performa tinggi dan dukungan luas dari perangkat keras. Namun, AES memiliki keterbatasan, terutama pada perangkat yang tidak dilengkapi akselerasi

perangkat keras dan potensi kerentanan terhadap serangan *cache-timing*. Untuk mengatasi keterbatasan tersebut, Daniel J. Bernstein [18] memperkenalkan keluarga *stream cipher* Salsa20 dan variannya, ChaCha20, yang lebih cepat dalam implementasi perangkat lunak. Sementara itu, Poly1305 dikembangkan sebagai MAC yang efisien, aman, dan sederhana.

RFC 8439 [19] mendefinisikan ChaCha20, Poly1305, dan kombinasinya dalam skema AEAD yang dikenal sebagai ChaCha20-Poly1305 sehingga dituliskan menjadi AEAD_CHACHA20_POLY1305. Input untuk AEAD_CHACHA20_POLY1305 adalah:

- Kunci 256-bit
- *Nonce* 96-bit — berbeda untuk setiap pemanggilan dengan kunci yang sama
- *Plaintext* dengan panjang sebarang
- AAD dengan panjang sebarang

Beberapa protokol mungkin memiliki input unik per pemanggilan yang panjangnya bukan 96 bit. Misalnya, IPsec dapat menentukan *nonce* 64-bit. Dalam kasus seperti itu, dokumen protokol harus mendefinisikan bagaimana cara mengubah *nonce* protokol menjadi *nonce* 96-bit, misalnya dengan menggabungkannya dengan suatu nilai konstan.

Primitif ChaCha20 dan Poly1305 digabungkan menjadi sebuah AEAD yang menerima kunci 256-bit dan *nonce* 96-bit sebagai berikut:

- Pertama, kunci sekali-pakai (*one-time key*) untuk Poly1305 dibangkitkan dari kunci 256-bit dan *nonce* menggunakan prosedur yang dijelaskan pada Bagian 2.6 RFC 8439.
- Selanjutnya, fungsi enkripsi ChaCha20 dipanggil untuk mengenkripsi *plaintext*, menggunakan kunci dan *nonce* yang sama, dengan *counter* awal diset ke 1.

- Terakhir, fungsi Poly1305 dipanggil dengan kunci Poly1305 yang dihitung sebelumnya, dan sebuah pesan yang dibentuk sebagai konkatenasi dari:
 - AAD
 - *padding1* — *padding* hingga 15 *byte* nol, untuk membuat total panjang menjadi kelipatan 16. Jika panjang AAD sudah merupakan kelipatan 16 *byte*, maka field ini bernilai nol.
 - *Ciphertext*
 - *padding2* — *padding* hingga 15 *byte* nol, untuk membuat total panjang menjadi kelipatan 16. Jika panjang *ciphertext* sudah merupakan kelipatan 16 *byte*, maka field ini bernilai nol.
 - Panjang AAD dalam satuan oktet (sebagai integer 64-bit *little-endian*).
 - Panjang *ciphertext* dalam satuan oktet (sebagai integer 64-bit *little-endian*).

Output dari AEAD adalah konkatenasi dari:

- *Ciphertext* dengan panjang yang sama dengan *plaintext*.
- *Tag* 128-bit, yaitu *output* dari fungsi Poly1305.

Proses dekripsi serupa, dengan perbedaan berikut:

- Peran *ciphertext* dan *plaintext* dibalik, sehingga fungsi enkripsi ChaCha20 diterapkan pada *ciphertext* untuk menghasilkan *plaintext*.
- Fungsi Poly1305 tetap dijalankan pada AAD dan *ciphertext*, bukan *plaintext*.
- *Tag* yang dihitung dibandingkan secara *bitwise* dengan *tag* yang diterima. Pesan hanya diautentikasi jika dan hanya jika kedua *tag* tersebut sama.

Beberapa catatan mengenai AEAD_CHACHA20_POLY1305 adalah sebagai berikut:

1. Jumlah data yang dapat dienkripsi dalam satu pemanggilan adalah sebanyak $2^{32} - 1$ blok, masing-masing berukuran 64 *byte*, karena keterbatasan ukuran *block counter field* pada fungsi blok ChaCha20. Proses ini menghasilkan total 274.877.906.880 *byte* (hampir 256 GB). Jumlah ini memadai untuk protokol lalu lintas seperti IPsec dan TLS, namun mungkin terlalu kecil untuk enkripsi file dan/atau disk. Untuk penggunaan tersebut, rancangan asli dapat digunakan kembali, dengan mengurangi *nonce* menjadi 64 bit dan menggunakan integer pada posisi 13 sebagai 32 bit teratas dari *counter* blok 64 bit, sehingga ukuran pesan total meningkat hingga lebih dari satu juta *petabyte* (tepatnya 1.180.591.620.717.411.303.360 *byte*).
2. Terlepas dari hal di atas, panjang *field ciphertext* pada konstruksi *buffer* yang dijalankan Poly1305 membatasi ukuran *ciphertext* (dan *plaintext*) hingga 2^{64} *byte*, yaitu enam belas ribu *petabyte* (tepatnya 18.446.744.073.709.551.616 *byte*).

Berikut adalah daftar parameter untuk konstruksi AEAD_CHACHA20_POLY1305:

- K_LEN (panjang kunci) = 32 oktet.
- P_MAX (ukuran maksimum *plaintext*) = 274.877.906.880 *byte*, atau hampir 256 GB.
- A_MAX (ukuran maksimum *associated data*) = $2^{64}-1$ oktet, sesuai panjang *field* untuk AAD.
- N_MIN = N_MAX = 12 oktet.
- C_MAX = P_MAX + panjang *tag* = 274.877.906.896 oktet.

Input AAD yang berbeda harus digabungkan menjadi satu input ke AEAD_CHACHA20_POLY1305. Pembentukan struktur pada input AAD, bila diperlukan, merupakan tanggung jawab aplikasi.

Pseudocode untuk AEAD_CHACHA20_POLY1305 adalah sebagai berikut:

```

pad16(x) :
    if (len(x) % 16) == 0
        then return NULL

        else return copies(0, 16-(len(x)%16))
    end

chacha20_aead_encrypt(aad, key, iv, constant,
    plaintext):
    nonce = constant | iv
    otk = poly1305_key_gen(key, nonce)
    ciphertext = chacha20_encrypt(key, 1, nonce,
    plaintext)
    mac_data = aad | pad16(aad)
    mac_data |= ciphertext | pad16(ciphertext)
    mac_data |= num_to_8_le_bytes(aad.length)
    mac_data |= num_to_8_le_bytes(ciphertext.length)
    tag = poly1305_mac(mac_data, otk)
    return (ciphertext, tag)

```

ChaCha20 dirancang untuk memberikan tingkat keamanan 256-bit. Sementara itu, mekanisme autentikasi Poly1305 dirancang agar pesan palsu ditolak dengan probabilitas $1 - (n/2^{102})$ untuk pesan sepanjang $16n$ -byte, bahkan setelah 2^{64} pesan sah terkirim. Dengan demikian, Poly1305 bersifat *strong unforgeability against chosen-message attacks* (SUF-CMA) menurut terminologi AE. Hal-hal yang perlu diperhatikan dalam implementasi AEAD_CHACHA20_POLY1305 adalah sebagai berikut.

1. Keunikan Nonce

Aspek paling penting dalam implementasi ChaCha20 adalah keunikan *nonce*. Penggunaan *counter* maupun *linear-feedback shift register* (LFSR) merupakan metode yang dapat diterima untuk

membangkitkan *nonce* unik. Alternatif lain adalah melakukan enkripsi terhadap sebuah *counter* menggunakan *block cipher* berukuran blok 64 bit, seperti DES.

Sebaliknya, penggunaan hasil pemotongan (*truncation*) dari *counter* yang dienkripsi dengan *block cipher* berukuran blok 128 bit atau 256 bit dinyatakan tidak dapat diterima, karena hasil tersebut berpotensi mengalami pengulangan dalam jangka waktu singkat.

Apabila sebuah *nonce* digunakan kembali, maka kunci sekali-pakai Poly1305 dan *keystream* ChaCha20 akan identik pada pesan-pesan terkait. Kondisi ini menyebabkan operasi XOR antara *plaintext* dapat terekspose, karena XOR dari *plaintext* sama dengan XOR dari *ciphertext*.

2. Kunci Poly1305

Kunci Poly1305 wajib bersifat tidak dapat diprediksi oleh pihak penyerang. Pemenuhan persyaratan ini pada dasarnya dapat dilakukan dengan membangkitkan kunci secara acak. Namun demikian, karena Poly1305 lazim digunakan dalam protokol komunikasi, *recipient* harus mengetahui kunci yang digunakan.

Oleh sebab itu, pemanfaatan pembangkit bilangan acak semu, misalnya melalui enkripsi sebuah *counter*, merupakan pendekatan yang dapat diterima. Alternatif lainnya adalah penggunaan algoritme ChaCha dengan kunci rahasia dan sebuah *nonce*.

3. Implementasi dalam *Constant-Time*

Algoritma ChaCha20-Poly1305 dirancang agar dapat diimplementasikan dengan waktu eksekusi konstan guna mencegah kerentanan terhadap *side-channel attacks*.

- ChaCha20: operasi yang digunakan meliputi penjumlahan, XOR, dan rotasi tetap. Seluruh operasi tersebut dapat dan sebaiknya diimplementasikan dalam waktu konstan. Akses terhadap *offset* pada *state* ChaCha maupun jumlah operasi yang dilakukan tidak

bergantung pada properti kunci, sehingga meniadakan potensi kebocoran informasi kunci melalui *cache misses*.

- Poly1305: operasi yang digunakan meliputi penjumlahan, perkalian, dan modulus pada bilangan lebih dari 128 bit. Operasi ini dapat dilakukan dalam waktu konstan, tetapi implementasi yang sederhana, misalnya dengan memanfaatkan *library* bilangan besar generik, tidak selalu menjamin sifat waktu konstan. Sebagai contoh, apabila operasi perkalian dipisahkan dari operasi modulus, hasil perhitungan dapat berada di bawah 2^{256} atau melebihi 2^{256} . Oleh karena itu, pengembang wajib berhati-hati dalam memilih implementasi agar tidak menimbulkan kerentanan akibat perbedaan waktu eksekusi.

4. Validasi *Tag*

Proses validasi autentikasi dilakukan melalui perbandingan bit demi bit antara *tag* yang dihitung dengan *tag* yang diterima. Dalam banyak kasus, *nonce* dan AAD tidak dianggap terpakai hingga sebuah pesan dinyatakan sah. Hal ini memungkinkan penyerang untuk mengirimkan beberapa pesan identik dengan *tag* berbeda hingga salah satunya lolos dari pemeriksaan.

Serangan ini sulit dilakukan jika penyerang harus mencoba seluruh kemungkinan *tag* sebanyak 2^{128} . Namun, jika waktu eksekusi pada operasi perbandingan *tag* memberikan informasi mengenai panjang prefiks yang identik, maka jumlah percobaan yang diperlukan dapat berkurang signifikan.

Oleh karena itu, implementasi dalam protokol *online* wajib menggunakan fungsi perbandingan dengan waktu konstan, bukan mengandalkan fungsi *library* yang lebih cepat namun tidak aman, seperti `memcmp()` pada bahasa C.

Selain itu, setiap protokol yang memanfaatkan algoritma ini wajib menyertakan *tag* autentikasi secara utuh untuk

meminimalkan peluang pemalsuan. Pemotongan *tag* (*tag truncation*) tidak diperkenankan.

G. Mekanisme EAX Prime (EAX')

Standar ANSI C12.22/IEEE Std 1703 yang digunakan dalam *Advanced Metering Infrastructure* (AMI) dan *Supervisory Control and Data Acquisition* (SCADA) memerlukan mekanisme enkripsi dan autentikasi yang mampu menangani pesan yang sangat besar, sangat kecil, maupun repetitif, serta mendukung format alamat pesan yang hampir tidak terbatas panjangnya. Tantangan ini tidak dapat dipenuhi secara optimal oleh mode AE seperti CCM dan GCM.

Mode CCM yang awalnya dikembangkan untuk IEEE 802.11, memiliki keterbatasan panjang *nonce* maksimum 13 byte, sifat *nonce* yang dapat diprediksi, serta hanya berlaku untuk AES-128. Sementara itu, GCM walaupun efisien dan paralel, namun terdapat kendala dalam konteks *on-line processing* dengan panjang alamat pesan tak terbatas, serta berisiko ketika *nonce* tidak unik [20].

Untuk menjawab kebutuhan tersebut, dipilihlah EAX sebagai basis, karena memiliki sifat *on-line*, mendukung *associated data*, sederhana, dan terbukti aman secara formal [2]. Namun, agar lebih efisien digunakan dalam perangkat tertanam berdaya terbatas yang umum dalam AMI/SCADA, EAX kemudian dimodifikasi menjadi EAX'.

Modifikasi ini berupa sejumlah optimisasi, antara lain pengurangan jumlah pemanggilan AES per pesan, pengurangan kebutuhan memori per kunci, serta penghapusan langkah-langkah autentikasi yang tidak menambah nilai keamanan. Analisis formal menunjukkan bahwa EAX' mempertahankan kekuatan kriptografi setara dengan EAX, namun lebih sesuai dengan konteks operasional C12.22 [20].

Penyederhanaan pada EAX' didorong oleh keinginan untuk mengurangi jumlah enkripsi blok AES yang diperlukan oleh EAX, serta untuk mengurangi jumlah penyimpanan terkait kunci yang dibutuhkan pada implementasi yang dioptimalkan untuk kecepatan (*time-optimized*

implementation), tanpa secara signifikan melemahkan kekuatan kriptografi dan ketahanan terhadap serangan yang ditawarkan oleh EAX.

Optimisasi pada EAX' mengurangi kebutuhan penyimpanan per kunci untuk implementasi yang dioptimalkan waktu, dari semula $K + 4B + 2T$ byte per kunci (88 *byte* ketika AES-128 digunakan sebagai *block cipher*) menjadi $K + 2B$ byte per kunci (48 *byte* ketika AES-128 digunakan sebagai *block cipher*). Di sini:

- K adalah ukuran kunci dari *block cipher* yang digunakan,
- B adalah ukuran blok dari *block cipher* tersebut, dan
- T adalah ukuran tag autentikasi yang diinginkan, semuanya dalam satuan *byte*.

Untuk implementasi yang dioptimalkan ruang (*space-optimized implementation*), di mana hanya diperlukan K byte penyimpanan per kunci, optimisasi ini menghilangkan kebutuhan akan tiga kali pemanggilan AES tambahan per pesan ketika digunakan untuk enkripsi, atau lima kali pemanggilan AES tambahan ketika sedang tidak digunakan.

Block cipher seperti AES-128 merupakan *keyed pseudo-random permutations (PRP)*, yang memetakan input sebesar ukuran blok (misalnya 16 *byte* = 128 bit) menjadi *output* dengan ukuran blok yang sama. Kekuatan dan kemampuan untuk menahan kriptanalisis dari mode kriptografi komposit yang menggunakan *block cipher* ini berhubungan langsung dengan kekuatan dan ketahanan PRP dasar tersebut. Analisis atas suatu mode baru harus meninjau dampaknya dalam meningkatkan atau mengurangi kekuatan tersebut, serta kemungkinan jalur serangan kriptanalisis baru yang dapat terekspos akibat penggunaan mode baru tersebut.

Pertimbangan terhadap bukti keamanan EAX, yang dipublikasikan dalam [2], menunjukkan bahwa bukti tersebut dapat dimodifikasi untuk mengakomodasi optimisasi-optimisasi ini. Dengan demikian, sifat keamanan EAX' adalah ekuivalen dengan EAX. Berikut adalah perbedaan

antara EAX' dari EAX beserta alasan dasarnya untuk setiap perubahan tersebut.

1. Dalam EAX', perhitungan **CMAC atas string kosong (null string)** ketika *header* bernilai *null* dihapuskan karena hasilnya hanya berupa konstanta bergantung kunci yang tidak menambah ketahanan kriptografi. Analisis menunjukkan bahwa penyertaannya tidak meningkatkan resistensi terhadap serangan, sehingga pengecualiannya tidak melemahkan keamanan. Sebaliknya, langkah ini mengurangi jumlah pemanggilan AES per pesan dan menjadikan EAX' lebih efisien dibandingkan EAX asli.
2. Dalam EAX', terdapat **pengecualian input ciphertext tag ketika string plaintext kosong (null plaintext string)** artinya jika input EAX berupa *plaintext* kosong, maka perhitungan CMAC atas gabungan nilai terkait tidak dilakukan. Langkah ini diterapkan dalam EAX' demi efisiensi, sebab hasil perhitungan tersebut pada dasarnya hanya berupa konstanta bergantung kunci yang tidak menambah ketahanan kriptografi. Dengan demikian, pengecualian ini tidak melemahkan keamanan, melainkan menyederhanakan proses dan mengurangi jumlah operasi AES yang harus dijalankan.
3. Dalam EAX' dilakukan **penyederhanaan proses incrementation nonce pada mode CTR** dengan memaksa dua bit awal *nonce* bernilai nol untuk menghindari masalah *inter-word carries*. Perubahan ini memang sedikit mengurangi rata-rata jumlah pesan yang dapat dienkripsi sebelum terjadi penggunaan ulang *nonce*, namun tidak memengaruhi panjang maksimum pesan yang dapat dilindungi. Hal ini karena bit *nonce* lainnya tetap tidak dapat diprediksi penyerang, sehingga potensi penggunaan ulang *nonce* tidak dapat dieksploitasi. Dengan demikian, penyederhanaan ini meningkatkan efisiensi tanpa menurunkan ketahanan kriptanalisis.
4. Dalam EAX', menggunakan **alternatif nilai awal (initial) blok CMAC**, tidak harus selalu menggunakan konstanta $0^n, 0^{n-1}$ dan $,0^{n-2}$ sebagaimana pada EAX asli, melainkan dapat diganti dengan

konstanta lain selama masing-masing nilai awal berbeda satu sama lain. Syarat utamanya adalah setiap rangkaian CMAC harus dimulai dengan blok pertama yang saling terpisah, sehingga meskipun sisa input identik, hasil keluarannya tetap berbeda. Dengan cara ini, nilai awal CBC menjadi bergantung kunci dan tidak dapat diprediksi oleh penyerang. Karena itu, pilihan konstanta dalam EAX asli pada dasarnya bersifat *arbitrary*.

5. Dalam EAX', **pra-enkripsi *initial* blok CMAC** dilakukan dengan memasukkan nilai awal yang sudah dienkripsi kunci ke dalam proses CBC, menggantikan cara lama yang menggunakan blok konstanta yang diketahui. Secara matematis, pendekatan ini terbukti identik dengan metode CMAC asli, karena hasil enkripsi atau dekripsi kunci atas nilai awal dapat saling dipetakan tanpa memengaruhi keamanan. Dengan demikian, perubahan ini merupakan variasi dari mekanisme penggunaan konstanta berbeda (isu nomor 4), sehingga tidak mengubah sifat keamanan, tetapi menyederhanakan perhitungan dalam implementasi.
6. Dalam EAX', terdapat **penggunaan konstanta D dan Q sebagai *initial* blok CMAC**. Dalam perhitungan CMAC' bersyarat, blok pertama menggunakan *initial* yang diturunkan dari *output* mode CTR. Nilai ini adalah kelipatan konstan dari nilai awal blok pertama CMAC' lainnya, yang berasal dari PRP berbasis kunci. Pada dasarnya, kedua perhitungan CMAC' ini menggunakan *initial* berbeda agar *output* CBC pertama berbeda secara bergantung pada kunci dan tidak dapat diprediksi oleh penyerang. Perhitungan CMAC' selalu melibatkan CBC minimal dua blok, sehingga meskipun input berikutnya sama, hasil CBC berbeda karena *output* pertama berbeda. CMAC' kedua hanya dihitung jika ada *plaintext/ciphertext*, sehingga hasilnya selalu berbeda dari yang pertama. Optimisasi ini memungkinkan implementasi menyimpan 32 *byte* lebih sedikit per kunci tanpa mengurangi keamanan mode EAX.

EAX menerima input berupa *Cleartext* (N) dan opsional *Plaintext* (P), kemudian menghasilkan *output* berupa MAC (T) dan opsional *Ciphertext* (C) yang bersesuaian.

N : *Cleartext*. Bagian dari Pesan ANSI C12.22 yang diautentikasi tetapi tidak dienkripsi untuk *confidentialitas*.

P : *Plaintext*. Bagian dari Pesan ANSI C12.22 yang dienkripsi untuk menjaga *confidentialitas*.

C : *Ciphertext*. Bagian dari Pesan ANSI C12.22 setelah dienkripsi untuk *confidentialitas*, yang berkorespondensi langsung dengan teks asli (*Plaintext*) P .

T : 4 bytes MAC yang disisipkan di akhir Pesan ANSI C12.22.

K : Kunci rahasia kriptografis (*Cryptographic secret key*). Panjang kunci disesuaikan dengan ukuran *block cipher* yang digunakan. Nilai standar (*default value*) ditetapkan pada AES-128/EAX'. Mode EAX' mendukung penggunaan *block cipher* alternatif lainnya.

Notasi dan simbol yang digunakan dalam algoritma EAX' adalah sebagai berikut.

$X \leftarrow Y$	Y di-assign ke X
$X \& Y$	<i>Bitwise AND</i> antara X dan Y , dengan X, Y , dan hasilnya memiliki ukuran yang sama.
$X \text{ XOR } Y$	<i>Bitwise exclusive-OR (XOR)</i> antara X dan Y , dengan X, Y , dan hasilnya memiliki ukuran yang sama.
$X \text{ XOR}_{\text{end}} Y$	<i>Bitwise exclusive-OR (XOR)</i> dari Y setelah diselaraskan (<i>aligned</i>) dengan bagian akhir <i>string</i> X, dengan X lebih panjang dari Y, dan hasilnya memiliki ukuran yang sama dengan X. Operasi ini setara secara tepat dengan: $X \text{ XOR } (0^{\text{size}(X)-\text{size}(Y)} Y)$

$\text{size}(X)$	Panjang X dalam satuan bit.
$\text{AES}_K(X)$	Enkripsi dari X menggunakan algoritma AES dengan kunci K . X , K , dan hasil enkripsi semuanya memiliki panjang yang sama, yaitu n bit.
$X \parallel Y$	Konkatenasi (<i>concatenation</i>) atau penggabungan <i>string</i> X dan Y .
X [first m bits]	m bit paling signifikan (<i>most significant bit</i>) dari X
$\text{msb}(X)$	<i>Most significant bit</i> dari X .
$X \ll 1$	<i>Left shift</i> dari X sebanyak satu posisi ke kiri (dengan <i>carry-in</i> bernilai 0).
$\text{ceiling}(X)$	Fungsi pembulatan ke atas (<i>ceiling function</i>), yang menghasilkan bilangan bulat terkecil yang tidak lebih kecil dari X .
n	Ukuran blok dalam bit dari <i>block cipher</i> yang digunakan. (Untuk AES-128, $n = 128$).
0^m	m bit bernilai 0. (Misalnya, $0^4 10^3$ berarti 00001000).
1^m	m bit bernilai 1. (Misalnya, $1^4 01^2 0$ berarti 11110110).

Algoritma yang dijalankan dalam proses EAX' adalah sebagai berikut.

1. Algoritma $\text{dbl}(X)$

Pada algoritma ini, polinomial tak tereduksi berbobot minimal (*minimal weight irreducible polynomial*), r_b , didefinisikan sama persis seperti pada [15]

```
-- modulo the first minimal weight irreducible polynomial of
degree  $n$ 
define  $r_b$  as  $0^{120}10000111$       -- for  $n = 128$ 
if  $\text{msb}(X) = 0$       -- most significant bit
```

```

then return  $X \ll 1$ 
else return  $(X \ll 1) \text{ XOR } r_b$ 

```

2. Algoritma *deriveKeyDependentConstants(K)*

```

 $L \leftarrow AES_K(0^n)$ 
  -- For the AES-128 block cipher  $n = 128$ 
 $D \leftarrow dbl(L)$ 
  --  $D$  is double the value of  $L$  in an appropriate field,
  as
  defined by  $r_b$ 
 $Q \leftarrow dbl(D)$ 
  --  $Q$  is quadruple the value of  $L$ 
  --  $dbl(dbl(L))$ 
  -- in the same field
return  $D$  and  $Q$ 

```

3. Algoritma *pad(S, D, Q)*

```

if (size( $S$ ) > 0) and ( $size(S) \bmod n = 0$ ) --  $n$  equals Cipher
  block size (in bits)
then return  $S \text{ XOR}_{\text{end}} D$  -- XORs into end of string
else return  $(S || 10^{n-1-(size(S) \bmod n)}) \text{ XOR}_{\text{end}} Q$  -- XORs into end
  of string; applies to null
  string

```

4. Algoritma *CBC'_K(t, S)*

```

Let  $S_1 \dots S_m \leftarrow S$  where  $size(S_i) = n$  --  $n$  equals Cipher
  block size (in bits)

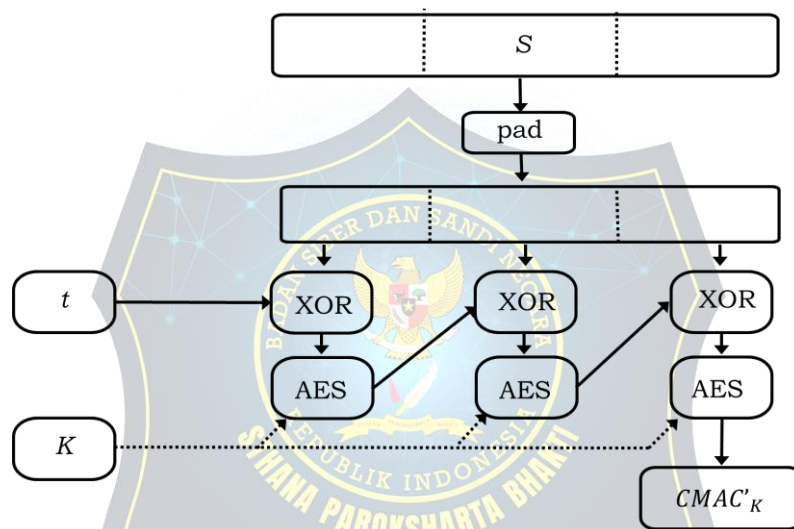
 $C_0 \leftarrow t$ 
for  $i \leftarrow 1$  to  $m$ 
  do  $C_i \leftarrow AES_K(S_i \text{ XOR } C_{i-1})$ 
return  $C_m$ 

```

Definisi CMAC' berikut berasal dari [15]. Definisi ini telah disusun ulang dengan nomenklatur yang serupa dengan yang digunakan dalam makalah EAX serta dengan sintaks yang digunakan oleh standar ANSI C12.22. Komputasi algoritma $CMAC'_K$ diilustrasikan pada Gambar 3.3.

5. Algoritma $CMAC'_K(t, S, D, Q)$

```
return  $CBC'_K(t, pad(S, D, Q))$ 
```



Gambar 3.3 Komputasi Algoritma $CMAC'_K$

Digambar ulang dari [20].

INTEGRATIF • SOLID • AKUNTABEL • INOVATIF

6. Algoritma $CTR'_K(N, S)$

Note: For this algorithm, the counter Ctr can be thought of as a single n -bit number represented as m bytes with the most significant byte placed first, (byte 0) and least significant byte placed last (byte $m-1$). By contrast, all the other functions that implement EAX' , including AES, are defined with the least significant byte placed first. This was done as an acknowledgement of extant existing practice in implementations of CTR mode as defined in [7].

```
 $m \leftarrow \text{ceiling}(\text{size}(S)/n)$ 
```

```
 $\text{Ctr} \leftarrow N \& 1^{n-32} 01^{15} 01^{15}$ 
```

```
-- Optimization, to avoid inter-word carries.
```

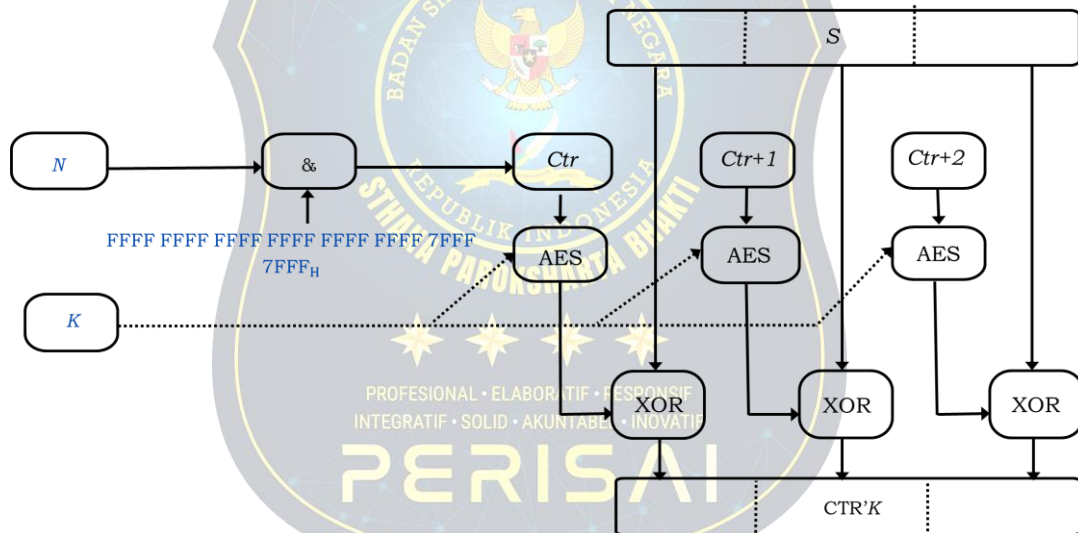
```
-- For AES,  $n = 128$  and  $1^{n-32} 01^{15} 01^{15}$  resolves to
```

```
-- FFFF FFFF FFFF FFFF FFFF FFFF 7FFF 7FFFH
```

```
 $\text{Pad} \leftarrow \text{AES}_K(\text{Ctr}) \parallel \text{AES}_K(\text{Ctr} + 1) \parallel \dots \parallel \text{AES}_K(\text{Ctr} + m - 1)$ 
```

```
return  $S \text{ XOR } \text{Pad}$  [first  $\text{size}(P)$  bits]
```

Komputasi algoritma CTR'_K diilustrasikan pada Gambar 3.4.



Gambar 3.4 Komputasi algoritma CTR'_K

Digambar ulang dari [20].

7. Algoritma $\text{EAX}'_K.\text{Encrypt}_K(N, P)$

```
deriveKeyDependentConstants( $K$ )
```

```
-- need be done only once per new key value
```

```
 $\text{Tag} \leftarrow \underline{N} \leftarrow \text{CMAC}_K(D, N, D, Q)$ 
```

```
-- EAX' optimization, first  $D$  used instead of  $E_K(0^n)$ 
```

```
if  $\text{size}(P) > 0$ 
```

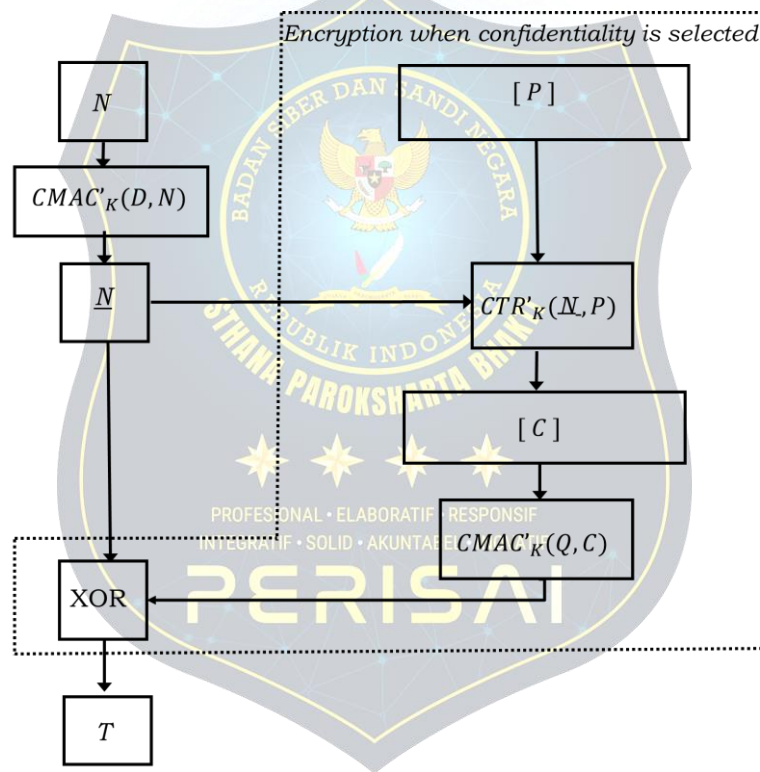
```
-- EAX' optimization, no contribution of a key-dependent  
constant for an empty  $C$ 
```

```

then
     $C \leftarrow \text{nullString}$ 
else
     $C \leftarrow \text{CTR}'_K(\underline{N}, P)$ 
 $\text{Tag} \leftarrow \text{Tag} \text{ XOR } \text{CMAC}'_K(Q, C, D, Q)$ 
    -- EAX' optimization, first Q used instead of  $E_K(0^{n-2}10)$ 
 $T \leftarrow \text{Tag}$  [first 32 bits]
return  $C$  and  $T$ 

```

Komputasi algoritma enkripsi EAX' diilustrasikan pada Gambar 3.5.



Gambar 3.5 Algoritma enkripsi EAX'
digambar ulang dari [20].

8. Algoritma $EAX'. \text{Decrypt}_K(N, C, T)$

```

deriveKeyDependentConstants(K)
    -- need be done only once per new key value
 $\text{Tag} \leftarrow \underline{N} \leftarrow \text{CMAC}'_K(D, N, D, Q)$ 
    -- EAX' optimization, first  $D$  used instead of  $E_K(0^n)$ 
if size(C) = 0 then

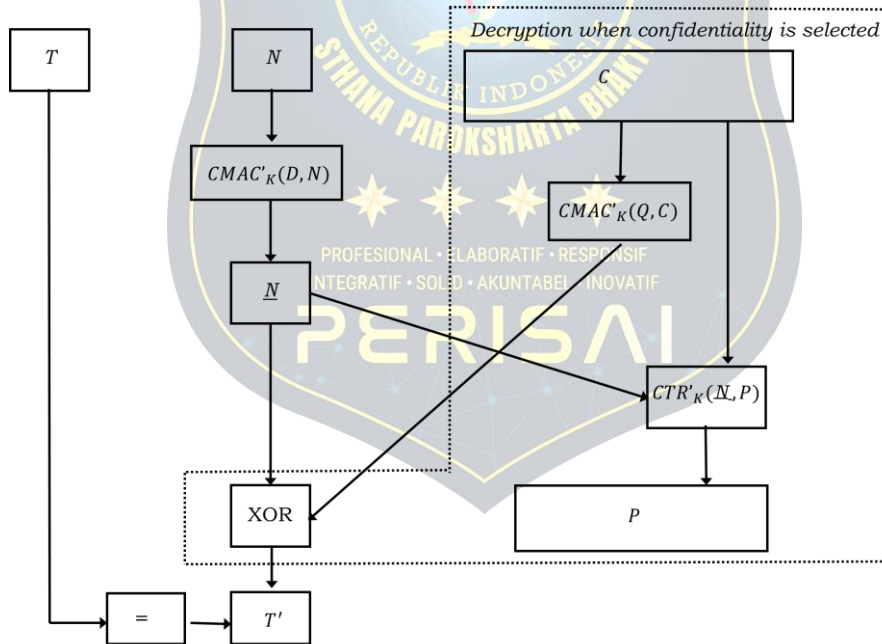
```

```

-- EAX' optimization, no contribution of a key-dependent
  constant for an empty  $C$ 
 $P \leftarrow \text{nullString}$ 
if  $T \neq \text{Tag}$  [first 32 bits] then
  return INVALID
else return  $P$                                 -- message is VALID
else
   $\text{Tag} \leftarrow \text{Tag} \text{ XOR } \text{CMAC}_K(Q, C, D, Q)$ 
  -- EAX' optimization, first  $Q$  used instead of  $\text{EK}(0n-210)$ 
  if  $T \neq \text{Tag}$  [first 32 bits] then return INVALID
   $P \leftarrow \text{CTR}'_K(\underline{N}, C)$ 
  --decrypt Ciphertext only when tags match
  return  $P$  -- message is VALID

```

Komputasi algoritma dekripsi EAX' diilustrasikan pada Gambar 3.6.



Gambar 3.6 Algoritma dekripsi EAX'

Digambar ulang dari [20].

BAB IV

PEMANFAATAN *AUTHENTICATED* *ENCRYPTION*



BAB IV

PEMANFAATAN AUTHENTICATED ENCRYPTION

AE dapat diterapkan dalam berbagai skenario untuk keperluan enkripsi dan autentikasi. Berikut adalah beberapa contoh penggunaan AE berdasarkan masing-masing mekanisme yang telah dijelaskan pada Bab III.

A. Mekanisme *Key Wrap*

1. AES *Key Wrap* (KW) dan AES *Key Wrap with Padding* (KWP)
 - a. AWS CloudHSM mengimplementasikan mekanisme *key wrap* baik menggunakan *padding* maupun tanpa *padding*. Fungsionalitas ini memungkinkan *wrapping* kunci simetris atau kunci lain secara aman dengan algoritma AES, sekaligus memastikan bahwa kunci dalam bentuk *plaintext* tidak pernah meninggalkan batas keamanan modul HSM. Dengan demikian, KW dalam CloudHSM berfungsi sebagai mekanisme penting untuk menjamin kerahasiaan dan integritas kunci kriptografi, serta memperkuat kepatuhan terhadap standar internasional dalam tata kelola keamanan kunci [21].
 - b. Azure *Key Vault* mendukung metode *Bring Your Own Key* (BYOK) untuk memungkinkan pelanggan mengimpor kunci mereka sendiri dari HSM lokal ke HSM Azure secara aman. Dalam proses ini, pelanggan terlebih dahulu menghasilkan *Key Exchange Key* (KEK) berupa kunci RSA di Azure *Key Vault*, yang hanya dapat digunakan untuk operasi impor. Selanjutnya, kunci target yang ada di HSM lokal dilindungi dengan kombinasi algoritma kriptografi, yaitu dienkripsi menggunakan kunci AES *ephemeral* dengan skema AES KWP, kemudian kunci AES tersebut di-*wrap* dengan KEK menggunakan RSA-OAEP. Hasil enkripsi ini membentuk *Key Transfer Blob* dalam format *JSON Web Encryption* (JWE), yang kemudian diunggah ke Azure *Key Vault*. Dengan mekanisme ini, kunci target tidak pernah muncul dalam bentuk *plaintext*, dan keamanan tetap terjaga sepanjang proses transfer ke infrastruktur Azure [22].

- c. Berdasarkan dokumen resmi Google Cloud KMS [23], proses impor kunci dilakukan dengan skema enkripsi hibrida yang mengombinasikan algoritma asimetris dan simetris. Pertama, Google Cloud menggunakan kunci publik dari *import job* untuk mengenkripsi kunci sementara AES-256 sekali pakai dengan algoritma RSA-OAEP. Selanjutnya, kunci AES-256 sekali pakai tersebut digunakan untuk *wrapping key material* target menggunakan KWP. Hasil dari kedua tahap ini digabungkan menjadi satu paket *wrapped key material* yang kemudian diunggah ke Google Cloud KMS. Dengan mekanisme ini, Google memastikan bahwa material kunci selalu dalam kondisi terenkripsi selama proses impor dan tidak pernah dapat diakses dalam bentuk *plaintext* di luar sistem yang berwenang, sehingga keamanan dan kerahasiaan kunci tetap terjaga.
- d. Dokumen IETF [24] menetapkan teknik untuk mengenkripsi perangkat lunak, perangkat keras, model *machine learning*, dan data personalisasi (*personalization data*) dengan memanfaatkan *IETF Software Update for Internet of Things (SUIT) manifest*. Mekanisme *key agreement* dapat dilakukan melalui *ephemeral-static* (ES) Diffie-Hellman (DH) dan KW. ES-DH menggunakan kriptografi kunci publik, sedangkan KW menggunakan kunci yang telah dibagikan sebelumnya (*pre-shared key*). Proses enkripsi terhadap *plaintext* dilakukan dengan kriptografi simetris konvensional.

KW digunakan untuk mengenkripsi *content-encryption key* (CEK) yang dihasilkan secara acak dengan *key-encryption key* (KEK). Konvensi COSE terkait penggunaan AES-KW ditetapkan dalam Bagian 8.5.2 [RFC9052] dan Bagian 6.2.1 [RFC9053]. CEK terenkripsi dibawa dalam struktur *COSE_recipient* yang memuat informasi yang diperlukan untuk menjalankan KW. Struktur *COSE_recipient*, yang merupakan sub-struktur dari *COSE_Encrypt*, berisi CEK yang telah dienkripsi dengan KEK. Untuk memastikan tingkat keamanan yang tinggi dalam penggunaan KW, KEK harus

memiliki entropi yang tinggi dan implementasi harus melindungi KEK dari kebocoran. Jika KEK *compromised* maka seluruh data yang dienkripsi dengannya, termasuk biner dan data konfigurasi, berpotensi terbuka. Struktur *COSE_Encrypt* menyampaikan informasi yang diperlukan untuk mengenkripsi *payload*, termasuk detail algoritma dan IV. Meskipun *payload* dapat dikirim sebagai konten terpisah (*detached content*), informasi enkripsinya tetap tertanam di dalam struktur *COSE_Encrypt.ciphertext*.

Terdapat tiga opsi *deployment* untuk penggunaan KW dalam enkripsi *payload*:

- 1) Jika semua *recipient* berbagi KEK yang sama, maka hanya ada satu struktur *COSE_recipient* yang berisi CEK terenkripsi. *Originator* menjalankan langkah-langkah berikut:
 - a) mengambil KEK tunggal yang digunakan bersama oleh semua *recipient*.
 - b) membuat CEK baru yang akan dipakai untuk mengenkripsi *payload*.
 - c) CEK dienkripsi dengan KEK tunggal, lalu hasilnya dimasukkan ke dalam struktur *COSE_recipient*
 - d) *Payload* dienkripsi menggunakan CEK.

Opsi ini sangat tidak dianjurkan. Karena apabila penyerang berhasil memperoleh KEK, maka ia dapat mengenkripsi dan mengirim *payload* kepada semua penerima yang dikonfigurasi menggunakan KEK tersebut.

- 2) Jika setiap *recipient* memiliki KEK yang berbeda, maka beberapa struktur *COSE_recipient* disertakan tetapi hanya satu CEK yang digunakan. Setiap *COSE_recipient* berisi CEK yang dienkripsi dengan KEK sesuai untuk *recipient* tertentu. Keuntungan dari pendekatan ini adalah *payload* hanya dienkripsi sekali dengan CEK, sementara KEK tidak dibagi antar penerima. Dengan

demikian, penerima yang sah tetap menggunakan KEK masing-masing untuk mendekripsi CEK dan kemudian memperoleh *plaintext*.

Langkah yang dilakukan *originator* yaitu:

- a) membuat CEK tunggal yang akan dipakai bersama untuk mengenkripsi *payload*.
 - b) Untuk setiap *recipient Ri*, *originator* mengambil KEK khusus milik *recipient* tersebut.
 - c) CEK yang sama tadi di-*wrap* dengan KEK masing-masing *recipient*.
 - d) *Payload* hanya dienkripsi sekali dengan CEK yang sama.
- 3) Menggunakan CEK yang berbeda-beda, masing-masing dienkripsi dengan KEK *recipient* yang berwenang. Pendekatan ini dipilih misalnya dalam situasi di mana distribusi *firmware* atau data sangat sensitive yang benar-benar harus dipisahkan antar *recipient*, sehingga kebocoran satu CEK/KEK tidak akan memengaruhi penerima lain.

Maka *originator* perlu melakukan langkah-langkah berikut:

- a) Untuk setiap *recipient Ri*, *originator* mengambil KEK unik *recipient* tersebut.
- b) menghasilkan CEK baru yang unik hanya untuk *recipient* tersebut.
- c) CEK *recipient* di-*wrap* dengan KEK miliknya.
- d) *Payload* untuk *recipient* tersebut dienkripsi dengan CEK yang baru saja dihasilkan.

2. Triple DES Encryption Algorithm Key Wrap (TKW)

Berdasarkan dokumen *IBMJCECCA z/OS Hardware Cryptography Reference Guide* [25], penerapan TKW direalisasikan melalui algoritma *Triple DES Encryption Algorithm* (TDEA) yang digunakan untuk

wrapping dan *unwrapping* kunci dalam skema “TR-31 key block”. Mekanisme ini memungkinkan kunci operasional TDEA, seperti *IMPORTER transport key*, di-wrap dengan aman menggunakan *EXPORTER transport key* TDEA sehingga dapat ditransfer atau disimpan tanpa mengekspos nilai kunci dalam bentuk *plaintext*. Proses *wrapping* dan *unwrapping* dijalankan dengan TDEA sehingga menjamin bahwa kunci hanya dapat diakses oleh entitas berwenang dalam lingkungan HSM atau modul kriptografi IBM. Dengan cara ini, IBM memastikan interoperabilitas dan keamanan pada sistem *legacy* yang masih menggunakan TDEA, sekaligus menjaga kesesuaian dengan praktik industri perbankan dan regulasi standar internasional.

B. Mekanisme CCM

1. AES_CCM pada JSON Web Tokens (JWT) dengan JSON Web Encryption (JWE)

JWT adalah format representasi klaim yang sering digunakan dalam autentikasi dan otorisasi dalam aplikasi web. Dalam beberapa kasus, JWT dapat dienkripsi menggunakan JWE. Melalui penggunaan AE dengan JWE, *payload* JWT dienkripsi dan diautentikasi sebelum dikirimkan. Hal ini bertujuan untuk memastikan bahwa informasi autentikasi yang dikirimkan dalam token tidak hanya terlindungi dari akses yang tidak sah tetapi juga tidak dapat dimanipulasi, sehingga menjaga integritas dan kepercayaan dalam proses autentikasi entitas.

JWT dikodekan sebagai objek JSON yang digunakan sebagai *payload* dalam struktur *JSON Web Signature* (JWS) atau sebagai *plaintext* dalam struktur JWE. JWT dapat ditandatangani secara digital menggunakan JWS untuk memastikan integritasnya dengan MAC, atau dienkripsi menggunakan JWE untuk menjaga konfidensialitasnya. JWT direpresentasikan menggunakan JWS *Compact Serialization* atau JWE *Compact Serialization*.

Dalam JWT, objek JSON yang disebut sebagai *JWT Claim Set* berisi pasangan nama/nilai (*Claim Names* dan *Claim Values*). Klaim ini

dapat digunakan untuk autentikasi entitas dalam sistem. Sementara itu, *JSON Object Signing and Encryption (JOSE) Header* menentukan operasi kriptografi yang diterapkan pada *JWT Claims Set*. Jika *JOSE Header* digunakan untuk JWS, maka JWT akan ditandatangani secara digital atau diberi MAC. Sebaliknya, jika *JOSE Header* digunakan untuk JWE, maka JWT akan dienkrpsi sepenuhnya untuk menjaga keamanannya [26].

JWE adalah struktur data berbasis JSON yang merepresentasikan konten yang telah dienkrpsi dan dikodekan menggunakan base64url. Struktur ini terdiri dari beberapa elemen, yaitu *JOSE header*, *JWE encrypted key*, *JWE initialization vector (IV)*, *JWE AAD*, *JWE ciphertext*, dan *JWE authenticated tag*. JWE menggunakan AE untuk memastikan konfidensialitas dan integritas *plaintext* serta integritas *JWE Protected Header* dan *JWE AAD* [27].

JWE mengimplementasikan berbagai algoritma kriptografi untuk mengenkripsi atau menentukan *Content Encryption Key (CEK)*. Algoritma tersebut termasuk RSAES-PKCS1-v1_5, RSAES OAEP, AES *Key Wrap* dengan berbagai panjang kunci (128-bit, 192-bit, dan 256-bit), serta metode *key agreement* seperti *Elliptic Curve Diffie-Hellman (ECDH-ES)*. Selain itu, terdapat pula algoritma enkripsi konten seperti AES_128_CBC_HMAC_SHA_256, AES_192_CBC_HMAC_SHA_384, AES_256_CBC_HMAC_SHA_512, serta AES GCM dengan panjang kunci 128-bit, 192-bit, dan 256-bit [28]. Sebagai contoh, *JWT Claims Set* dapat dienkrpsi menggunakan JWE sebelum dikirim ke penerima dengan *JWT Claim Set* berikut [26]:

```
{ "iss": "joe",  
  "exp": 1300819380,  
  "http://example.com/is_root": true  
}
```

Dalam contoh ini, klaim "iss" (*issuer*) menunjukkan bahwa token diterbitkan oleh "joe", klaim "exp" (*expiration time*) menunjukkan waktu

kedaluwarsa token dalam format *Unix timestamp*, dan klaim "http://example.com/is_root" digunakan sebagai klaim kustom untuk menyatakan bahwa pengguna memiliki hak akses tertinggi.

Untuk melakukan enkripsi, digunakan algoritma RSAES-PKCS1-v1_5 untuk mengenkripsi CEK dan menghasilkan *JWE Encrypted Key*. Selain itu, AES_128_CBC_HMAC_SHA_256 digunakan untuk melakukan AE, menghasilkan *ciphertext* (*JWE Ciphertext*) serta tanda autentikasi (*JWE Authentication Tag*). *JWE Protected Header* dalam bentuk JSON adalah sebagai berikut:

```
{  "alg": "RSA1_5",  
  "enc": "A128CBC-HS256" }
```

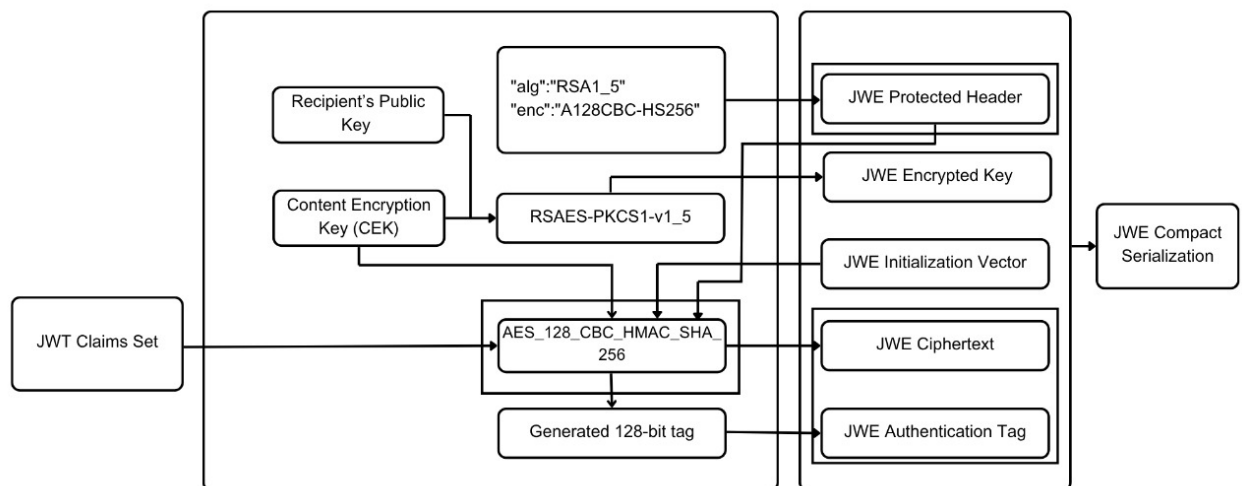
Selanjutnya mengonversi *JWE Protected Header* ini ke format BASE64URL(UTF8(*JWE Protected Header*)). Pembuatan JWE dimulai dengan membangkitkan CEK secara acak, lalu mengenkripsinya dengan kunci publik penerima menggunakan RSAES-PKCS1-v1_5 untuk menghasilkan *JWE Encrypted Key*, yang kemudian dikodekan dalam Base64URL. Setelah itu, JWE IV dibangkitkan secara acak dan dikodekan dalam Base64URL.

Selanjutnya, AAD ditetapkan sebagai hasil *encoding* dari *JWE Protected Header* dalam Base64URL. *JWT Claims Set* dienkrpsi menggunakan AES_128_CBC_HMAC_SHA_256, dengan CEK sebagai kunci, IV sebagai parameter tambahan, dan AAD untuk memastikan integritas. Proses ini menghasilkan *ciphertext* dan *Authentication Tag* 128-bit, yang dikodekan dalam Base64URL. Terakhir, elemen-elemen ini disusun dalam format *JWE Compact Serialization*: BASE64URL(*JWE Protected Header*). BASE64URL(*JWE Encrypted Key*). BASE64URL(*JWE Initialization Vector*). BASE64URL(*JWE Ciphertext*). BASE64URL(*JWE Authentication Tag*). Hasil akhir dalam contoh ini adalah:

```
eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU  
2In0. (Protected Header)
```

AxY8DCtDaGlslbGljb3RoZQ. (Initialization Vector)
MK0le7UQrG6nSxTLX6Mqwt0orbHvAKeWnDYvpIAeZ72deHxz3roJDxQyhxx0wKaMHDjUEOKIwrtkHthpqEanSBNYHZgmNOV7sl
n1Eu9g3J8. (Ciphertext)
fik51VwhsxJ-siBMR-YFiA (Authentication Tag)

Proses pembuatan JWE tersebut digambarkan dalam Gambar x dengan bagian yang ditandai dengan kotak merah menunjukkan komponen yang berperan dalam AE.



Gambar 4.1 Proses Pembuatan JWE

Diadaptasi dari [29].

Dengan menggunakan metode ini, JWT dienkripsi menggunakan AE untuk menjamin kerahasiaan, keutuhan, dan validitas data autentikasi. Selain mencegah akses tidak sah, metode ini juga

memungkinkan penerima memverifikasi keaslian token melalui *authentication tag*, sehingga memastikan bahwa token berasal dari sumber yang terpercaya dan tidak mengalami manipulasi. Dengan demikian, hanya entitas yang sah yang dapat mengakses dan memvalidasi token, sehingga meningkatkan keamanan dalam sistem autentikasi dan otorisasi.

C. Mekanisme EAX dan EAX'

Mekanisme EAX memang tidak dipakai secara langsung dalam produk atau standar protokol populer. Walaupun jarang diadopsi di standar global besar, EAX tetap dimanfaatkan dalam beberapa ruang lingkup tertentu, yaitu:

1. Pada standar ANSI C12.22 yang digunakan untuk protokol antarmuka dengan jaringan komunikasi data, diterapkan mode sandi EAX' (EAX *prime*) sebagai mekanisme pengamanan [30]. Pemanfaatan EAX' ini dilatarbelakangi oleh kebutuhan khusus dalam pertukaran pesan *Supervisory Control and Data Acquisition* (SCADA), khususnya pada sistem *Automated Meter Reading* (AMR) yang beroperasi dalam infrastruktur *Advanced Metering Infrastructure* (AMI). Meskipun awalnya dirancang untuk keperluan tersebut, EAX' juga dapat diterapkan pada berbagai perangkat tertanam (*embedded devices*) berskala kecil yang berkomunikasi dalam lingkungan SCADA. Mekanisme EAX' dipilih dibandingkan CCM, karena memiliki fleksibilitas lebih tinggi dalam penanganan pesan dengan panjang variabel dan struktur *header* yang tidak terenkripsi, serta tidak memerlukan pemrosesan ulang *nonce* yang ketat seperti pada CCM. Selain itu, EAX' mampu memberikan autentikasi dan kerahasiaan data secara efisien dalam satu proses enkripsi, tanpa memerlukan perubahan besar pada perangkat keras atau perangkat lunak yang sudah ada. Struktur EAX' yang sederhana namun kuat secara kriptografis juga menjadikannya sesuai untuk sistem dengan sumber daya terbatas dan mudah diintegrasikan ke dalam sistem komunikasi data berstandar

- industri. Dengan demikian, penggunaan EAX' pada ANSI C12.22 dapat memastikan keautentikan, integritas, dan konfidensialitas pesan yang ditransmisikan antarperangkat dalam jaringan AMI [20].
2. Selain diimplementasikan dalam standar ANSI C12.22 untuk protokol antarmuka dengan jaringan komunikasi data, mode sandi EAX' juga mendapat dukungan pada tingkat perangkat keras. IP Cores, Inc. merilis core IP AES seri GCE1 yang mendukung mode enkripsi EAX', serta mode lain seperti GCM dan CCM, yang memungkinkan para pembuat *chip* untuk mengintegrasikan enkripsi dan autentikasi ke dalam *embedded system* yang digunakan dalam remote sensing, protokol industri, atau *smart meter*. Dengan adanya dukungan pada tingkat perangkat keras, EAX' dapat dimanfaatkan secara optimal untuk memberikan perlindungan data *end-to-end* sekaligus menjaga kinerja sistem pada perangkat dengan sumber daya terbatas [31].
 3. Pemanfaatan mekanisme EAX telah didukung pada tingkat *library* kriptografi perangkat lunak. Salah satu contohnya adalah wolfCrypt, bagian dari *library* keamanan wolfSSL, yang menambahkan dukungan terhadap AES-EAX, yaitu skema *authenticated encryption* (AE) dua tahap (*two-pass*) yang sederhana dan efisien sebagaimana dijelaskan oleh Bellare, Rogaway, dan Wagner [2]. EAX menyediakan mekanisme enkripsi dan autentikasi terintegrasi dengan pendekatan *encrypt-then-authenticate*, sehingga memudahkan penerapan keamanan data tanpa perlu mengelola dua proses terpisah. Dalam wolfCrypt, pengguna dapat mengaktifkannya dengan *flag* `--enable-aeseax`, serta melakukan enkripsi/otentikasi dalam satu langkah praktis (*encrypt-then-authenticate*) melalui API seperti `wc_AesEaxEncryptAuth` dan `wc_AesEaxDecryptAuth` [32].
 4. Dalam *library* Crypto++ dan BouncyCastle, EAX disediakan sebagai salah satu mode AEAD, yang memungkinkan proses enkripsi dan autentikasi data dilakukan secara terpadu. Implementasi ini mengikuti konsep *encrypt-then-authenticate* sebagaimana

didefinisikan oleh Bellare, Rogaway, dan Wagner. Di Crypto++, EAX dapat diakses melalui kelas `EAX_Mode`, yang mendukung algoritma *block cipher* seperti AES, Camellia, Serpent, atau Twofish untuk memberikan fleksibilitas bagi pengembang untuk menyesuaikan algoritma sesuai kebutuhan keamanan dan performa sistem [33]. Sementara itu, BouncyCastle—yang banyak digunakan di lingkungan Java dan .NET—menyediakan dukungan serupa melalui kelas `EAXBlockCipher`, memungkinkan integrasi mudah dalam aplikasi lintas platform [34]. Pemanfaatan EAX pada kedua *library* ini memperlihatkan bahwa EAX tidak hanya relevan untuk implementasi industri tertentu, tetapi juga menjadi opsi praktis dan serbaguna bagi pengembang yang membutuhkan AE dengan keamanan formal yang kuat dan kompatibilitas luas di berbagai platform perangkat lunak.

5. Mekanisme EAX juga telah diimplementasikan dalam ekosistem bahasa pemrograman Rust [35], yang dikenal dengan fokusnya pada keamanan memori dan efisiensi sistem. Salah satu *library* (*crate*) yang mengimplementasikan mode ini adalah `eax`, bagian dari ekosistem RustCrypto AEAD, yang merealisasikan EAX sebagai kombinasi AES-CTR dan CMAC untuk mencapai autentikasi dan enkripsi terintegrasi. `Crate eax` dirancang mengikuti prinsip AEAD dan menyediakan API yang sederhana namun fleksibel. `Crate eax` mendukung operasi *in-place*, sehingga proses enkripsi dan dekripsi dapat dilakukan langsung pada *buffer* data tanpa alokasi tambahan di memori—fitur ini sangat berguna untuk sistem dengan sumber daya terbatas seperti *embedded devices* atau aplikasi *real-time*. Selain itu, *library* ini juga memungkinkan penggunaan *nonce* dan pengindeksan tag autentikasi yang fleksibel, serta mendukung lingkungan tanpa heap (*heapless*). Pemanfaatan EAX dalam Rust melalui `crate eax` menunjukkan bahwa mode ini tetap relevan dan adaptif di era bahasa pemrograman modern yang menekankan keamanan tingkat sistem, efisiensi memori, dan portabilitas.

D. Mekanisme EtM

1. IP *Encapsulating Security Payload* (ESP) [36]

Protokol ESP dirancang untuk memberikan keamanan pada jaringan IPv4 dan IPv6. ESP berfungsi memberikan *confidentiality*, *data origin authentication*, *connectionless integrity*, *anti-replay service* (sebagai bentuk proteksi terhadap *partial sequence integrity*), serta *confidentiality limited traffic flow*.

Meskipun pada ESP layanan *confidentiality* dan *integrity* dapat diterapkan secara terpisah, pada umumnya ESP digunakan untuk menerapkan keduanya sekaligus, yaitu paket akan diproteksi baik dari sisi *confidentiality* maupun *integrity*. Saat kedua layanan tersebut diterapkan secara terpisah, maka proses pada sisi *originator* adalah data terlebih dahulu dienkapsulasi ke dalam *field ESP Payload*, kemudian ditambahkan *padding* jika diperlukan, dan hasilnya dienkripsi menggunakan kunci. Apabila layanan *integrity* diaktifkan, proses enkripsi dilakukan terlebih dahulu sebelum algoritma *integrity* diterapkan. Perlu dicatat bahwa proses enkripsi tidak mencakup *field Integrity Check Value* (ICV).

Urutan pemrosesan ini memungkinkan *recipient* mendeteksi dan menolak paket yang diulang (*replayed*) atau palsu dengan cepat, sebelum proses dekripsi dilakukan. Hal ini membantu mengurangi potensi dampak serangan *Denial of Service* (DoS). Selain itu, urutan tersebut juga memungkinkan pemrosesan paralel pada sisi *recipient*, yaitu dekripsi dapat dilakukan bersamaan dengan pemeriksaan *integrity*.

Karena *field ICV* tidak dienkripsi, maka algoritma *integrity* yang menggunakan kunci (*keyed integrity algorithm*) wajib digunakan untuk menghitung ICV. Setelah proses enkripsi selesai, *originator* menghitung nilai ICV atas seluruh paket ESP kecuali *field ICV* itu sendiri. Dengan demikian, perhitungan ICV mencakup *SPI*, *Sequence Number*, *Payload Data*, *Padding* (jika ada), *Pad Length*, dan *Next*

Header. Empat *field* terakhir dalam bentuk *ciphertext*, karena proses enkripsi dilakukan terlebih dahulu.

Pada sisi *recipient*, langkah pertama dalam verifikasi integritas nilai ICV atas seluruh paket ESP (kecuali *field* ICV) menggunakan algoritma integritas yang ditentukan. Hasil perhitungan tersebut kemudian dibandingkan dengan nilai ICV yang terdapat dalam paket. Jika nilai keduanya sama, maka datagram dianggap valid dan diterima. Namun jika hasilnya berbeda, maka datagram harus dibatalkan (*discarded*) karena dianggap tidak valid.

Setelah verifikasi berhasil, *recipient* kemudian melakukan dekripsi terhadap ESP *Payload Data*, *Padding*, *Pad Length* dan *Next Header*, memproses *padding* sesuai dengan spesifikasi algoritma enkripsi, memeriksa *field Next Header*, dan merekonstruksi kembali datagram IP asli.

Apabila digunakan algoritma gabungan (*combined mode algorithm*), yaitu algoritma yang menyediakan layanan konfidensialitas dan integritas secara bersamaan, maka pada sisi *originator*, data dienkapsulasi ke dalam *field* ESP *Payload Data*, ditambahkan *padding* jika diperlukan kemudian dilakukan enkripsi dan proteksi integritas sekaligus terhadap hasil enkapsulasi dan *padding* menggunakan kunci serta *combined mode algorithm*.

Sementara pada sisi *recipient*, proses dimulai dengan melakukan dekripsi dan memeriksa integritas terhadap ESP *Payload Data*, *Padding*, *Pad Length*, dan *Next Header* dengan menggunakan kunci dan *combined mode algorithm* yang sesuai. Jika pemeriksaan integritas gagal, datagram IP yang diterima harus dibuang karena dianggap tidak valid. Setelah itu, penerima memproses *padding*, memeriksa *field Next Header*, dan mengekstrak datagram IP asli. [36].

2. Protokol *Transport Layer Security* (TLS) dan Datagram TLS (DTLS)

Dalam RFC 7366 [37], mekanisme EtM diperkenalkan sebagai pengganti model lama MAC-then-encrypt pada protokol TLS dan DTLS.

EtM digunakan untuk meningkatkan keamanan karena mekanisme MAC-*then-encrypt* telah lama menjadi subjek berbagai kerentanan dan serangan keamanan. Implementasinya dilakukan dengan cara mengubah urutan operasi pada proses proteksi data. Pada skema lama, struktur datanya adalah $encrypt(data || MAC || pad)$, sedangkan pada EtM, data diproses menjadi $encrypt(data || pad) || MAC$. Dengan perubahan ini, MAC dihitung atas *ciphertext*, bukan *plaintext*, sehingga integritas data dapat diverifikasi terlebih dahulu sebelum proses dekripsi dilakukan. Negosiasi penggunaan EtM dilakukan selama *handshake* TLS/DTLS.

Selain itu, RFC 7366 menekankan bahwa mekanisme EtM hanya berlaku untuk *block cipher* yang memiliki operasi enkripsi dan MAC terpisah, dan tidak berlaku untuk *stream cipher* maupun AEAD *cipher* yang sudah memiliki autentikasi internal. Jika verifikasi MAC gagal, pemrosesan harus segera dihentikan, menghasilkan *error bad_record_mac* pada TLS atau pembuangan *record* pada DTLS, sehingga serangan berbasis *timing* dapat dihindari.

3. Protokol Kerberos

Kerberos v5 merupakan protokol autentikasi jaringan yang digunakan untuk memverifikasi identitas entitas dalam jaringan yang tidak aman. Tujuan protokol Kerberos adalah memverifikasi bahwa entitas yang mencoba mengakses layanan/sumber daya adalah pihak yang sah, bukan penyerang yang menyamar.

Protokol ini dirancang untuk memberikan autentikasi yang kuat bagi aplikasi klien/server dengan menggunakan kriptografi kunci simetris [38] dan menggunakan pihak ketiga tepercaya. Kerberos dibuat oleh MIT sebagai solusi terhadap masalah keamanan jaringan. Protokol Kerberos menggunakan kriptografi yang kuat sehingga klien dapat membuktikan identitasnya kepada server (dan sebaliknya) melalui koneksi jaringan yang tidak aman. Setelah klien dan server menggunakan Kerberos untuk saling membuktikan identitas, mereka

juga dapat mengenkripsi semua komunikasi mereka untuk menjamin privasi dan integritas data selama interaksi berlangsung.

Dalam sistem keamanan Kerberos v5, data yang dikirimkan antara klien, server, dan KDC harus dilindungi dari penyadapan dan manipulasi. Untuk memenuhi kebutuhan ini, Kerberos menggunakan mekanisme enkripsi yang menjamin konfidensialitas dan integritas data. RFC 8009 [39] memperkenalkan pendekatan yang lebih modern dengan menerapkan prinsip AE.

Sebelum adanya RFC 8009, Kerberos menggunakan algoritma seperti AES dengan HMAC-SHA1 untuk melindungi data seperti yang dijelaskan dalam RFC 3962 [40]. Namun, perkembangan teknologi kriptografi dan peningkatan kekuatan serangan membuat standar tersebut dianggap kurang aman. RFC 8009 hadir untuk memperkuat keamanan dengan menggunakan AES untuk enkripsi dan HMAC-SHA2 untuk autentikasi, yang menawarkan tingkat perlindungan yang lebih tinggi terhadap manipulasi dan penyadapan. Algoritma yang dapat digunakan adalah “aes128-cts-hmac-sha256-128” dan “aes256-cts-hmac-sha384-192” [39].

AE dalam RFC 8009 tidak menggunakan mode AEAD seperti pada TLS 1.3, namun prinsip dasarnya tetap sama. Data yang dienkripsi tidak hanya dijaga konfidensialitasnya, tetapi juga dilindungi dari modifikasi melalui proses autentikasi yang melekat. Berikut urutan proses AE pada Kerberos:

- Enkripsi dilakukan dengan algoritma AES dalam mode *Cipher Text Stealing* (CTS), yang memungkinkan enkripsi blok data tanpa perlu *padding*.
- Setelah proses enkripsi, HMAC-SHA2 digunakan untuk membuat *checksum* dari *ciphertext*. *Checksum* ini menjadi semacam tanda tangan kriptografi yang akan diverifikasi saat data diterima.

- *Checksum* dan *ciphertext* dikirim bersama, sehingga penerima dapat memverifikasi keaslian dan integritas data sebelum melakukan dekripsi.

Kerberos v5 secara fungsional menerapkan skema *Encrypt-then-MAC* (EtM). Meskipun istilah "*Encrypt-then-MAC*" tidak disebut secara eksplisit dalam dokumen, urutan proses *authenticated encryption* pada Kerberos memastikan bahwa integritas data diverifikasi sebelum dilakukan dekripsi, sesuai dengan prinsip EtM. Pendekatan ini memberikan perlindungan terhadap manipulasi data dan serangan seperti *padding oracle*, serta mempertahankan panjang pesan dengan menggunakan teknik *ciphertext stealing* [39].

Untuk mendukung mekanisme ini, Kerberos v5 menggunakan struktur kunci khusus. Dari satu kunci utama (*base key*), sistem akan menurunkan dua kunci berbeda, yaitu untuk proses enkripsi AES dan untuk HMAC-SHA2 sebagai alat autentikasi. Dengan pendekatan ini, proses autentikasi dan enkripsi saling terikat, sehingga jika data dimodifikasi oleh pihak ketiga, verifikasi *checksum* akan gagal dan data tidak akan diterima.

E. Mekanisme GCM

1. AES-GCM pada IP Sec *Encapsulating Security Payload* (ESP)

Dalam RFC 4106 [41], AES-GCM diterapkan pada mekanisme IPsec *Encapsulating Security Payload* (ESP) untuk memberikan layanan kerahasiaan dan autentikasi asal data. Secara spesifik, penggunaan AES-GCM terletak pada bagian *payload* ESP, yaitu bagian yang berisi *plaintext* yang diubah menjadi *ciphertext* serta dilengkapi dengan *authentication tag* atau *Integrity Check Value* (ICV). Mode operasi GCM berperan untuk mengamankan isi paket ESP melalui enkripsi dalam *counter mode* dan memastikan keaslian data melalui operasi *Galois field*. Penerapan AES-GCM dalam IPsec ESP memberikan manfaat berupa efisiensi yang tinggi karena proses enkripsi dan autentikasi dilakukan secara paralel, serta membuat kinerja menjadi optimal baik pada implementasi perangkat keras

maupun perangkat lunak sehingga mendukung kecepatan sekitar 10 hingga 40 gigabit per detik sesuai kebutuhan jaringan berkecepatan tinggi. Dalam penerapan di dunia nyata, IPsec dengan ESP umumnya digunakan sebagai komponen utama pada *Virtual Private Network* (VPN), yaitu sistem yang memungkinkan dua jaringan atau perangkat berkomunikasi secara aman melalui internet dengan cara mengenkripsi dan memverifikasi setiap paket data.

2. AES-GCM pada SSH

Dalam RFC 5647, AES-GCM digunakan pada *Secure Shell* (SSH) *Transport Layer Protocol* sebagai algoritma AE yang menyediakan layanan kerahasiaan sekaligus integritas data. Penggunaan spesifik AES-GCM dijelaskan sebagai penerapan mode AEAD dalam SSH, dengan dua varian yaitu AEAD_AES_128_GCM dan AEAD_AES_256_GCM. Mode AEAD tersebut menggabungkan fungsi enkripsi dan autentikasi untuk melindungi *payload*, *padding length*, serta *random padding* pada paket biner SSH, sementara *packet length* diproses sebagai AAD dan menghasilkan *ciphertext* serta *authentication tag* sepanjang 16 oktet yang ditempatkan pada *MAC field* di akhir paket. Penggunaan AES-GCM memberikan manfaat berupa efisiensi tinggi, karena mampu menyediakan *confidentialitas* sekaligus integritas data melalui proses enkripsi berbasis *counter mode* yang dapat diproses secara paralel, serta mekanisme autentikasi yang dapat diimplementasikan dengan kecepatan tinggi pada perangkat keras. SSH sendiri merupakan protokol login jarak jauh yang aman yang menyediakan algoritma untuk proses autentikasi, *key agreement*, *confidentialitas*, dan layanan integritas data.

3. Camellia-GCM pada TLS 1.2

Dalam RFC 6367 [42], Camellia-GCM digunakan pada protokol TLS sebagai salah satu algoritma *cipher suites* baru yang ditambahkan untuk mendukung penggunaan algoritma *block cipher* Camellia. RFC 6367 menjelaskan bahwa Camellia sebelumnya telah digunakan sebagai *cipher suites* dengan mekanisme HMAC-SHA-256

yang dijelaskan dalam RFC 5932. Namun pada RFC 6367 diperluas dengan varian yang mendukung GCM sebagai mode AE pada TLS 1.2. Penjelasan spesifik mengenai Camellia-GCM terdapat pada bagian *GCM-based cipher suites* dan *Pre-Shared Key (PSK) cipher suites*, yang mendefinisikan dua puluh *cipher suites* berbasis GCM dan enam *cipher suites* versi PSK yang juga menggunakan Camellia berbasis GCM didalamnya. Pada bagian *Cipher Suite Definitions* dalam RFC 6367, dijelaskan bahwa Camellia-GCM dirancang mengikuti struktur AES-GCM sebagaimana didefinisikan dalam RFC 5116 [43] dan RFC 5288 [44], karena Camellia dan AES memiliki karakteristik yang sama dalam ukuran kunci (128/256 bit) dan panjang blok (128 bit). TLS sendiri merupakan protokol keamanan yang memungkinkan aplikasi klien dan server berkomunikasi melalui Internet dengan aman, dengan cara mencegah terjadinya penyadapan, perubahan isi data, maupun pemalsuan pesan selama proses pengiriman. TLS digunakan secara luas dalam berbagai layanan digital, seperti HTTPS untuk mengamankan koneksi antara browser dan situs web, serta pada layanan email agar isi pesan tetap terlindungi saat dikirim atau diterima melalui jaringan.

4. ARIA-GCM pada TLS 1.2 di Korea

Dalam RFC 6209 [45], ARIA-GCM digunakan pada protokol TLS sebagai bagian dari perluasan *cipher suites* yang mendukung algoritma *block cipher* ARIA. ARIA sendiri merupakan algoritma *block cipher* yang dikembangkan oleh peneliti di Korea pada tahun 2003 yang kemudian ditetapkan sebagai standar nasional Korea pada tahun 2004 dan digunakan dalam layanan pemerintah Korea kepada publik. RFC 6209 menambahkan varian baru yang mengombinasikan algoritma ARIA dengan GCM sebagai mekanisme AE pada TLS 1.2. Penjelasan spesifik mengenai ARIA-GCM terdapat pada bagian *GCM-based cipher suites* dan *Pre-Shared Key (PSK) cipher suites*, yang mendefinisikan dua puluh *cipher suites* berbasis GCM dan enam *cipher suites* versi PSK yang juga menggunakan ARIA berbasis GCM

didalamnya. Pada bagian *Cipher Suite Definitions* dalam RFC 6209, dijelaskan bahwa ARIA-GCM dirancang dengan karakteristik yang sama dalam ukuran kunci (128/256 bit) dan panjang blok (128 bit) seperti AES-GCM, serta mengikuti konsep AES-AEAD yang dijelaskan dalam RFC 5116 [43] dan konsep AES-GCM untuk TLS yang dijelaskan dalam RFC 5288 [44].

5. SM4-GCM pada TLS 1.3 di Tiongkok

Berdasarkan RFC 8998 [46], SM4-GCM digunakan pada protokol TLS versi 1.3 sebagai bagian dari *cipher suites* baru bernama TLS_SM4_GCM_SM3 yang mengombinasikan SM4-GCM dengan fungsi *hash* SM3. *Cipher suites* TLS_SM4_GCM_SM3 memanfaatkan SM4 sebagai *block cipher* yang beroperasi dalam GCM untuk menyediakan layanan enkripsi sekaligus autentikasi yang diperlukan oleh TLS 1.3. Pada bagian pendefinisian algoritma dijelaskan bahwa AEAD_SM4_GCM bekerja dengan empat input, yaitu *key*, *initialization vector* (IV) atau *nonce*, *plaintext*, dan *associated data*, kemudian menghasilkan dua output berupa *ciphertext* dan *authentication tag* (kode autentikasi pesan). *Authentication tag* dikonstruksi berdasarkan informasi dari TLS *record header* sesuai dengan ketentuan TLS 1.3 dan nilai *nonce* harus unik untuk setiap operasi enkripsi. Algoritma ShangMi, termasuk SM4, merupakan standar kriptografi nasional Tiongkok yang ditetapkan oleh *Standardization Administration of the People's Republic of China* dan diwajibkan penggunaannya dalam berbagai regulasi keamanan informasi di negara tersebut.

6. AES-GCM dalam IEEE 802.1AE MAC Security

Dalam IEEE Std 802.1AE-2018 *Media Access Control (MAC) Security* [47], AES-GCM digunakan sebagai *cipher suites*. Terdapat empat *cipher suites* yang disebutkan dalam IEEE Std 802.1AE-2018 *MAC Security* meliputi GCM-AES-128, GCM-AES-256, GCM-AES-XPN-128, dan GCM-AES-XPN-256. *Cipher suites* GCM-AES-128 merupakan *cipher suites default* dan bersifat wajib. Keempat *cipher*

suites tersebut digunakan untuk menyediakan layanan *integrity protection* dan *confidentiality protection*. Pada mode *integrity protection*, keempat cipher suite tersebut menggunakan kombinasi alamat MAC tujuan dan sumber, SecTAG, serta User Data sebagai *associated data* (*A*), sementara *plaintext* (*P*) bernilai *null* dan *Secure Data* adalah User Data yang tidak diubah. Pada mode *confidentiality protection* tanpa *confidentiality offset*, nilai *A* terdiri dari alamat MAC tujuan dan sumber serta SecTAG, *P* adalah User Data, dan *Secure Data* adalah ciphertext (*C*) hasil enkripsi. Berdasarkan penjelasan dalam standar IEEE Std 802.1AE-2018 MAC Security, khusus GCM-AES-128 dan GCM-AES-256 juga mendukung *confidentiality protection* dengan *confidentiality offset*, di mana *A* mencakup sebagian awal User Data sesuai nilai *offset*, *P* adalah sisa User Data, dan *Secure Data* merupakan gabungan bagian *offset plaintext* dengan *ciphertext*. Sementara itu, GCM-AES-128 dan GCM-AES-256 tidak menyediakan mekanisme *confidentiality offset*.

Media Access Control Security (MACsec) sebagaimana didefinisikan dalam standar IEEE 802.1AE merupakan mekanisme keamanan jaringan yang memungkinkan sistem-sistem terotorisasi yang terhubung ke LAN untuk menjaga kerahasiaan data yang ditransmisikan serta melindungi jaringan dari frame yang dikirim atau dimodifikasi oleh perangkat tidak sah. Penerapan MACsec mendukung berbagai tujuan penting, seperti menjaga konektivitas dan layanan jaringan yang benar, mengisolasi serangan *denial of service*, melokalisasi sumber komunikasi jaringan hingga LAN asalnya, serta memungkinkan pembangunan jaringan publik berbasis infrastruktur LAN yang digunakan bersama oleh berbagai pihak, termasuk organisasi yang saling tidak mempercayai. Selain itu, MACsec memungkinkan komunikasi yang aman antar organisasi melalui LAN dan dapat diterapkan secara bertahap tanpa mengganggu operasi jaringan yang sedang berjalan.

7. AES-GCM pada *Secure Real-time Transport Protocol* (SRTP)

Dalam RFC 7714 [48], AES-GCM diterapkan pada *Secure Real-time Transport Protocol* (SRTP) untuk memberikan layanan kerahasiaan, autentikasi pesan, dan perlindungan terhadap *replay attack* pada trafik RTP dan RTCP. RFC 7714 mendefinisikan dua konfigurasi, yaitu AEAD_AES_128_GCM dan AEAD_AES_256_GCM, yang masing-masing menggunakan kunci 128-bit dan 256-bit dengan *authentication tag* sepanjang 16 oktet. GCM memanfaatkan *Counter Mode* untuk proses enkripsi yang dapat di-*pipelining* secara efisien, sementara fungsi autentikasinya berbasis operasi *Galois field* yang cocok untuk implementasi perangkat keras berkecepatan tinggi maupun *compact circuit*. Penerapan AES-GCM pada SRTP dan SRTCP menggunakan konsep AEAD sebagaimana didefinisikan dalam RFC 5116. SRTP digunakan untuk mengamankan komunikasi suara dan video melalui internet misalnya saat berkomunikasi melalui telepon lewat internet (VoIP), *video call*, atau *video conference*.

F. Mekanisme AEAD

1. AEAD pada Transport Layer Security 1.3

Tujuan utama *Transport Layer Security* (TLS) adalah menyediakan saluran aman antara dua pihak yang berkomunikasi. Satu-satunya persyaratan dari lapisan *transport* yang mendasarinya adalah aliran data yang andal dan berurutan. Secara khusus, saluran aman harus menyediakan properti berikut [49]:

- a. Autentikasi: Sisi server saluran selalu diautentikasi; sisi klien bersifat opsional. Autentikasi dapat terjadi melalui kriptografi asimetris (misalnya, RSA [RSA], *Elliptic Curve Digital Signature Algorithm* (ECDSA) [ECDSA], atau *Edwards-Curve Digital Signature Algorithm* (EdDSA) [RFC8032]) atau kunci pra-berbagi simetris.
- b. Konfidensialitas: Data yang dikirim melalui saluran setelah pembuatan hanya dapat dilihat oleh titik akhir. TLS tidak menyembunyikan panjang data yang dikirimkannya, meskipun

titik akhir dapat menutupi catatan TLS untuk mengaburkan panjang dan meningkatkan perlindungan terhadap teknik analisis lalu lintas.

- c. Integritas: Data yang dikirim melalui saluran setelah pembentukan tidak dapat dimodifikasi oleh penyerang tanpa terdeteksi.

TLS terdiri dari dua komponen utama [49]:

- a. Protokol *handshake* mengautentikasi pihak-pihak yang berkomunikasi, menegosiasikan mode dan parameter kriptografi, dan menetapkan material kunci bersama (*shared keying material*). Protokol *handshake* dirancang untuk menahan gangguan (*resist tampering*), sehingga penyerang aktif tidak dapat memaksa kedua pihak untuk menegosiasikan parameter yang seharusnya digunakan dalam kondisi koneksi yang aman.
- b. Protokol *record* menggunakan parameter yang ditetapkan oleh protokol *handshake* untuk melindungi lalu lintas antara pihak yang berkomunikasi. Protokol *record* membagi lalu lintas menjadi serangkaian *record*, yang masing-masing dilindungi secara independen menggunakan kunci lalu lintas (*traffic keys*).

TLS 1.3 sebagaimana distandarisasi pada RFC 8446, adalah versi terkini protokol TLS, yang menawarkan peningkatan keamanan dan kinerja, dan diimplementasikan oleh berbagai *library* dan kerangka kerja, termasuk wolfSSL, MatrixSSL, OpenSSL, IBM, web browser seperti Chrome, Safari, dan Firefox.

TLS 1.3 menggunakan AEAD pada *record layer* [49], yaitu lapisan yang bertugas mengenkripsi dan mendekripsi data aplikasi serta data *handshake* setelah *handshake* negosiasi awal selesai, data dienkripsi dan diautentikasi dalam *record layer* untuk memastikan konfidensialitas, integritas, dan autentikasi data.

- a. *Record Protection (Post-Handshake)*
- Setelah TLS *handshake* selesai dan *keys* telah dinegosiasikan, semua data (termasuk data aplikasi seperti HTTP, dan pesan *handshake* lanjutan) dienkripsi menggunakan algoritma AEAD.
 - Protokol ini mengenkripsi setiap TLS *record* menggunakan AEAD dengan input:
 - *Plaintext*: data yang akan dienkripsi (misalnya HTTP *request*).
 - *Additional Data*: *header record* (seperti jenis *record* dan versinya) yang tidak dienkripsi tapi di-autentikasi.
 - *Key* dan *Nonce*: berasal dari *key schedule* TLS 1.3.
- b. TLS 1.3 hanya mendukung *cipher suite* yang menggunakan AEAD sebagai berikut:
- TLS_AES_128_GCM_SHA256
 - TLS_AES_256_GCM_SHA384
 - TLS_CHACHA20_POLY1305_SHA256
 - TLS_AES_128_CCM_SHA256
 - TLS_AES_128_CCM_8_SHA256
- Cipher suite* ini menentukan algoritma AEAD yang digunakan, yaitu GCM, CCM, ChaCha20-Poly1305.
- c. *Key Schedule* dan AEAD
- TLS 1.3 menggunakan HMAC-based *Key Derivation Function* (HKDF) untuk menghasilkan AEAD *key* dan *nonce* dari *shared secret* yang diperoleh melalui *handshake* (biasanya dengan *Diffie-Hellman*).
 - AEAD digunakan setelah *client* dan server selesai bertukar *Finished messages* dan menyepakati *session keys*.

2. AEAD pada *Internet Key Exchange* v.2

Internet Key Exchange (IKE) adalah protokol kriptografi yang dirancang untuk mengotomatiskan proses negosiasi dan pembentukan *Security Association* (SA) dalam arsitektur IPsec (*Internet Protocol Security* (IPsec)). Versi kedua dari protokol tersebut yang dikembangkan sebagai pembaruan dan penyempurnaan terhadap IKE versi pertama dikenal sebagai IKEv2. IKEv2 adalah protokol yang bertugas menyusun parameter keamanan, meliputi algoritma kriptografi serta kunci enkripsi yang digunakan untuk komunikasi yang aman. Protokol IKEv2 didefinisikan secara formal dalam RFC 7296 dan telah distandardisasi oleh IETF [50].

IKEv2 dirancang untuk mengamankan proses negosiasi parameter keamanan antara dua pihak dalam komunikasi jaringan. Adapun tujuan dari IKEv2 meliputi:

- Melakukan autentikasi dua arah (*mutual authentication*) antara entitas yang saling berkomunikasi.
- Membangun dan memelihara SA baik untuk protokol IKE itu sendiri maupun untuk protokol IPsec seperti *Encapsulating Security Payload* (ESP) dan *Authentication Header* (AH).
- Menyediakan mekanisme manajemen kunci yang efisien dan aman, termasuk penggunaan algoritma kriptografi seperti Diffie-Hellman untuk menghasilkan *shared key*.
- Mendukung *rekeying* SA tanpa mengganggu koneksi yang sedang berlangsung.
- Menangani kondisi jaringan dinamis, termasuk mobilitas dan translasi alamat jaringan (NAT traversal).

IKEv2 mengimplementasikan autentikasi entitas sebagai bagian yang tidak terpisahkan dari proses pertukaran pesan, khususnya pada tahapan *IKE_AUTH exchange*. Autentikasi dilakukan setelah pembentukan parameter awal kriptografi

melalui *IKE_SA_INIT exchange*, saat kunci bersama telah dibangkitkan menggunakan metode Diffie-Hellman dan nilai *nonce*. Selama *IKE_AUTH exchange*, masing-masing pihak mengirimkan identitasnya (*IDi* dan *IDr*), serta melakukan autentikasi menggunakan metode yang disepakati, yang dapat berupa:

- *Pre-shared key* (PSK)
- Sertifikat digital (misalnya, X.509), atau
- *Extensible Authentication Protocol* (EAP).

Autentikasi tersebut tidak hanya menjamin bahwa kedua entitas adalah pihak yang sah (*legitimate*), tetapi juga memastikan integritas pesan dan kerahasiaan identitas melalui enkripsi berbasis kunci yang telah disepakati sebelumnya.

IKEv2 mendukung dan merekomendasikan penggunaan algoritma AEAD. Berdasarkan RFC 5282, algoritma AEAD yang didukung oleh IKEv2 adalah sebagai berikut [51]:

- AEAD_AES_128_GCM_8
- AEAD_AES_256_GCM_8
- AEAD_AES_128_GCM_12
- AEAD_AES_256_GCM_12
- AEAD_AES_128_CCM_SHORT
- AEAD_AES_256_CCM_SHORT
- AEAD_AES_128_CCM_SHORT_8
- AEAD_AES_256_CCM_SHORT_8
- AEAD_AES_128_CCM_SHORT_12
- AEAD_AES_256_CCM_SHORT_12

Selain itu, terdapat juga algoritma ChaCha20-Poly1305 yang digunakan dalam IKEv2 sebagaimana dijelaskan dalam RFC 7634 [52].

BAB V

PENUTUP



BAB V

PENUTUP

Dokumen ini telah menyajikan uraian menyeluruh mengenai konsep, prinsip, serta implementasi AE sebagai salah satu mekanisme penting dalam keamanan informasi. Dimulai dengan pengenalan dasar mengenai konsep dan mekanisme AE serta pemanfaatannya, dokumen ini menekankan bahwa AE tidak hanya menjamin konfidensialitas, tetapi juga melindungi integritas dan keautentikan data. Hal tersebut menjadikan AE relevan untuk berbagai aplikasi modern, mulai dari komunikasi data sehari-hari hingga perlindungan infrastruktur informasi vital.

Dokumen ini juga telah menguraikan berbagai mekanisme AE, serta contoh-contoh implementasi populer dalam *library* kriptografi dan protokol keamanan. Setiap mekanisme dijelaskan dari sisi prinsip kerja, kelebihan, keterbatasan, serta konteks penggunaannya. Dengan demikian, pembaca diharapkan memperoleh pemahaman yang komprehensif tentang mekanisme AE yang tersedia, serta mampu menyesuaikan penerapannya sesuai kebutuhan dan risiko operasional yang dihadapi.

Dokumen ini diharapkan dapat menjadi referensi praktis bagi berbagai pihak mulai dari pengembang sistem, auditor, regulator, hingga penyelenggara infrastruktur untuk memahami, memilih, dan menerapkan AE secara tepat. Keberhasilan implementasi AE akan sangat bergantung pada kolaborasi antar pemangku kepentingan, penguatan kapasitas teknis, serta pembaruan dokumen secara berkala seiring perkembangan ancaman dan standar keamanan global.

REFERENSI

- [1] M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm," *Journal of Cryptology*, pp. 469-491, 2008.
- [2] M. Bellare, P. Rogaway and D. Wagner, "The EAX Mode of Operation (A Two-Pass Authenticated-Encryption Scheme Optimized for Simplicity and Efficiency)," *Fast Software Encryption, 11th International Workshop, FSE 2004*, no. Springer-Verlag, pp. 389-407, 2004.
- [3] "Standar Nasional Indonesia, SNI ISO/IEC 19772:2020 Keamanan informasi — Enkripsi terautentikasi (ISO/IEC 19772:2020, IDT, Eng)," Badan Standardisasi Nasional, 2025.
- [4] "Standar Nasional Indonesia, SNI ISO/IEC 18033-3:2010 Teknologi informasi – Teknik keamanan – Algoritma enkripsi – Bagian 3: Penyandian blok (ISO/IEC 18033-3:2010 dan ISO/IEC 18033-3:2010 Amd.1:2021, IDT, Eng)," Badan Standardisasi Nasional, Jakarta, 2022.
- [5] "Keputusan Kepala BSSN Nomor 433 Tahun 2025 tentang Algoritma Kriptografi Indonesia," Badan Siber dan Sandi Negara, 2025.
- [6] M. Dworkin, "NIST Special Publication 800-38C Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality," National Institute of Standards and Technology, 2007.
- [7] "Standar Nasional Indonesia, SNI ISO IEC 10116:2017 Teknologi informasi – Teknik keamanan – Mode operasi untuk penyandian blok n-bit (ISO/IEC 10116:2017 dan ISO/IEC 10116:2017/Amd 1:2021, IDT, Eng)," Badan Standardisasi Nasional, 2025.
- [8] Standar Nasional Indonesia, SNI ISO IEC 9797-1:2011 Teknologi informasi — Teknik keamanan — Message Authentication Codes (MACs) — Bagian 1: Mekanisme menggunakan penyandian blok (ISO/IEC 9797-1:2011 dan ISO/IEC 9797-1:2011/Amd 1:2023, IDT, Eng), Badan Standardisasi Nasional, 2025.
- [9] "NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," National Institute of Standards and Technology, November 2007.

- [10] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping (NIST SP 800-38F)," Institute of Standards and Technology, 2012.
- [11] NIST, "NIST Special Publication 800-131A: Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths," NIST, January, 2011.
- [12] D. Whiting, R. Housley and N. Ferguson, "RFC 3610: Counter with CBC-MAC (CCM)," IETF, 2003.
- [13] P. Rogaway, M. Bellare, J. Black and T. Krovetz, "OCB: A block-Cipher Mod of Operation for Efficient Authenticated Encryption," in *8th ACM Conference on Computer and Communications Security*, pp 196-295, ACM Press, 2001.
- [14] T. Iwata and . K. Kurosawa, "OMAC: One-key CBC MAC," in *Proceedings Fast Software Encryption*, 2003.
- [15] NIST, "NIST Special Publication 800-38B: 2005, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication," NIST, May 2005.
- [16] ANSI, "ANSI X9.24-1:2004, Retail Financial Services Symmetric Key Management — Part 1: Using Symmetric Techniques," 2005.
- [17] ISO, "ISO/IEC 11770-6:2016, Information technology — Security techniques — Key management Part 6: Key derivation," ISO, 2016.
- [18] D. J. Bernstein, "ChaCha, a variant of Salsa20," 2008.
- [19] Y. Nir and A. Langley, "Request for Comments: 8439 ChaCha20 and Poly1305 for IETF Protocols," Internet Research Task Force (IRTF), June 2018.
- [20] A. Moise, . E. Berozet, T. Phinney, M. Burns and A. C. S. C. , "EAX' Cipher Mode," ANSI C12 SC17 Committee.
- [21] A. W. Services, "Manage AES key wrapping in AWS CloudHSM," [Online]. Available:
<https://docs.aws.amazon.com/cloudhsm/latest/userguide/manage-aes-key-wrapping.html>.
- [22] Microsoft, "HSM-protected keys and Bring Your Own Key (BYOK) in Azure Key Vault, Azure Key Vault documentation," [Online]. Available:
<https://learn.microsoft.com/en-us/azure/key-vault/keys/hsm-protected-keys-byok>.

- [23] G. Cloud, "Key wrapping," Cloud Key Management Service documentation, [Online]. Available: <https://cloud.google.com/kms/docs/key-wrapping>.
- [24] B. Moran, H. Tschofenig, H. Birkholz and R. Hurst, "Firmware Encryption for IoT Devices (draft-ietf-suit-firmware-encryption-24)," Internet Engineering Task Force (IETF), 2025.
- [25] IBM, "IBMJCECCA z/OS Hardware Cryptography Reference Guide," [Online]. Available: <https://public.dhe.ibm.com/software/Java/Java80/IBMJCECCA/zOS/HWCryptoRefGuide.html>.
- [26] M. Jones, J. Bradley and N. Sakimura, "RFC 7519: JSON Web Token (JWT)," Internet Engineering Task Force (IETF), Mei 2015.
- [27] M. Jones and J. Hildebrand, "RFC 7516: JSON Web Encryption (JWE)," Internet Engineering Task Force, Mei 2015.
- [28] M. Jones, "RFC 7518: JSON Web Algorithms (JWA)," Internet Engineering Task Force, Mei 2015.
- [29] Mastercard, "Securing Sensitive Data," [Online]. Available: <https://developer.mastercard.com/loyalty-user-management/documentation/securing-sensitive-data/>. [Accessed 19 June 2025].
- [30] "fandom.com," [Online]. Available: https://cryptography.fandom.com/wiki/EAX_mode. [Accessed 22 Agustus 2025].
- [31] "www.design-reuse.com," [Online]. Available: <https://www.design-reuse.com/news/202518183-ip-cores-inc-ships-an-aes-encryption-core-supporting-the-eax-encryption-mode-of-ansi-c12-22/>. [Accessed 22 Agustus 2025].
- [32] "wolfCrypt now supports AES EAX," wolfSSL, 11 October 2023. [Online]. Available: <https://www.wolfssl.com/wolfcrypt-now-supports-aes-eax/>. [Accessed 19 September 2025].
- [33] C. Library, "Authenticated Encryption Modes: EAX Mode Documentation," 14 April 2021. [Online]. Available: https://www.cryptopp.com/wiki/EAX_Mode#AE. [Accessed 1 October 2025].

- [34] "Bouncy Castle Cryptography Library 1.81," [Online]. Available: <https://downloads.bouncycastle.org/java/docs/bcprov-jdk14-javadoc/org/bouncycastle/crypto/modes/EAXBlockCipher.html>. [Accessed 19 September 2025].
- [35] RustCrypto, "EAX: Authenticated Encryption with AES-CTR + CMAC (RustCrypto AEAD Implementation)," [Online]. Available: <https://docs.rs/eax/latest/eax/>. [Accessed 19 September 2025].
- [36] S. Kent, "Request for Comment: 4303 IP Encapsulating Security Payload (ESP)," BBN Technologies: Network Working Group, December 2005.
- [37] P. Gutmann, "Request for Comments: 7366, Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)," University of Auckland: Internet Engineering Task Force (IETF), September 2014.
- [38] "Kerberos: The Network Authentication Protocol," 26 June 2024. [Online]. Available: <https://web.mit.edu/kerberos/>. [Accessed 2 May 2025].
- [39] M. Jenkins, M. Peck and K. Burgin, "Request for Comments: 8009 - AES Encryption with HMAC-SHA2 for Kerberos 5," Internet Engineering Task Force (IETF), October 2016.
- [40] K. Raeburn, "Request for Comments: 3962 - Advanced Encryption Standard (AES) Encryption for Kerberos 5," Internet Engineering Task Force, February 2005.
- [41] J. Viega and D. McGrew, "Request for Comments: 4106, The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)," Internet Engineering Task Force, June 2005.
- [42] S. Kanno and M. Kanda, "Request for Comments: 6367, Addition of the Camellia Cipher Suites to Transport Layer Security (TLS)," Internet Engineering Task Force, September 2011.
- [43] D. McGrew, "Request for Comments: 5116, An Interface and Algorithms for Authenticated Encryption," Internet Engineering Task Force, January 2008.
- [44] J. Salowey, A. Choudhury and D. McGrew, "Request for Comments: 5288, AES Galois Counter Mode (GCM) Cipher Suites for TLS," Internet Engineering Task Force, August 2008.
- [45] W. Kim, J. Lee, J. Park and D. Kwon, "Request for Comments: 6209, Addition of the ARIA Cipher Suites to Transport Layer Security (TLS)," Internet Engineering Task Force, April 2011.

- [46] P. Yang and A. Group, "RFC: 8998, ShangMi (SM) Cipher Suites for TLS 1.3," Internet Engineering Task Force, March 2021.
- [47] "IEEE Standard for Local and metropolitan area networks-Media Access Control (MAC) Security," IEEE Standards Association, 2018.
- [48] D. McGrew and K. Igoe, "Request for Comments: 7714, AES-GCM Authenticated Encryption in the Secure Real-time Transport Protocol (SRTP)," Internet Engineering Task Force (IETF), December 2015.
- [49] E. Rescorla, "Request for Comments: 8446 -The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force (IETF), 2018.
- [50] C. Kaufman, P. E. Hoffman, P. Eronen, T. Kivinen and Y. Nir, "Request for Comments: 7296 - Internet Key Exchange Protocol Version 2 (IKEv2)," Internet Engineering Task Force (IETF), October 2014.
- [51] D. L. Black and D. McGrew, "Request for Comments: 5282 - Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol," Internet Engineering Task Force, August 2008.
- [52] Y. Nir, "Request for Comments: 7634 - ChaCha20, Poly1305, and Their Use in the Internet Key Exchange Protocol (IKE) and IPsec," Internet Engineering Task Force (IETF), August 2015.





**BADAN SIBER
DAN SANDI
NEGARA**



**Deputi Bidang Strategi dan Kebijakan Keamanan Siber dan Sandi
Badan Siber dan Sandi Negara**

Jl. Cijayanti No. 70, Kec. Babakan Madang, Kabupaten Bogor

2025

PUBLIK