FIT5037: Network Security
Symmetric key cryptography

Faculty of Information Technology
Monash University

Copyright Notice

Lecture Topics:

- **Symmetric key cryptography**
- Asymmetric key cryptography
- Pseudorandom Number Generators and hash functions
- Authentication Methods and AAA protocols
- Security at Network layer
- Security at Network layer (continued)
- Security at Transport layer
- Security at Application layer
- Computer system security and malicious code
- Computer system vulnerabilities and penetration testing
- Intrusion detection
- Denial of Service Attacks and Countermeasures /

Outline

- Symmetric Encryption Model
- AES
- Modes of operation
- Stream Ciphers
- Authenticated Encryption
- AEAD Modes of operation

Cryptography

from the Greek word Crypto which means hidden

Cryptography[1] refers to the mathematical science that deals with *transforming* data:

- to render its meaning unintelligible (i.e., to hide its semantic content),
- prevent its undetected alteration,
- or prevent its unauthorised use.
- If the transformation is reversible, cryptography also deals with restoring encrypted data to intelligible form.

[1] RFC4949 Internet Security Glossary, Version 2

Cryptography is a required security service for information and network security and a means to achieve:

- authentication
- data integrity
- confidentiality and privacy
- digital signature

Cryptographic Techniques

Main cryptographic techniques:

- Encryption
  - Symmetric encryption
  - Asymmetric encryption
- Hash function
- Message Authentication
  - Symmetric: Message Authentication Code
  - Asymmetric: Digital Signature

Symmetric - Asymmetric encryption names

Symmetric:

- shared-secret/shared-key encryption
- single-key/one-key encryption
- secret-key/private-key encryption
- conventional encryption

Asymmetric:

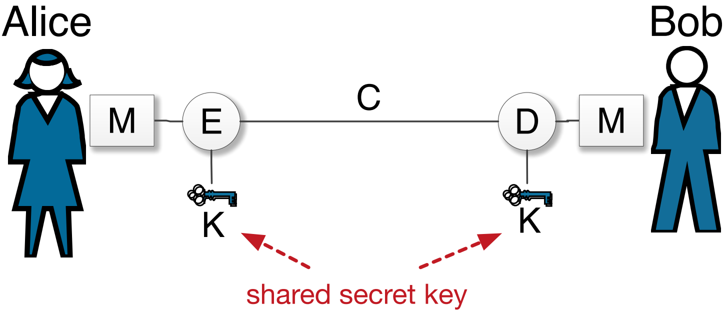
- public key encryption
- dual-key encryption

Encryption and Decryption

- If $\mathcal{K}$ denotes the *key space*, $\mathcal{M}$ the *message space*, and $\mathcal{C}$ the *ciphertext space* then:
  - Encryption $\mathcal{E}_k, k \in \mathcal{K}$ uniquely identifies a bijection (one-to-one and onto) from $\mathcal{M}$ to $\mathcal{C}$
  - or it is the transformation of data into a form that is almost impossible (computationally infeasible) to read without the appropriate knowledge
- $\mathcal{D}_k = \mathcal{E}^{-1}{}_k, k \in \mathcal{K}$ and is a bijection from $\mathcal{C}$ to $\mathcal{M}$
  - Decryption is the reverse of encryption - the transformation of encrypted data back into an intelligible form
- A key or key pair (appropriate knowledge) is necessary for encryption and decryption
  - A key is a digital object

MONASH University

Symmetric Encryption Model



- Alice: $C = E(K, M)$ (sometimes expressed as $C = E_K(M)$)
- Alice $\rightarrow$ Bob: C
- Bob: $M = D(K, C)$

Security depends on the *secrecy of the key* not secrecy of the algorithm

Unconditional Security [a]

[a] Handbook of applied cryptography, Chapter 1

- Adversary has unlimited power
- Observation of ciphertext provides no additional information to the adversary about plaintext

Example:

- One-time pad:
  - random key as long as the message
  - key is used only once

Difficulties of one-time pad:

- random key as long as the message
- sharing the key with recipient

Computational Security [a]

[a] Handbook of applied cryptography, Chapter 1

- Adversary has polynomial time computation power (N^c where N is security parameter and c is constant)
- Algorithm is secure if the perceived level of computation required to defeat it using the best *known* attack exceeds the computational resources of an adversary

Types of Symmetric Key Ciphers

Block cipher:

- process one input block at a time
- produce one output block for each input block
- common block sizes: 64, 128 and 256 bits

Stream cipher:

- process input elements continuously
- produces one element at a time
- common element size: 1 bit or 1 byte
- operation is generally eXclusive OR (XOR) between input elements and stream cipher key

Advanced Encryption Standard (AES)

- Developed by Joan Daemen and Vincent Rijmen from Belgium – also known as Rijndael algorithm
- Designed for:
  - simplicity
  - flexibility
  - efficiency
- fast for both implementations in software and hardware, and code compactness on many CPUs
- Published as a standard in FIPS 197: Advanced Encryption Standard [2]

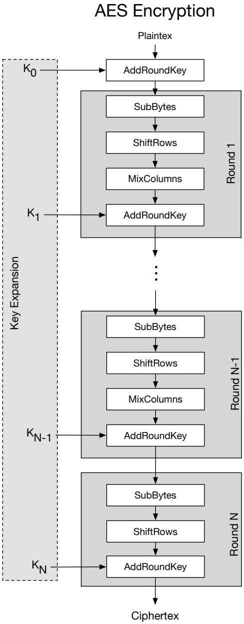
[2] FIPS 197 Advanced Encryption Standard

AES Properties

- Operates on 128-bit blocks
  - Larger block size contributes to security (better diffusion)
- supports three different key lengths: 128, 192 and 256 bits (same block size)
  - Larger key size contributes to security (better confusion, harder to brute force)
- has 10, 12, and 14 rounds of operation respective to the key size
- Rijndael in general is flexible enough to work with key and block size of any multiple of 32 bit with minimum of 128 bits and maximum of 256 bits.

AES Overall Encryption Process

- Consists of N rounds:
  - 10 for a 16-byte key;
  - 12 rounds for a 24-byte key;
  - 14 rounds for a 32-byte key.
- The first $N - 1$ rounds consist of:
  - SubBytes,
  - ShiftRows,
  - MixColumns,
  - AddRoundKey
- The final round N contains
  - SubBytes,
  - ShiftRows,
  - AddRoundKey



AES State

- Each round function takes one or more 4×4 matrices as input and produces a 4×4-byte matrix as output.
- The 4×4 bytes matrix is the state
- The state:
  - at the beginning is the plaintext block (filled column-wise)
  - during round operations contains intermediate results
  - at the end is the ciphertext block
- For 128-bit (16-byte) block $P = p_0 p_1 \ldots p_{15}$:

p_0	p_4	p_8	p_{12}
p_1	p_5	p_9	p_{13}
p_2	p_6	p_{10}	p_{14}
p_3	p_7	p_{11}	p_{15}

MONASH University

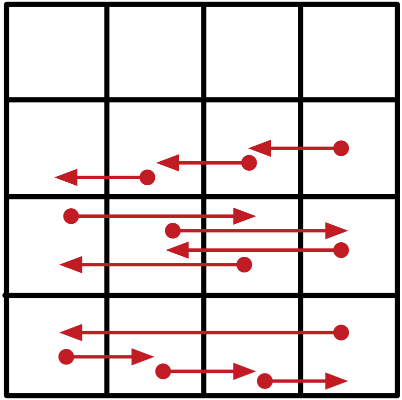
AES Round Functions: SubBytes

- SubBytes: byte substitution using a fixed 16×16 bytes S-box on every byte of state
  - Leftmost hex digit specifies the row
  - Rightmost hex digit specifies the column
  - Byte value at row and column substitutes the input byte

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

AES Round Functions: ShiftRows

- ShiftRows (simple permutation):
  - 1st row: No shift
  - 2nd row: 1-Byte left circular shift
  - 3rd row: 2-Byte left circular shift
  - 4th row: 3-Byte left circular shift



- MixColumns: matrix multiplication in GF(2^8) modulo $m(x) = x^8 + x^4 + x^3 + x + 1$:
  - State matrix is multiplied by a constant matrix
  - each element of the state matrix is a polynomial in GF(2^8)
  - addition is XOR of the coefficients of polynomials in GF(2^8)
  - multiplication of elements are accelerated using repeated shift and XOR

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

- $b_{0,0} = (02 \times a_{0,0} \oplus 03 \times a_{1,0} \oplus a_{2,0} \oplus a_{3,0}) \bmod m(x)$
- $b_{1,0} = (a_{0,0} \oplus 02 \times a_{1,0} \oplus 03 \times a_{2,0} \oplus a_{3,0}) \bmod m(x)$
- and so on ...

AES Round Functions: AddRoundKey

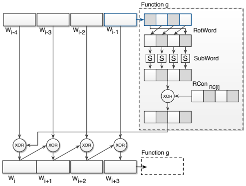
- AddRoundKey: XOR state with the round key

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \bigoplus \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

- $b_{0,0} = a_{0,0} \oplus k_{0,0}$
- $b_{1,0} = a_{1,0} \oplus k_{1,0}$
- and so on ...

AES Key Expansion

- key expansion function generates N + 1 round keys
- each round key is a distinct 4×4 bytes matrix
- expanded key is an array of 32-bit words W_i:
  - 128-bit key, 10 rounds: 44 words
  - 192-bit key, 12 rounds: 52 words
  - 256-bit key, 14 rounds: 60 words
- key expansion for 128-bit key

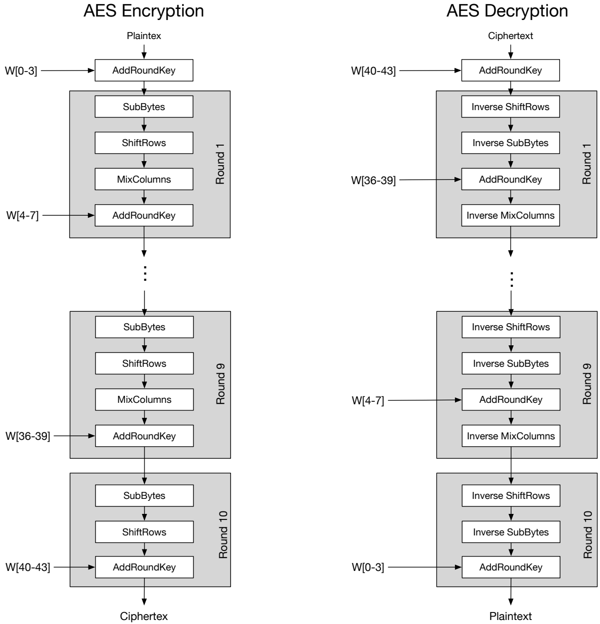


AES Key Expansion Algorithm

Key expansion algorithm:

- The first four/six/eight words are initialised with 128/192/256 bits key
- takes W_{i-4}, W_{i-3}, W_{i-2}, W_{i-1} from previous round as input
- produces $g(W_{i-1})$ where g has three sub-functions:
  - RotWord: 1-Byte left circular shift
  - SubWord: substitutes each byte using AES S-Box
  - XOR with 32-bit Round Constant RC[j] where:
    - $RC[1] = x^0$ (i.e. 1)
    - $RC[2] = x$ (i.e. 2)
    - $RC[j] = x \cdot RC[j-1] = x^{j-1}, j > 2$
- produce for 128-bit key ($N_k = 4$):
  - $W_i = W_{i-4} \oplus g(W_{i-1})$
  - $W_{i+1} = W_{i-3} \oplus W_i$
  - $W_{i+2} = W_{i-2} \oplus W_{i+1}$
  - $W_{i+3} = W_{i-1} \oplus W_{i+2}$
- $N_k = 6$ for 192 and $N_k = 8$ for 256-bit key

AES Encryption / Decryption Process



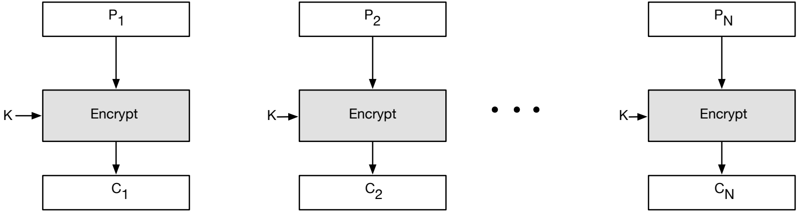
Modes of Operation

- Block ciphers encrypt fixed size blocks
  - e.g. AES encrypts 128-bit blocks with 128-bit key
- How to en/decrypt arbitrary amounts of data in practice?
- NIST SP 800-38A defines 5 modes of operations
  - Electronic Codebook (ECB) mode
  - Cipher Block Chaining (CBC) mode
  - Cipher Feedback (CFB) mode
  - Output Feedback (OFB) mode
  - Counter (CTR) mode
- Covers both block cipher and stream cipher
- any block cipher can be used in any mode

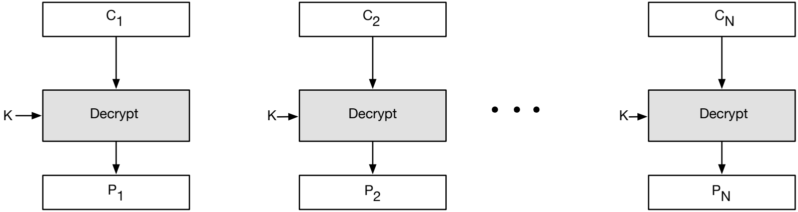
ECB (Electronic Codebook) mode

- Simplest mode of operation
- One block is processed at a time
  - input data is padded out to become an integer multiple of block size
  - each block is encrypted and decrypted independently
  - lost data blocks do not affect decryption of other blocks
  - error is not propagated, limited to single block
  - All blocks are encrypted with the same key
- Concern:
  - same plaintext produces same ciphertext under the same key
    - does not hide pattern
    - if message contains repetitive elements, these elements can be identified
    - traffic analysis is possible

ECB (Electronic Codebook) mode



(a) Encryption



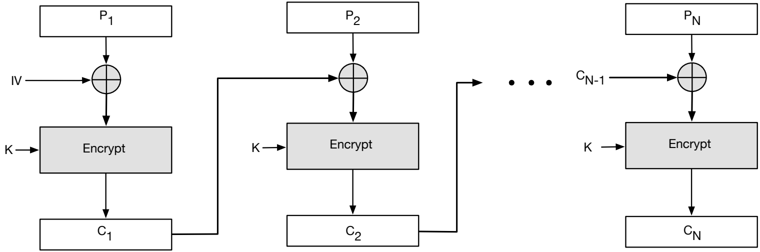
(b) Decryption

CBC (Cipher Block Chaining) mode

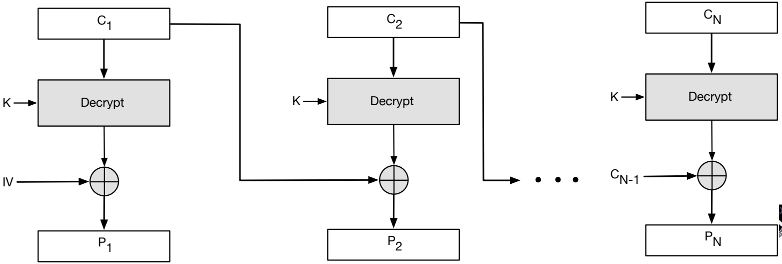
- Each plaintext block is XORed with the previous ciphertext block and encrypted
  - first plaintext block is XORed with an Initialization Vector (IV)
  - same encryption key is used for each block
  - $c_0 = IV, c_i = E(k, c_{i-1} \oplus p_i), 1 \leq i \leq n$
- More secure
  - better hides repetitive patterns
    - current plaintext block is affected by previous ciphertext block prior to encryption
  - different IV for different messages with the same key
    - IV need not be secret but must be unpredictable
- Disadvantage:
  - encryption of a data block becomes dependent on all the blocks prior to it
  - a lost block of data will prevent decoding of the next block of data

MONASH University

CBC (Cipher Block Chaining) mode



(a) Encryption



Stream Ciphers

- process message bit by bit (as a stream)
- have a pseudo random keystream $(k_1, k_2, \ldots, k_n)$
- bitwise XOR of plaintext with key stream
- to decrypt XOR with the same key stream
- randomness of stream key removes statistical properties in message
  - $c_i = p_i \oplus k_i$
- but must never reuse stream key
  - otherwise can potentially recover messages

Stream Cipher Properties

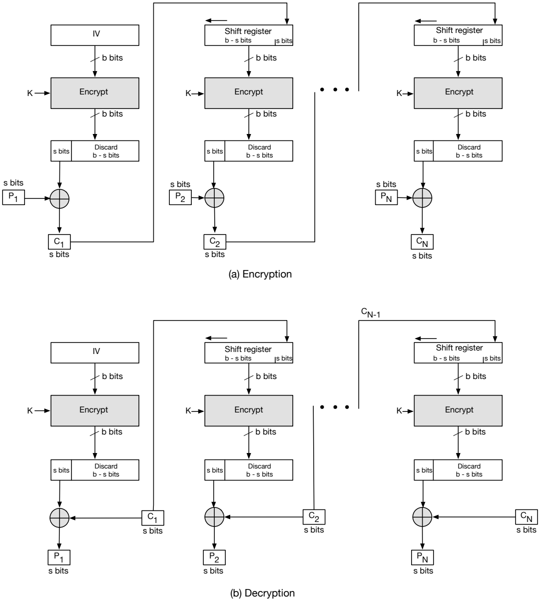
- some design considerations for stream cipher key are:
  - long period with no repetitions
  - statistically random
  - non-linear complexity (pseudo-random function)
- when properly designed, can be as secure as a block cipher with same size key
- but usually simpler and faster
- Stream cipher modes of operation:
  - CFB
  - OFB
  - CTR

CFB (Cipher feedback) mode

- Message is treated as a stream of bits
- Stream cipher does not require any padding
- Added to (XORed with) the output of the block cipher
- Result is fed-back for next stage (hence the name)
- standard allows variable number of bits (1,8 or 128) to be fed-back
  - denoted by CFB-1, CFB-8, CFB-128 etc.
- It is most efficient to use all 128 bits (CFB-128)
  - $c_0 = IV$
  - $c_i = p_i \oplus E_k(c_{i-1}),\ i \geq 1$

MONASH University

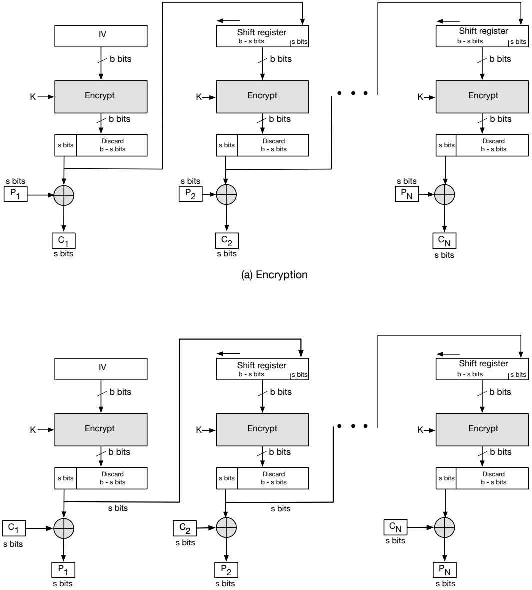
CFB (Cipher feedback) mode



OFB (Output feedback) mode

- Message is treated as a stream of bits
- Similar to CFB, but
  - quantity XORed with each plaintext block is generated independently of both the plaintext and ciphertext
  - output of cipher block is XORed with message
- Feedback is independent of message
- more efficient to use all 128 bits (OFB-128)
  - $x_0 = IV$
  - $c_i = p_i \oplus \overbrace{E_k(x_{i-1})}^{x_i}, i \geq 1$
- uses: stream encryption over noisy channels

OFB (Output feedback) mode



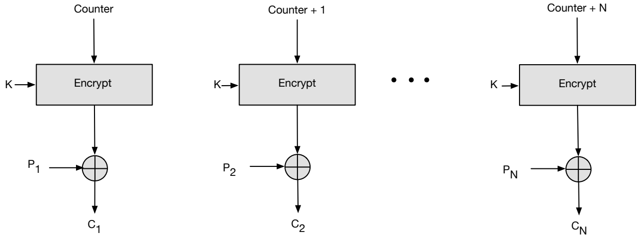
(a) Encryption

(b) Decryption

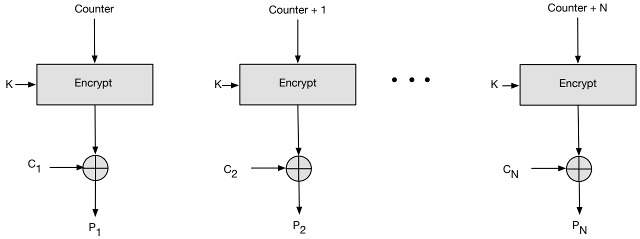
Counter (CTR) mode

- similar to OFB but encrypts using counter value rather than any feedback value
- must have a different counter value for every plaintext block (never reused)
  - $Counter = IV$
  - $c_i = p_i \oplus E(k, Counter + i - 1),\ i \geq 1$
- uses: high-speed network encryptions

Counter (CTR) mode



(a) Encryption



(b) Decryption

Advantages and Limitations of CTR

- efficiency
  - can do parallel encryptions in h/w or s/w
  - can pre-process in advance of need
  - good for bursty high speed links
- random access to encrypted data blocks
  - e.g. to decrypt block c_i: $p_i = c_i \oplus E_k(\text{Counter} + i - 1)$
- must ensure never reuse counter values under the same key, otherwise could break

Authenticated Encryption (ECB mode problem)

Does symmetric encryption provide message authenticity and integrity?

Consider the following simple scenario in ECB mode:

- Message $m : m_1, m_2, \ldots, m_t$ where $|m_i| = b$ bits (block size of the block cipher)
- Cipher: $c : c_1, c_2, \ldots, c_t$ where $c_i = E(K, m_i)$
- The cipher is transmitted over the network as $c_1||c_2||\ldots||c_t$
- The cipher is intercepted by attacker and modified as: $c_2||c_1||\ldots||c_t$
- Will this be detected at receiver?
- How about CBC and CTR modes?

MONASH University

Authenticated Encryption (CBC mode problem)

Consider the following simple scenario in CBC mode:

- Message $m : m_1, m_2, \ldots, m_t$ where $|m_i| = b$ bits (block size of the block cipher)
- Cipher: $c : c_0, c_1, c_2, \ldots, c_t$ where $c_0 = IV$ and $c_i = E(K, c_{i-1} \oplus m_i), i \geq 1$
- The cipher is transmitted over the network as $c_0||c_1||c_2|| \ldots ||c_t$
- can the attacker change the ciphertext in such a way that the recovered message is changed under attacker's control?
- what the attacker needs?

Authenticated Encryption (CTR mode problem)

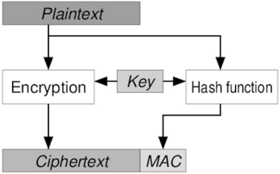
Consider the following simple scenario in CTR mode:

- Message $m : m_1, m_2, \ldots, m_t$ where $|m_i| = b$ bits (block size of the block cipher)
- Cipher: $c : c_0, c_1, c_2, \ldots, c_t$ where $c_0 = IV$ and $c_i = E(K, IV + i - 1) \oplus m_i, i \geq 1$
- can the attacker change the ciphertext in such a way that the recovered message is changed under attacker's control?
- what the attacker needs?

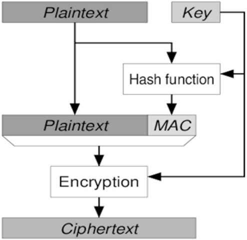
How Can We add Integrity and Authentication?

Old School solutions - Encrypt and MAC - Encrypt then MAC - MAC then Encrypt

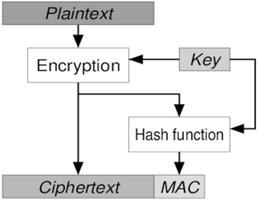
Encryption and Seperate Authentication



Encrypt and Mac



Mac then Encrypt



Encrypt then MAC

Galois Counter Mode (GCM): Overview

- an Authenticated Encryption with Associated Data (AEAD) mode of operation
- uses Encrypt-then-MAC method
- can use a block cipher with 128-bit block size (e.g. AES) in its inner components
- uses a variation of Counter mode
- provides authenticity (in symmetric notion) of confidential data (up to 64 GB per invocation)
- standardised in NIST SP 800 38d (NIST version is presented here)
- GCM encryption:
  - encrypts the confidential data
  - produces an authentication tag on both confidential data and any additional non-confidential data
- Security consideration: IV must not be reused (under the same key) otherwise secrecy of the messages may be compromised

GCM Design: Symbols

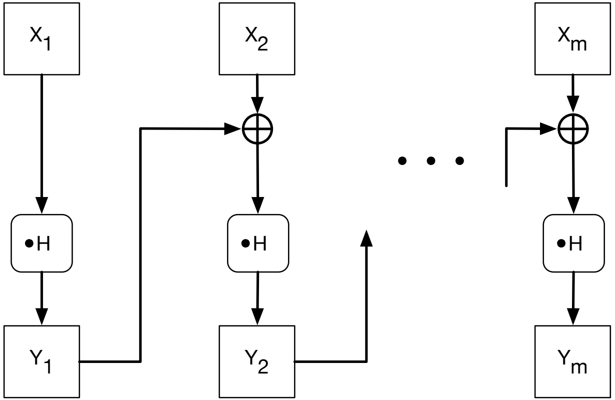
- A: the additional authenticated data
- C: ciphertext
- H: hash subkey
- ICB: Initial Counter Block
- IV: Initialisation Vector
- K: block cipher key
- P: plaintext
- T: authentication tag
- t: bit length of the authentication tag
- 0^s: bit string consists of s 0 bits

- R: constant within the algorithm for the block multiplication operation
- $E_K(X)$: output of block cipher encryption on block X using key K
- $GCTR_K(ICB,X)$: output of GCTR function on block X using key K and initial counter block ICB
- $GHASH_H(X)$: output of GHASH function on block X under the hash subkey H
- $X \bullet Y$: product of two blocks X and Y as elements of certain binary Galois field (polynomial in $GF(2^{128})$)

GCM Design: GHASH

- GHASH is a keyed hash function
  - however is **NOT** on its own a cryptographic hash function
  - it should only be used in the context of GCM
- length of X is a multiple of 128-bit
- the hash subkey $H = \mathrm{E_K}(0^{128})$
- $X \bullet Y$ is done in GF(2^{128}) modulo polynomial $g(Z) = Z^{128} + Z^7 + Z^2 + Z + 1$
- in Galois filed addition is simply XOR
- in effect GHASH calculates: $X_1 \bullet H^m \oplus X_2 \bullet H^{m-1} \oplus \cdots \oplus X_{m-1} \bullet H^2 \oplus X_m \bullet H \bmod g(Z)$
- numbers in NIST document are represented in *little endian* format ($g(Z) \equiv R = 11100001 \mid\mid 0^{120}$)

GHASH Diagram

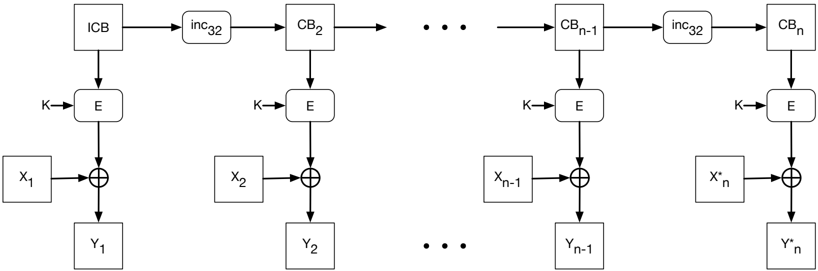


$GHASH_H(X_1 \| X_2 \| \ldots \| X_m) = Y_m$

GCM Design: GCTR

- the inc_{32} function increments the least significant 32 bits of the input value
- $CB_1 = ICB$
- X^*_n (and correspondingly Y^*_n) may be less than 128 bits
- GCTR uses a block cipher with 128-bit block size as key stream generator
- Encryption similar to counter mode is the XOR of the key stream with plaintext blocks

GCM Design: GCTR Diagram

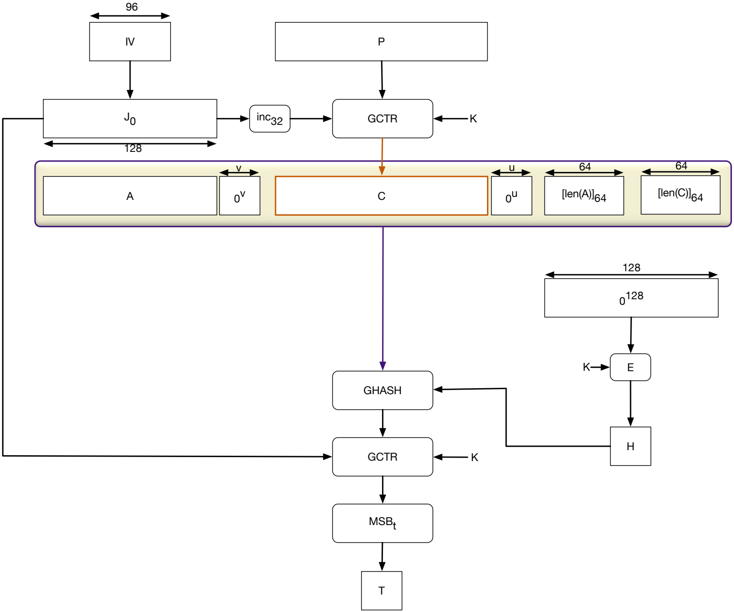


$GCTR_K(ICB, X_1 \| X_2 \| \dots \| X^*_m) = Y_1 \| Y_2 \| \dots \| Y^*_m$

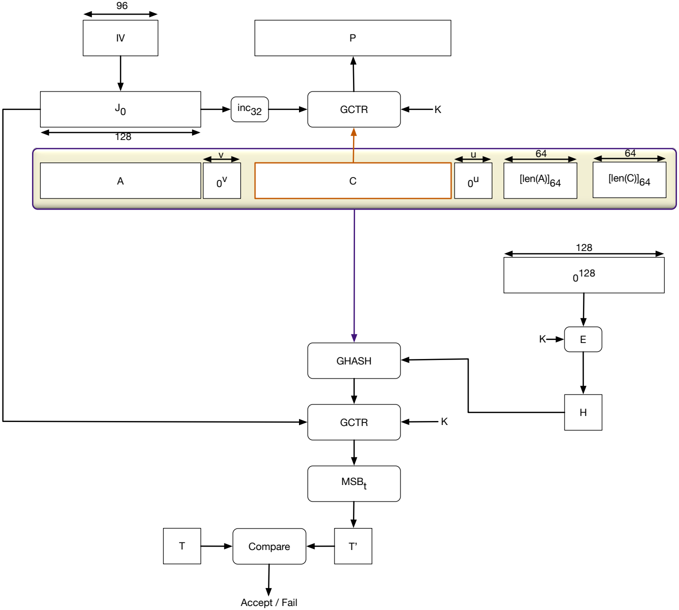
GCM Overall Algorithm

- J_0=IV || 0^{31} || 1 (if len(IV)=96)
  - if len(IV) $\neq$ 96 it will be padded with zero and length of IV and passed to GHASH
    - s = 128 $\cdot \lceil len(IV)/128 \rceil$ - len(IV)
    - J_0=GHASH$_H$(IV || 0^{s+64} || $[\text{len(IV)}]_{64}$)
- C=GCTR$_K$(inc$_{32}$(J_0), P)
- A is padded with zero to the closest multiple of 128 bits (v bits)
  - v=128 $\cdot \lceil len(A)/128 \rceil - len(A)$
- C is padded with zero to the closest multiple of 128 bits (u bits)
  - u=128 $\cdot \lceil len(C)/128 \rceil - len(C)$
- S = GHASH$_H$(A || 0^v || C || 0^u || $[\text{len(A)}]_{64}$ || $[\text{len(C)}]_{64}$)
- T = MSB$_t$(GCTR$_K$(J_0,S))
- Return ciphertext C and authentication tag T

GCM Overall Structure (Encryption)



GCM Overall Structure (Decryption)



Counter Mode with CBC-MAC (CCM): Overview

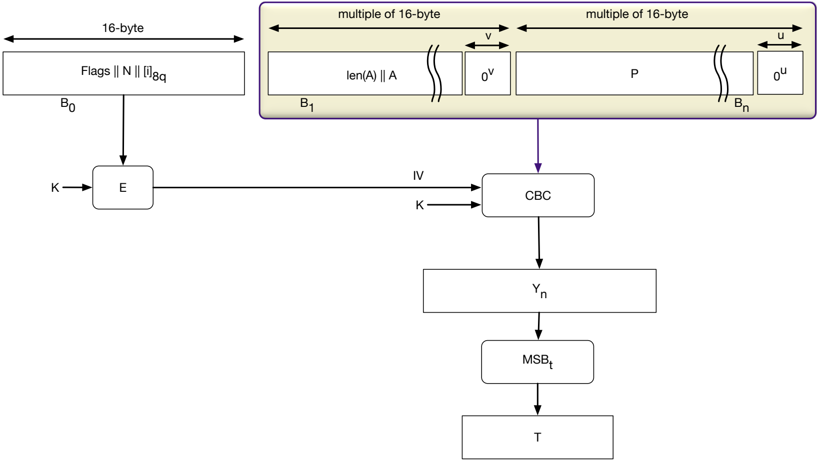
- an Authenticated Encryption with Associated Data
- uses MAC-then-Encrypt method
- Counter mode for encryption
- CBC MAC for authentication
- Proposed in RFC 3610
- Standardised in NIST SP 800 38c
- Nonce (IV) must not be repeated (under the same key) otherwise security is compromised

CCM Overall Design

- create the blocks $B_0, B_1, \ldots, B_n$ where:
  - $B_0 =$ Flags || N || Q
  - Flags (1 byte): Reserved || Adata || $[(t\text{-}2)/2]_3$ || $[q\text{-}1]_3$
    - Reserved bit is set to 0
    - Adata (1 bit): whether associated data is used in MAC
    - t: length of the authentication tag (represented in 3 bits, cannot be zero)
    - q: the size of the length field (bytes), how many bytes used to store the message length
  - N (15-q bytes): random IV (nonce)
  - Q (q bytes): Message length
  - if associated data is present (Adata=1) then:
    - encode length of associated data and concatenate the length and associated data and pad with zero (become multiple of 128-bit) append the result to B_0
  - append the message blocks after (optional) associated data
    - pad with zero to closest multiple of 128-bit block
- generate authentication tag T for blocks $B_0, B_1, \ldots, B_n$
  - $\mathrm{Y} = \mathrm{CBC}_K(B_0, B_1, \ldots, B_n)$
  - $\mathrm{T} = \mathrm{MSB_t}(\mathrm{Y_n})$

MONASH University

CCM: CBC-MAC

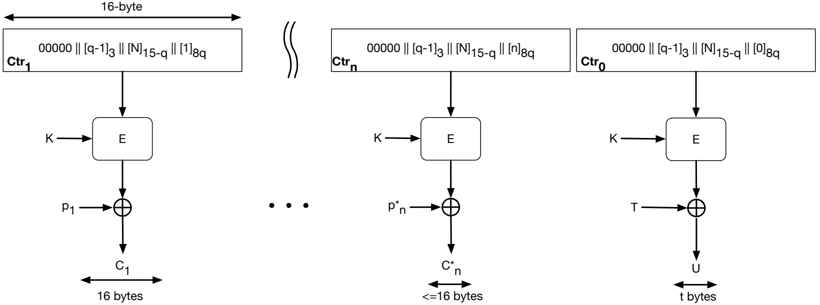


16-byte
Flags || N || [i]8q
B0
multiple of 16-byte
multiple of 16-byte
v
u
len(A) || A
0v
P
0u
B1
Bn
K
E
IV
K
CBC
Yn
MSBt
T

CCM: Counter

- create $\mathrm{Ctr_0}$, $\mathrm{Ctr_1}$, ..., $\mathrm{Ctr_n}$
  - n=$\lceil$ len(P)/16 $\rceil$ (16-byte blocks)
  - $\mathrm{Ctr_i}$= 00000 || $[\mathrm{q\text{-}1}]_3$ || $[\mathrm{N}]_{15\text{-}q}$ || $[\mathrm{i}]_{8q}$
  - q: number of bytes used to store the message length in byte (included in MAC calculation in block B_0)
    - e.g. q=3, 3 bytes will be used, if message is 4096 bits (512 bytes) then 512 is represented in binary in 3 bytes
- create blocks $S_0, S_1, \ldots, S_n$:
  - S_j=$\mathrm{E}_K(\mathrm{Ctr_j})$
- return $C = (P \oplus MSB_{len(P)}(S_1||S_2||\ldots||S_n))||(T \oplus MSB_t(S_0))$

CCM: Counter (diagram)



CHACHA20 Cipher Overview

- a high-speed stream cipher first proposed by Daniel J. Berntein[3]
- standardised by IETF in RFC 8439
- not sensitive to timing attacks (in contrast with AES)
- when combined with POLY-1305 becomes an AEAD construction
- refer to RFC for references on security of ChaCha (and Salsa)

[3] ChaCha, a variant of Salsa20

CHACHA20 Algorithm

- operates on a state matrix of 4×4 32-bit unsigned integers
- elements are referred to by their indices

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

- input to the algorithm:
  - a 256-bit (eight 32-bit little-endian integers) key
  - a 32-bit little-endian integer block count
  - a 96-bit (three 32-bit little-endian integers) nonce
- output is 64 bytes of key stream

CHACHA20 State Initialisation

- state initialisation:
  - first four words (0-3) are constants (c):
    - 0x61707865, 0x3320646e,0x79622d32, 0x6b206574
  - next eight words (4-11) are the key (k)
  - word 12 is block counter (b)
  - words 13-15 are the nonce (n)

```
cccccccc cccccccc cccccccc cccccccc
kkkkkkkk kkkkkkkk kkkkkkkk kkkkkkkk
kkkkkkkk kkkkkkkk kkkkkkkk kkkkkkkk
bbbbbbbb nnnnnnnn nnnnnnnn nnnnnnnn
```

CHACHA20 Round Operations

- the cipher performs 20 rounds where each is comprised of 4 quarter rounds
- the Quarter Round performs the following basic operations
  - operates on four 32-bit unsigned integers: a, b, c, d
    - $a = a + b \pmod{2^{32}}$, $d = d \oplus a$, $d = ROTL_{16}(d)$
    - $c = c + d \pmod{2^{32}}$, $b = b \oplus c$, $b = ROTL_{12}(b)$
    - $a = a + b \pmod{2^{32}}$, $d = d \oplus a$, $d = ROTL_{8}(d)$
    - $c = c + d \pmod{2^{32}}$, $b = b \oplus c$, $b = ROTL_{7}(b)$

MONASH University

CHACHA20 QUARTERROUND

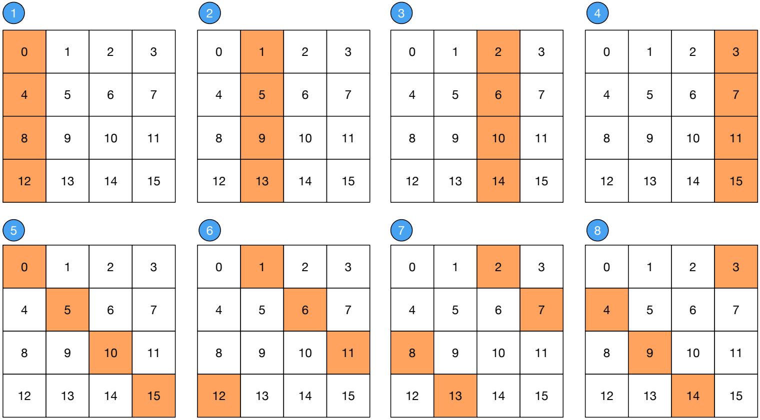
- the QUARTERROUND(w,x,y,z) operates on elements indexed as w, x, y, z
- the 20 rounds are the 10 iterations of the following two rounds (8 quarter rounds)

```
QUARTERROUND(0, 4, 8, 12)
QUARTERROUND(1, 5, 9, 13)
QUARTERROUND(2, 6, 10, 14)
QUARTERROUND(3, 7, 11, 15)
QUARTERROUND(0, 5, 10, 15)
QUARTERROUND(1, 6, 11, 12)
QUARTERROUND(2, 7, 8, 13)
QUARTERROUND(3, 4, 9, 14)
```

- the first four are "column rounds"
- the second four are "diagonal round"
- at the end of 20 rounds (10 iterations) the words of the initial state is added modulo 2^{32} to the words of the output state

MONASH University

- 10 iterations of the QUARTERROUND performed as follows



CHACHA20 Encryption

- choose a counter
- generate a 64-byte key stream using the 256-bit key, counter and nonce
- encrypt (XOR) 64 bytes of the plaintext with key stream (or what remains of plaintext)
- increment the counter and continue until all plaintext bytes are encrypted

POLY1305 Overview

- a one-time authenticator designed by D. J. Bernstein
- input:
  - 256-bit one-time key
  - a message of arbitrary length
- output:
  - 128-bit tag
- in original article[4] AES is used to encrypt the nonce as a method for pseudorandomness
  - CHACHA20 can be used to generate key pseudorandomly
- key must only be used once (the algorithm is biased and key reuse compromises security)

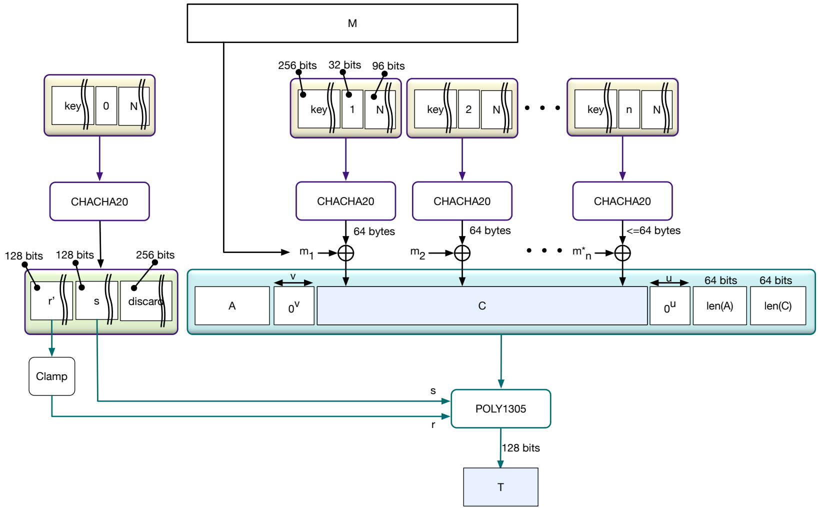
[4] The Poly1305-AES message-authentication code

POLY1305 Algorithm

- divide 256-bit key into two parts: `r` and `s`
  - the pair `(r,s)` must be unique and unpredictable for each invocation of algorithm
  - `r` may be a constant but must have a certain format
    - `r = r & 0x0ffffffc0ffffffc0ffffffc0fffffff` (clamped to have the required format)
- set the constant prime to $P = 2^{130} - 5$
  - `P=3fffffffffffffffffffffffffffffffb`
- set accumulator variable `Acc=0`
- divide the message to 16-byte blocks (last block may be shorter) and do the following for each block
  - read the block as little-endian number
  - add 2^{128} to a 16-byte block (or $2^{120} \ldots 2^{8}$ for shorter blocks accordingly)
  - if not 17 bytes pad with zero
  - set `Acc=((Acc+block) * r) mod P`
- set `Acc = Acc + s`
- return least significant 128 bits of `Acc` as tag

MONASH University

- first generate the one-time key for POLY1305:
  - use 256-bit integrity key as a key for CHACHA20
  - set block counter to zero
  - use a counter method for nonce (nonce must be unique per invocation under the same key)
  - run CHACHA20 which results in 512-bit state
    - use the first 128 bits as `r` and the next 128 bits as `s`
    - discard the remaining 256 bits
- `C=CHACHA20(K, counter=1, N, M)`
- `T=POLY1305(A || padding || C || padding || len(A) || len(C))`
- return `C,T`


M
256 bits
32 bits
96 bits
key
0
N
key
1
N
key
2
N
key
n
N
CHACHA20
CHACHA20
CHACHA20
CHACHA20
64 bytes
64 bytes
<=64 bytes
m1
m2
m*n
128 bits
128 bits
256 bits
r'
s
discard
A
v
0v
C
u
0u
64 bits
64 bits
len(A)
len(C)
Clamp
s
r
POLY1305
128 bits
T

References

- RFC4949 Internet Security Glossary, Version 2
- Handbook of applied cryptography, Chapter 1
- NIST SP 800-38A Recommendation for Block Cipher Modes of Operation
- NIST SP 800 38d Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
- RFC 3610: Counter with CBC-MAC (CCM)
- NIST SP 800 38c Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality
- RFC 8439: ChaCha20 and Poly1305 for IETF Protocols
- ChaCha, a variant of Salsa20
- The Poly1305-AES message-authentication code
- FIPS 197 Advanced Encryption Standard
- Cache-Collision Timing Attacks Against AES