# CO 487 Course Notes

## Applied Cryptography

**Michael Socha**

**University of Waterloo**

**Winter 2019**

# Contents

# 1 Course Overview

This course is an applied introduction to modern cryptography. Topics covered include:

- Symmetric-key encryption
- Hash functions
- Authenticated encryption
- Public-key encryption
- Signature schemes
- Key establishment
- Key management
- Examples of deployed cryptography (e.g. SSL, cryptocurrencies, WPA)

# 2 Introduction - What is Cryptography?

Information security (also known as cybersecurity) deals with protecting information assets from unauthorized acquisition, damage, disclosure, manipulation, loss, or use. Cryptography deals with the mathematical, algorithmic and implementation aspects of information security.

Cybersecurity more broadly includes the study of computer security, network security and software security.

## 2.1 Goals of Cryptography

In short, cryptography is about securing communications in the face of malicious adversaries. When describing cryptographic scenarios, Alice and Bob are used to indicate two parties who wish to communicate with one another across some channel, while Eve is a malicious adversary. Eve may attempt to read or modify the data being transmitted.

The main goals of cryptography are to provide:

- **Confidentiality:** Keeps data secret from unauthorized entities.
- **Data integrity:** Ensures data has not been altered by unauthorized means.
- **Data origin authentication:** Determines the sender of data.
- **Non-repudiation:** Provides proof of data origin and integrity, which can be used to prevent senders from disputing a previous action.

# 3 Symmetric-Key Encryption

## 3.1 Introduction

A symmetric-key encryption scheme (SKES) consists of:

- $M$ - the plaintext space
- $C$ - the ciphertext space
- $K$ - the key space
- a family of encryption functions - $\forall k \in K, E_k : M \to C$
- a family of decryption functions - $\forall k \in K, D_k : C \to M$, such that $\forall k \in K, \forall m \in M, D_k(E_k(m)) = m$

The idea behind using SKES to achieve confidentiality is:

1. Alice and Bob agree on a secret key $k$ (note that this cannot be done over an unsecured channel, or Eve may read the key).
2. Alice computes $c = E_k(m)$, and sends Bob ciphertext $c$, which is an encrypted form of the plaintext $m$.
3. Bob computes $m = D_k(c)$ to retrieve the plaintext $m$.

## 3.2 SKES Security

A security model defines the computational power of an adversary, and how they interact with the communicating parties. It is generally assumes that the adversary knows everything about the SKES besides the value of $k$.

### 3.2.1 Types of Attacks (Adversary Interactions)

- **Passive Attacks:**
  - Ciphertext-only attack - Eve only knows some ciphertext generated by Alice and Bob.
  - Known-plaintext attack - Eve knows some plaintext and the corresponding ciphertext.
- **Active Attacks:**
  - Chosen-plaintext attack - Eve can choose some plaintext and obtain the corresponding ciphertext.
- **Other Attacks:**

- Clandestine attacks - Bribery, blackmail, etc.
- Side-channel attackers - Involves monitoring of encryption and decryption equipment.

### 3.2.2 Computational Power of Adversary

- **Information-theoretic security:** Eve has infinite computational resources.
- **Complexity-theoretic security:** Eve has a polynomial-time Turing Machine.
- **Computational Security:** Eve has some fixed amount of computing power.

### 3.2.3 Goal of Adversary

The adversary may try to:

1. Recover the secret key.
2. Systematically recover plaintext from ciphertext, though without necessarily knowing the key.
3. Learn some information about plaintext based on its ciphertext (other than its length).

If an adversary can succeed at 1 or 2, then the SKES is said to be completely/totally insecure. If none of these goals can be achieved, the SKES is said to be semantically secure.

### 3.2.4 Secure SKES

A SKES is said to be secure if it is semantically secure when attacked with a chosen-plaintext attack by a computationally-bounded adversary.

## 3.3 Desirable Properties for SKES

- Efficient algorithms should be known for encryption and decryption functions.
- The secret key should be large enough to render exhaustive key search infeasible, though still relatively small.
- The scheme should be secure against everyone, even against the designers of the system (no security through obscurity).

## 3.4 Security Level

A cryptographic schema is said to have a security level of $l$ bits if the fastest known attack scheme takes around $2^l$ operations. For context, in this course, we consider $2^{40}$ operations very feasible, $2^{80}$ barely feasible, and $2^{128}$ infeasible. Security levels of 128 bits are desirable in practice.

## 3.5 Some Simple Ciphers

### 3.5.1 Simple Substitution Cipher

The simple substitution cipher involves replacing units of plaintext with ciphertext. For example, if the key is that $a$ maps to $Y$ and $b$ maps to $Z$, then $m = abbba$ results in $c = YZZZY$.

This cipher is totally insecure against a chosen-plaintext attack, since the mapping can be easily determined. When the mapping can be determined using character frequency analysis, it is also totally insecure against a ciphertext-only attack.

### 3.5.2 Polyalphabetic Cipher

The idea behind a polyalphabetic cipher is to allow a unit of plaintext to be encrypted to one of many possible ciphertext units. A common example is the Vigenere cipher, which uses modulo addition with a key. For example if the message is *thisisamessage* and the key is $CRYPTO$, and the ciphertext is computer as follows:

| m = | t | h | i | s | i | s | a | m | e | s | s | a | g | e |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| k = | C | R | Y | P | T | O | C | R | Y | P | T | O | C | R |
| c = | V | Y | G | H | B | G | C | D | C | H | L | O | I | V |

The Vigenere cipher is totally insecure against a chosen-plaintext attack, since the key can easily be figured out through modulo arithmetic.

### 3.5.3 One-Time Pad

The one-time pad uses modulo addition with a key at least as long as the message itself. For example, if the message is *message* and the key is $SMFJDLG$, the ciphertext can be computer as follows:

| m = | m | e | s | s | a | g | e |
|---|---|---|---|---|---|---|---|
| k = | S | M | F | J | D | L | G |
| c = | F | Q | X | C | D | R | K |

The one-time pad is like the Vigenere cipher, but a different key is used to encrypt each message. This makes the one-time pad semantically secure against a ciphertext-only attack, even by an adversary with unlimited resources. However, the one-time pad is impractical due to:

1. Its long key length
2. Key reuse introducing a vulnerability (Since $c_1 = m_1 + k$ and $c_2 = m_2 + k$, $c_1 - c_2 = m_1 - m_2$, so if one message is known, another can be computed.

## 3.6 Stream Ciphers

As an alternative to using a random key, a pseudorandom bit generator (PRBG) can be used to generate a key. The PBRG is seeded with a secret value shared between Alice and Bob, and then generates new "random-looking" keys based on its previous state.

### 3.6.1 RC4 Cipher

RC4 is a simple, fast cipher for which no catastrophic weakness has been found. It consists of two components, namely a key scheduling algorithm and a keystream generator.

The high-level idea of the key scheduling algorithm is to generate a random-looking permutation of 0 to 255 (stored in $S$) based on the secret key ($K$).

```
for i from 0 to 255
  S[i] := i // initialize to identity permutation

j := 0
for i from 0 to 255
  j := j + S[i] + K[i mod keylength] mod 256
  swap(S[i], S[j])
```

The keystream generator accepts the permutation of $S$ resulting from the key scheduling algorithm as input. The keystream generator continues to modify $S$ while producing a bytestream.

```
i := 0
j := 0
while we need to generate another keystream byte
  i := (i + 1) mod 256
  j := (j + S[i]) mod 256
  swap(S[i], S[j])
  t := S[i] + S[j] mod 256
  output S[t]
```

The keystream bytes are modulo added (i.e XORed) to the plaintext bytes to produce ciphertext bytes. There are known biases in the first few bytes of the keystream, so these bytes should be discarded.

RC4 was used for applications such as SSL and WEP. More recently, several weaknesses have been discovered, so RC4 is no longer widely used.

## 3.7 Block Ciphers

Block ciphers break up a plaintext into fixed-size blocks, and then encrypt the blocks one at a time. In contrast, stream ciphers encrypt the plaintext one character at a time (i.e. blocks are of length 1).

Some desirable properties of block ciphers are:

- **Diffusion:** Each ciphertext bit should depend on all plaintext bits.
- **Confusion:** The relationship between key bits and ciphertext bits should be complicated.
- **Algorithm simplicity**
- **Efficiency:** High speed of encryption and decryption.

### 3.7.1 Feistel Ciphers

Feistel ciphers are a class of block ciphers with the following parameters:

- $n$ - half the block length
- $h$ - number of encryption rounds
- $l$ - key length
- $M = \{0,1\}^{2n}, C = \{0,1\}^{2n}, K = \{0,1\}^{l}$

A key scheduling algorithm determines $h$ subkeys $k_1, k_2, ..., k_h$ from a key $k$. Each subkey $k_i$ defines a component function $f_i : \{0,1\}^n \to \{0,1\}^n$. Feistel ciphers encrypt by iteratively applying these component functions as follows:

- Plaintext is $m = (m_0, m_1)$
- Round 1 of encryption results in $(m_1, m_2)$, where $m_2 = m_0 \oplus f_1(m_1)$
- Round 2 of encryption results in $(m_2, m_3)$, where $m_3 = m_1 \oplus f_2(m_2)$
- . . .
- Ciphertext is $c = (m_h, m_{h+1})$

Decryption applies these functions in reverse order.

**New Data Seal (NDS)** is a type of Feistel cipher with $n = 64$ and $h = 16$. $k$ is a randomly selected function that converts one byte to another. Since there are 256 unique bytes, there are $256^{256} = 2^{2048}$ unique keys, so $l = 2048$. Subkeys $k_i$ are simply $k$ itself. The component function $f : \{0,1\}^{64} \to \{0,1\}^{64}$ is defined as follows:

1. Divide input $z$ into 8 bytes $(z_1, z_2, ..., z_8)$.
2. Divide each byte $z_j$ into two nibbles $(n_1^j, n_2^j)$.
3. Apply some publicly known operations $S_0$ to $n_1^j$ and $S_1$ to $n_2^j$ to produce new nibbles $p_1^j$ and $p_2^j$.
4. Let $z^*$ be the byte obtained by taking the first bit of every byte. Compute $t = k(z^*)$.
5. If the jth bit of $t$ is 1, then swap $p_1^j$ and $p_2^j$.
6. Permute the resulting 64 bits by some publicly known operation $P$.

The key of NDS can be determined after a chosen-plaintext attack of around 32,000 carefully selected inputs. Thus, NDS is totally insecure, which underlines the importance of using different subkeys for different rounds of a Feistel cipher.

**Data Encryption Standard (DES)** is a type of Feistel cipher with $n = 32$, $h = 16$ and $l = 56$. Unlike NDS, DES selects different subkeys for each round of encryption; each subkey is a selection of 48 bits of $k$.

Although DES has been important in the development of modern cryptography, its short key length makes it susceptible to brute-force attacks. Another problem is the small block size ($2 \cdot 32 = 64$), which means that a collision between ciphertext blocks is expected approximately every $\sqrt{2^{64}} = 2^{32}$ blocks.

### 3.7.2 Multiple Encryption

Multiple encryption involves encrypting a message multiple times. For example, if DES encryption is done twice with independent keys, then $l = 2 * 56 = 112$, seemingly reducing the feasibility of exhaustive key search.

However, so-called meet-in-the-middle attacks can be used to greatly reduce the security level of the encryption scheme. For example, if $c = E_{k2}(E_{k1}(m))$, then instead of performing $2^{2l}$ operations to find the key $(k_1, k_2)$, the result of $E_{k2}^{-1}(c)$ can be saved for all $2^l$ possible values of $k_2$. Then, $E_{k1}(m)$ can be run for all $2^l$ possible values of $k_1$, and then matches can be searched for such that $E_{k1}(m) = E_{k2}^{-1}(c)$. Thus, the security level of double encryption is roughly $2 \cdot 2^l = 2^{l+1}$, making it not much more secure than single encryption.

Triple encryption is more resistant to meet-in-the-middle attacks, requiring around $2^{2*56} = 2^{112}$ steps to break (i.e. "middle" here means two levels deep on one side and one level deep on the other). In light of its enhanced security, triple DES actually has some uses in production.

### 3.7.3 Modes of Operation

Modes of operation concern how to use a block cipher to encrypt some plaintext messages $m = m_1 m_2 ... m_t$. The simplest mode is called electronic codebook (ECB) mode, where plaintext blocks map to ciphertext blocks independently of one another. Since identical plaintexts result in identical ciphertexts, ECB mode is not semantically secure against chosen-plaintext attacks.

Cipher block chaining (CBC) mode involves each block of plaintext being XORed with the one encrypted before it. Since identical blocks of plaintext no longer map the the same ciphertext, CBC mode is secure against chosen-plaintext attacks.

## 3.8 Advanced Encryption Standard (AES)

AES is the most widely used production SKES. Established in 2001 by the US National Institute of Standards and Technology (NIST), AES is a block cipher using 128 bit blocks and supports key sizes of 128, 192, and 256 bits.

# 4 Hash Functions

A hash function is a mapping of a message of an arbitrary length to a message of fixed length. An n-bit hash function $H$ can be defined as $H : \{0,1\}^{\leq L} \rightarrow \{0,1\}^n$. Generally speaking, $L$ is very large while $n$ is fairly small. The output of a hash function is known as a hash value or message digest.

## 4.1 Desirable Properties

A few desirable properties hash functions for cryptography applications are:

- **Efficiency:** $H(x)$ can be computed quickly for all valid inputs $x$.
- **Preimage resistance:** Given a hash value $y \in \{0,1\}^n$, it is computationally infeasible to find a value $x \in \{0,1\}^{\leq L}$ such that $H(x) = y$ (i.e. the hash function is hard to reverse). This property allows the storage of hash values of sensitive data. For example, instead of storing passwords in plaintext, a login system can store hard-to-reverse hashes of passwords, and to login, the hash of the user's inputted password is compared to the stored hash.
- **Second preimage resistance:** Given an input $x \in \{0,1\}^{\leq L}$, it is computationally infeasible to find another input $x' \in \{0,1\}^{\leq L}$ such that $H(x') = H(x)$. This property can be used to detect if a message is edited through unauthorized means.
- **Collision Resistance:** It is computationally infeasible to find two distinct inputs $x$ and $x'$ such that $H(x') = H(x)$. Such pairs $(x, x')$ are known as collisions. Collision resistance requires second preimage resistance as a precondition. Also, if $H$ is somewhat uniform (i.e. outputs have a roughly equal amount of corresponding inputs), then collision resistance guarantees preimage resistance.

A hash function that is preimage resistant is called a one-way hash function. A hash function with collision resistance is called a collision-resistant hash function. A hash function that is both preimage resistant and collision resistant is called a cryptographic hash function.

## 4.2 Generic Attacks

Generic attacks on hash functions are attacks that do not exploit any properties specific to a hash function (i.e. the hash function is simply treated as a random function).

### 4.2.1 Finding Preimages

To find a preimage for a hash value $y \in \{0,1\}^n$, random values of $x \in \{0,1\}^{\leq L}$ need to be selected until $H(x) = L$. The expected number of values of $x$ tested before a match is found is $2^n$, so this attack is computationally infeasible for large enough values of $n$ (e.g. 128).

### 4.2.2 Finding Collisions

To find collisions, random values $x \in \{0,1\}^n$ can be saved along with $H(x)$. The expected number of values $x$ tested before a collision is found is roughly $\sqrt{2^n} = 2^{\frac{n}{2}}$.

## 4.3 Hash Functions from Block Ciphers

Hash functions can be constructed from block ciphers. An example of this is the Davies-Meyer hash function, which works by splitting a message into fixed-sized blocks that serve as keys for a block cipher, and then iteratively feeding the results of these block ciphers into one another. The Davies-Meyer hash function is defined as follows:

- Let $H_0$ be some initializing value $IV$
- Let $E_k$ be an m-bit block cipher with an n-bit key $k$
- Let a message $x$ be broken up into $t$ n-bit blocks: $x = x_1 x_2 ... x_t$
- $H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1}$ for $i = 0, 1, ..., t$, and $H(x) = H_t$

## 4.4 Merkle Meta-Method

The Merkle meta-method is a technique for building iterated hash function using so-called compression functions. If the compression function $f$ is collision-resistant, then so is the hash function $H$, which is defined as follows:

- Let $H_0$ be some initializing value $IV$
- Let $f$ be a compressing function $f : \{0,1\}^{n+r} \to \{0,1\}^n$
- Let input $x$, which has a bitlength of less than $2^r$, be divided into $t$ r-bit blocks: $x = x_1 x_2 ... x_t$
- Let $x_{t+1}$ be the right-justified value of $b$. $x_{t+1}$ is referred to as a length block.
- $H_i = f(H_{i-1}, x_i)$ for $i = 0, 1, ..., t+1$, and $H(x) = H_{t+1}$

The Merkle meta-method simplifies the problem of building collision-resistant hash functions to one of simply building collision-resistant compression functions.

## 4.5 Cryptographic Hash Functions in Production

MDx (e.g. MD4, MD5) is a family of 128-bit iterated hash functions. These functions have weak collision resistance and suffer from other vulnerabilities, so their use is declining.

The Secure Hash Algorithm (SHA) family of iterated hash functions developed by the NSA and NIST. SHA-1 is a 160 bit hash, but due to a collision detection algorithm that takes

around $2^{63}$ steps, it is being phased out in favor of the newer SHA-2 and SHA-3, which support custom hash lengths.

# 5 Message Authentication Code Schemes

## 5.1 Formulation

A message authentication code (MAC) is a family of functions $H_k : \{0,1\}^* \rightarrow \{0,1\}^n$ parameterized with an $l$ bit key $k$. Each function $H_k$ should be possible to efficiently compute, and $H_k(x)$ is known as the tag or MAC of $x$ for the key $k$.

## 5.2 Applications

MAC schemes can be used to provide data integrity and data origin authentication. The high-level idea is that Alice and Bob agree to some secret key $k \in \{0,1\}^l$. Alice can compute a tag $t = H_k(x)$ and send $(x,t)$ to Bob, and Bob verifies that $t = H_k(x)$. Since the adversary should not know the secret key $k$, this verification is used to confirm that the message was not altered, and if it is a new message, that it originated from Alice.

## 5.3 Security

Although the adversary does not know $k$, they are allowed to obtain $t$ for messages $(x,t)$ that were already sent. A MAC scheme is considered secure if it is computationally infeasible to compute $H_k(x_i)$ for some $x_i$ that has not yet been sent. For security purposes, each function in an ideal MAC scheme should act as a random function.

Two generic types of attacks on MAC schemes are:

- **Guessing the MAC** of the message $x$. If $H_k(x) \in \{0,1\}^n$ and $H_k$ acts as a random function, the probability of success is $\frac{1}{2^n}$.
- **Exhaustive key search**. If $k \in \{0,1\}^l$, then there are $2^l$ possible keys.

## 5.4 Block Cipher MACs

A common implementation of MACs using block ciphers is known as cipher block chaining MAC (CBC-MAC). The CBC-MAC algorithm is:

1. Divide a message $x$ into $n$ bit blocks $x_1, x_2, ..., x_n$.
2. Compute $H_1 = E_k(x_1)$, where $k$ is the secret key and $E$ is a block cipher.
3. For $i$ from 2 to $r$, compute $H_i = E_k(H_{i-1} \oplus x)$.
4. $H(x) = H_r$.

If $E$ is a secure encryption scheme, then CBC-MAC is a secure MAC scheme assuming that fixed-length inputs are used. If variable-length messages are allowed, then CBC-MAC is susceptible to the following chosen-message attack:

1. Select an arbitrary $n$ bit block $x_1$.
2. Obtain the tag $t_1 = E_k(x_1)$.
3. Obtain the tag $t_2 = E_k(t_1)$. $t_2$ is the tag of the 2-block message $(x_1, 0)$, since $t_2 = E_k(0 \oplus E_k(x_1)) = E_k(E_k(x_1)) = E_k(t_1)$.

Encrypted CBC-MAC is a variation of CBC-MAC where the last block is encrypted using a separate key $s$, and if $E$ is secure, encrypted CBC-MAC is secure for inputs of any length.

## 5.5 Hash Function MACs

Hash functions were not designed to be keyed, so it is not obvious how to use them to implement MACs. The high-level idea of "keying" hash functions is to incorporate the key in the input to the hash function.

Let $H$ be a hash function with an $n$ bit message digest, and let $n+r$ be the input blocklength of its compression function $f : \{0,1\}^{n+r} \to \{0,1\}^n$. Let $k \in \{0,1\}^n$ be the secret key, and let $K$ denote $k$ padded with $r - n$ 0s so that its bitlength is $r$. The key $k$ can be incorporated into hash functions as follows:

- **Secret Prefix Method:** $H_k(x) = H(K, x)$. This is insecure to a length-extension attack. Suppose $(x, H_k(x))$ is known, and suppose the bitlength of $x$ is a multiple of $r$. Then $H(x||y)$ can be computed for any $y$ without knowledge of $k$.
- **Secret Suffix Method:** $H_k(x) = H(x, K)$. This method appears to be secure, provided that $H$ is collision resistant.
- **Envelope Method:** $H_k(x) = H(K, x, K)$. No exploits have been found for the envelope method.

### 5.5.1 HMAC

HMAC stands for hash-based message authentication code. Given two $r$ bit string opad and ipad, HMAC is implemented as follows:

$$H_k(x) = H(K \oplus opad, H(K \oplus ipad, x))$$

If $H$ is a secure hash function, then HMAC is secure. It is used widely in practice, typically with SHA-256 as its hash function. HMAC is also commonly used as a key derivation function, where a secret key $k$ is used to compute several further session keys (e.g. $HMAC_k(1), HMAC_k(2), HMAC_k(3)$, etc). The keys appear to be randomly generated, so even if an adversary discovers information about one of them, the rest remain secret.

# 6 Authenticated Encryption

An encryption scheme can provide confidentiality, while a MAC scheme can provide data integrity and data origin authentication. Authenticated encryption combines an encryption scheme with a MAC scheme to provide all of these properties.

## 6.1 Common Implementations

### 6.1.1 Encrypt-and-MAC

For an encryption function $E_{k_1}$ and a MAC function $H_{k_2}$, Alice sends $(c, t) = (E_{k_1}(m), H_{k_2}(m))$ to Bob. Bob obtains $m = E_{k_1}^{-1}(c)$, and then verifies that $t = H_{k_2}(m)$. While simple and intuitive, encrypt-and-MAC can have security vulnerabilities since there is no guarantee of data integrity on the ciphertext.

### 6.1.2 Encrypt-then-MAC

For an encryption function $E_{k_1}$ and a MAC function $H_{k_2}$, Alice sends $(c, t) = (E_{k_1}(m), H_{k_2}(E_{k_1}(m)))$ to Bob. Bob verifies that $t = H_{k_2}(c)$, and then obtains $m = E_{k_1}^{-1}(c)$. Provided that $E$ and $H$ are from secure encryption and MAC schemes, then the encrypt-then-MAC method is secure.

### 6.1.3 AES Galois/Counter Mod (GCM)

AES-GCM authenticated encryption. Suppose the data to be authenticated, but not encrypted is $A = (A_1, A_2, \ldots, A_v)$, and the data to be both authenticated and encrypted is $M = (M_1, M_2, \ldots, M_u)$. Let the secret key be $k \in \{0,1\}^{128}$. AES-GCM is implemented as follows:

**Alice:**

1. Let $L = L_A || L_M$, where $L_A, L_M$ are the bitlengths of $A, M$.
2. Select $IV \in \{0,1\}^{96}$ and let $J_0 = IV || 0^{31} || 1$.
3. Encryption:
   (a) For $i = 1$ to $u$, $J_i = J_{i-1} + 1$ and $C_i = AES_k(J_i) \oplus M_i$.
4. Authentication:
   (a) Let $T = 0^{128}$
   (b) Compute $H = AES_k(0^{128})$.

(c) For $i = 1$ to $v$, $T = (T \oplus A_i) \cdot H$. This multiplication is treated as polynomial multiplication where some binary string $a = a_0 a_1 \dots a_{127}$ is represented as $a_0 + a_1 x + \dots + a_{127} x^{127}$.

(d) For $i = 1$ to $u$, $T = (T \oplus C_i) \cdot H$.

(e) $T = (T \oplus L) \cdot H$.

(f) Compute $t = AES_k(J_0) \oplus T$.

5. Output $(IV, A, C, T)$.

**Bob:**

1. Let $L = L_A || L_M$, where $L_A, L_M$ are the bitlengths of $A, M$.
2. Let $J_0 = IV || 0^{31} || 1$.
3. Authentication:
   (a) Let $T = 0^{128}$
   (b) Compute $H = AES_k(0^{128})$.
   (c) For $i = 1$ to $v$, $T = (T \oplus A_i) \cdot H$.
   (d) For $i = 1$ to $u$, $T = (T \oplus C_i) \cdot H$.
   (e) $T = (T \oplus L) \cdot H$.
   (f) Compute $t' = AES_k(J_0) \oplus T$.
   (g) Accept if $t' = t$, else reject.
4. Decryption:
   (a) For $i = 1$ to $u$, $J_i = J_{i-1} + 1$ and $M_i = AES_k(J_i) \oplus C_i$.
5. Output $(A, M)$

AES-GCM is secure, and because its encryption and decryption operations can be parallelized, it can be more efficient than algorithms such as encrypt-then-MAC. AES-GCM also supports authenticating but not encrypting pieces of data (e.g. a header section containing metadata).

# 7 Introduction to Public-Key Cryptography

## 7.1 Drawbacks of Symmetric-Key Cryptography

### 7.1.1 Key Establishment

Common ways for establishing keys in SKES are:

- **Point-to-point key distribution.** This involves Alice selecting a key and sending it to Bob over some secure channel, such as a trusted courier or in some face-to-face meeting. These methods do not scale well to an increased number of participants.
- **Using a trusted third part (TTP).** A key distribution center (KDC) can be used as a centralized store for keys, and users authenticate against it to retrieve keys. However, the KDC is a single point of failure and is an attractive target for attacks.

### 7.1.2 Key Management

A network of $n$ users has $\binom{n}{2} \in \theta(n^2)$ possible communication channels, so the number of key pairs to maintain grows quadratically to the number of users.

### 7.1.3 Implementing Non-Repudiation

Symmetric-key encryption schemes cannot be used to achieve non-repudiation on their own.

## 7.2 Public-Key Cryptography

It used to be widely believed in the cryptography community that in order for two parties to communicate securely over an insecure network, they need to first exchange some secret information. This was challenged by Ralph Merkle in 1974, when he demonstrated that two parties can establish a secret key by communicating over an authenticated but insecure channel. Merkle's first public-key cryptography construction, known as a Merkle Puzzle, is described as follows:

1. Alice creates $N$ puzzles $P_1, P_2, \ldots, P_N$, each of which take a substantial amount of time to solve. $N$ is very large. Solving a puzzle reveals a session key $sk_i$ and a serial number $n_i$.
2. Alice sends $P_1, P_2, \ldots, P_N$ to Bob.
3. Bob obtains $j \in \{1, 2, \ldots N\}$ at random and solves puzzle $P_j$ to obtain $sk_j$ and $n_j$.
4. Bob sends $n_j$ to Alice.
5. The secret session key is the one corresponding to $n_j$ (i.e. $sk_j$).

### 7.2.1 Formulation

In general, key pairs for public-key cryptography work as follows:

1. Each entity $A$ generates a key pair $(P_A, S_A)$. $P_A$ is known as the public key and is not secret, while and $S_A$ is known as the private key and should remain secret. It should be infeasible to recover $S_A$ from $P_A$.
2. To encrypt messages for Bob, Alice obtains an authentic copy of Bob's $P_B$, computes $c = E(P_B, m)$, where $E$ is some encryption function, and sends $c$ to Bob.
3. To decrypt $c$, Bob computes $m = D(S_B, c)$, where $D$ is some decryption function.

### 7.2.2 Digital Signatures

Public-key cryptography also supports signing messages. Signing a message works as follows:

1. Alice computes $s = Sign(S_A, m)$ and sends $m$ and $s$ to Bob, where Sign is some function used for digital signatures.
2. Bob obtains an authentic copy of $P_A$, and accepts if the output of $Verify(P_A, m, s)$ is acceptable, where Verify is some function for verifying digital signatures.

### 7.2.3 Advantages over Symmetric-key Cryptography

Public-key cryptography addresses many of the drawbacks of symmetric-key cryptography. In particular, public key cryptography has the following advantages over SKES:

- It does not require a secure communication channel for key establishment.
- Each user maintains a single key pair for decrypting messages from any other user.
- A signed message an be verified by anyone, since the secret key does not need to be known.
- Digital signatures can also be used to implement non-repudiation.

The main drawback of public-key cryptography is that is it typically more computationally intensive than symmetric-key cryptography.

### 7.2.4 Hybrid Schemes

Public-key and symmetric-key encryption schemes are often used together. For example, an encryption scheme could encrypt some symmetric key $k$ using public-key encryption.

# 8 RSA

## 8.1 RSA Public-Key Encryption Scheme

### 8.1.1 Key Generation

Alice generates a public-private key pair as follows:

1. Alice selects two prime numbers $p, q$.
2. Let $n = pq$, and let $\phi(n) = (p-1)(q-1)$.
3. Alice selects some $e$ such that $gcd(e, \phi(n)) = 1$.
4. Alice solves $ed = 1 \mod \phi(n)$.
5. Alice's public key is $(n, e)$, and the private key is $d$.

### 8.1.2 Encryption and Decryption

Bob encrypts some text $m$ to send to Alice as follows:

1. Bob obtains Alice's public key $(n, e)$.
2. Bob computes $c = m^e \mod n$.
3. Bob sends ciphertext $c$ to Alice.

Alice decrypts ciphertext $c$ as $m = c^d \mod n$.

### 8.1.3 Security

The security of the RSA encryption scheme is based on that the private key $d$ should be very computationally intensive to determine from public key $(n, e)$. Factoring $n$ into its prime factors $p, q$ would break the scheme, but no efficient method for doing so is known; the most efficient known general-purpose methods are subexponential to the number of bits. To maximize resistance to factoring attacks, $p$ and $q$ should be very large prime numbers of equal length.

## 8.2 RSA Signature Scheme

Bob signs a message $m$ as follows:

1. Bob computes $M = H(m)$, where $H$ is some hash function.
2. Bob computes $s = M^d \mod n$.
3. Bob sends the signed message $(m, s)$.

Alice verified a message $(m, s)$ as follows:

1. Alice obtains an authentic copy of Bob's public key $(n, e)$.
2. Alice computes $M = H(m)$.
3. Alice computes $M' = s^e \mod n$.
4. Alice accepts $(m, s)$ if and only if $M = M'$.

### 8.2.1 Security

The goal of an adversary attacking a signature scheme is to either:

- Totally break the scheme by recovering the private key.
- Forge a signature for a single message, which is known as existential forgery.

A signature scheme is said to be secure if it is existentially unforgeable by a computationally-bounded adversary launching a chosen-message attack.

The basic RSA signature scheme is actually not secure if SHA-256 is used for $H$. To ensure security, a full-domain hash (FDH) paradigm should be used, where a message $m \in \{0, 1\}^*$ is mapped to $[0, n - 1]$.