CS3235 - Computer Security
AY25/26, Y3S1
Notes

Sim Ray En Ryan

December 3, 2025

Contents

1 Lecture 7: Cryptography and Applications

1.1 Introduction to Cryptography

1.1.1 Classical Ciphers

- **Cryptography:** Defined as Encryption + Decryption, primarily for confidentiality.
- **Caesar Cipher:** A simple shift cipher, shifting letters by 3 places.
- **Keyed Shift Cipher:** Generalizes the Caesar cipher with a key k from 0 to 25. It is insecure as it can be broken by trying all 26 possible keys.
- **Substitution Cipher:** Uses a substitution table (bijection) to map each letter to another. This is also insecure.

1.1.2 Frequency Analysis

- This is an attack method used to break classical ciphers like the substitution cipher.
- It relies on the fact that certain letters and letter combinations (like E, T, A, O in English) appear more frequently than others. By matching the ciphertext's letter frequencies to standard English frequencies, the substitution table can be reverse-engineered.

1.1.3 Modern Cryptography Principles

- Unlike classical cryptography, modern cryptography is built on rigorous principles:
- **Formal Definitions:** Provides a precise, mathematical model of what security means.
- **Assumptions:** Clearly states the threat model and computational assumptions (e.g., "Factoring is hard").
- **Proof of Security:** Involves a formal mathematical proof that the scheme is secure under its definitions and assumptions.

1.2 Defining Encryption

1.2.1 Components

- An encryption scheme consists of three algorithms:
- **KeySetup() → k:** Generates a private key.
- **Enc(m, k) → c:** Encrypts a message m with key k to get ciphertext c.
- **Dec(c, k) → m:** Decrypts ciphertext c with key k to recover m.

- **Correctness:** A scheme must satisfy $\forall m \in M, k \in K, Dec(Enc(m,k),k) = m$.
- **Spaces:** We define K (Key space), M (Message space), and C (Ciphertext space). K should have a uniform distribution.

1.3 Perfect Secrecy

1.3.1 Formal Definition

- An encryption scheme achieves **perfect secrecy** if, for all messages $m \in M$ and all ciphertexts $c \in C$:
- $Pr[M = m|C = c] = Pr[M = m]$
- This means the probability of a message being m *after* seeing the ciphertext c is the same as the probability *before* seeing it.

1.3.2 Analysis of Shift Cipher

- The shift cipher is proven to **not** be perfectly secure.
- Using Bayes's theorem ($Pr[A|B] = Pr[B|A] \times \frac{Pr[A]}{Pr[B]}$) and the Law of Total Probability, we can show that $Pr[M = m|C = c]$ is not equal to $Pr[M = m]$. For example, if M can be 'one' or 'ten', observing the ciphertext 'rqh' makes the probability of $M =$'ten' zero, which is different from its initial probability.

2 Lecture 8: One-Time Pad

2.1 Definition and Properties

2.1.1 XOR Properties

- The One-Time Pad (OTP) relies on the bitwise XOR ($\oplus$) operation.
- **Commutative:** $A \oplus B = B \oplus A$
- **Associative:** $(A \oplus B) \oplus C = A \oplus (B \oplus C)$
- **Identity:** $A \oplus 0 = A$
- **Self-Inverse:** $A \oplus A = 0$

2.1.2 OTP Algorithm

- **KeyGen:** Generate a uniformly random n-bit key k.
- **Encryption:** $c = m \oplus k$
- **Decryption:** $m = c \oplus k$ (This works because $c \oplus k = (m \oplus k) \oplus k = m \oplus (k \oplus k) = m \oplus 0 = m$)

2.2 Security of One-Time Pad

2.2.1 Proof of Perfect Secrecy

- The One-Time Pad is a perfectly secure encryption scheme.
- A formal proof demonstrates that $Pr[M = m|C = c] = Pr[M = m]$.
- This is because for any given plaintext m and ciphertext c, there exists exactly one unique key k (where $k = m \oplus c$) that maps them. Since the key k is chosen uniformly at random, every possible plaintext is equally likely to be the "true" plaintext, regardless of the observed ciphertext.

2.3 Limitations of Perfect Secrecy

2.3.1 Key Size Requirement

- A major theorem states that for any perfectly secure encryption scheme, the size of the key space must be at least as large as the size of the message space: $|K| \geq |M|$.
- This is proven in two parts:
- 1. $|M| \leq |C|$ (from the correctness property; the mapping must be one-to-one).

- 2. $|C| \leq |K|$ (from the secrecy property; if $|K| < |C|$, some ciphertexts would be impossible for a given m, leaking information).
- **Implication:** The key must be at least as long as the message, which is highly impractical.

2.4 Key Re-use (The "One-Time" Rule)

2.4.1 The Two-Time-Pad Attack

- The "One-Time" in OTP is a strict requirement. Reusing the same key k for two different messages (m and m') is catastrophic.
- An attacker who intercepts both ciphertexts (c and c') can compute:
- $c \oplus c' = (m \oplus k) \oplus (m' \oplus k) = m \oplus m'$
- This leaks the XOR of the two plaintexts, which is often enough to recover both messages using statistical analysis (e.g., if the plaintexts are images or text).

2.5 Kerckhoffs's Principle

- This principle states that a cryptosystem should be secure even if everything about the system, except the key, is public knowledge.
- Relying on the secrecy of the algorithm ("security by obscurity") is a flawed model, as algorithms can be leaked or reverse-engineered (e.g., RC4).

3 Lecture 9: Integrity

3.1 Threat Model

- We now consider an active adversary, **Mallory**, who is on the communication channel.
- Mallory's goal is to alter the message m into a different message m' without being detected by the receiver (Bob).
- This threatens message **integrity**.

3.2 Message Authentication Codes (MAC)

- A MAC is a cryptographic primitive designed to ensure:
- **Integrity:** Any change to the message m will be detected.
- **Sender Authenticity:** Bob is assured that the message was sent by Alice (or someone else who possesses the shared secret key k).

3.3 Cryptographic Hash Functions

3.3.1 Properties

- A function H that maps an arbitrary-length input x to a fixed-size output (digest).
- **Collision Resistance:** It is computationally hard to find two different inputs x and x' such that $H(x) = H(x')$.
- **One-Way (Not Invertible):** Given a digest d, it is computationally hard to find an input x such that $H(x) = d$.

3.3.2 Applications

- **Password Hashing:** Store $H(password)$ instead of the plaintext password. When a user logs in, compute the hash of their input and compare it to the stored hash.
- **Software Integrity:** A developer publishes the hash of their software (e.g., vlc.exe). Users can download the file, compute its hash, and compare it to the published hash to ensure the file hasn't been modified (e.g., to include a virus).

3.4 Building MACs

3.4.1 Hash-based MAC (HMAC)

- A common way to build a MAC is by using a hash function combined with a secret key k.

- Alice sends the message m along with a $tag = H(m, k)$.
- Bob receives m and computes his own $tag' = H(m, k)$ using the shared key. If $tag = tag'$, the message is authentic and has not been altered.
- Mallory cannot forge a valid tag for a new message m' because she does not know k.

3.5 Perfect MACs

3.5.1 Definition

- A MAC is "perfect" (information-theoretically secure) if an adversary's probability of forging a valid tag for a new message is no better than random guessing.
- $Pr[\text{Adversary succeeds}] = 1/|T|$, where $|T|$ is the size of the tag space.
- This is achieved if:

$$P(tag(m, k) = t \cap tag(m', k) = t') = \frac{1}{|T|^2}$$

3.5.2 Key Space Requirement

- A key theorem for perfect MACs states that the size of the key space $|K|$ must be at least the square of the tag space size $|T|$.
- $|K| \geq |T|^2$
- For an n-bit tag, the key must be at least 2n bits.

4 Lecture 10: Public Key Cryptography (PKC)

4.1 Symmetric vs. Asymmetric Cryptography

- **Symmetric Key:** Uses the *same* key for encryption and decryption (e.g., One-Time Pad, AES).
- **Public Key (Asymmetric):** Uses a *pair* of keys: a public key (pk) for encryption and a private key (vk) for decryption.

4.1.1 Key Distribution Problem

- Symmetric key crypto has a major challenge: how to securely share the secret key in the first place?
- In a network of n users, symmetric keys require $n(n-1)/2$ total keys, each needing a secure channel for setup.
- PKC solves this: each user has one public key (which can be broadcast openly) and one private key. This requires only n key pairs.
- **Correctness:** $Dec(Enc(m, pk), vk) = m$
- **Secrecy:** Given pk and c, it is difficult to determine m.

4.2 RSA Algorithm

4.2.1 Key Generation

- 1. Choose two large, random primes, p and q.
- 2. Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$. ($\phi(n)$ is Euler's Totient function).
- 3. Choose a public exponent e (often a prime like 3 or 65537) such that $gcd(e, \phi(n)) = 1$.
- 4. Compute the private exponent d such that $d \cdot e \equiv 1 \pmod{\phi(n)}$. This is done using the Extended Euclidean Algorithm.
- **Public Key:** $pk = (n, e)$
- **Private Key:** $vk = d$ (The primes p and q can be discarded).

4.2.2 Encryption and Decryption

- **Encryption:** $c = m^e \pmod{n}$
- **Decryption:** $m = c^d \pmod{n}$

- **Why it works (Math):** $c^d \equiv (m^e)^d \equiv m^{ed} \pmod{n}$. Since $ed \equiv 1 \pmod{\phi(n)}$, we have $ed = k \cdot \phi(n) + 1$ for some integer k. So, $m^{ed} \equiv m^{k\cdot\phi(n)+1} \equiv (m^{\phi(n)})^k \cdot m \pmod{n}$. By Euler's Theorem, $m^{\phi(n)} \equiv 1 \pmod{n}$. Thus, $1^k \cdot m \equiv m \pmod{n}$.

4.2.3 Concrete Example

- Let $p = 5$, $q = 11$. Then $n = 55$ and $\phi(n) = (5-1)(11-1) = 40$.
- Choose $e = 3$ (since $gcd(3, 40) = 1$).
- Find d such that $3d \equiv 1 \pmod{40}$.
- Using Extended Euclidean Algorithm: $40 = 13 \cdot 3 + 1 \implies 1 = 40 - 13 \cdot 3$.
- So $d = -13$, which is $27 \pmod{40}$.
- **Public Key:** $(n = 55, e = 3)$. **Private Key:** $d = 27$.
- **Encrypt message** $m = 9$**:** $c = 9^3 \pmod{55} = 729 \pmod{55} = 14$.
- **Decrypt ciphertext** $c = 14$**:** $m = 14^{27} \pmod{55} = 9$.

4.2.4 Security (Factoring Problem)

- The security of RSA relies on the computational difficulty of **integer factorization**.
- It is easy to multiply p and q to get n, but it is extremely hard to find p and q given only n.
- An attacker who can factor n can compute $\phi(n)$ and thus derive the private key d.
- Note: RSA is *not* perfectly secure; a brute-force (unbounded) adversary can factor n. Its security is computational.

4.3 ElGamal Algorithm

4.3.1 Security (Discrete Logarithm Problem)

- ElGamal is a PKC scheme based on the **Discrete Logarithm Problem (DLP)**.
- DLP: Given a prime p, a generator g, and a value y, it is computationally hard to find the exponent x such that $y = g^x \pmod{p}$.

4.3.2 Encryption and Decryption (Probabilistic)

- **KeyGen:** $vk = x$. $pk = (p, g, y)$ where $y = g^x \pmod{p}$.
- **Encryption:** To encrypt m, choose a new random k $(1 \le k \le p-2)$.
- $c_1 = g^k \pmod{p}$

- $c_2 = m \cdot y^k \pmod{p}$
- The ciphertext is the pair (c_1, c_2). ElGamal is probabilistic because of the random k.
- **Decryption:**
- 1. Compute $s = (c_1)^x \pmod{p}$ (which equals $(g^k)^x = g^{kx}$)
- 2. Compute $s^{-1} \pmod{p}$
- 3. $m = c_2 \cdot s^{-1} \pmod{p}$ (This works because $c_2 \cdot (g^{kx})^{-1} = (m \cdot y^k) \cdot (g^{kx})^{-1} = (m \cdot (g^x)^k) \cdot (g^{kx})^{-1} = m$)

4.3.3 Comparison: RSA vs. ElGamal

	RSA	**ElGamal**
Basis	Factoring Problem	Discrete Log Problem
Nature	Deterministic	Probabilistic (due to random k)
Ciphertext	$c = m^e \pmod{n}$	$(c_1, c_2) = (g^k, m \cdot y^k) \pmod{p}$
Speed	Fast encryption (if e is small), slow decryption	Slower encryption/decryption
Quantum	Vulnerable	Vulnerable

4.4 Digital Signatures

4.4.1 Concept

- PKC enables digital signatures by ”reversing” the process.
- **Sign:** A user (Alice) uses her *private key vk* to create a signature on a message m.
- **Verify:** Anyone can use Alice's *public key pk* to verify that the signature is valid and was created by Alice.

4.4.2 RSA Signatures

- **Sign:** $sig = H(m)^d \pmod{n}$
- **Verify:** Check if $(sig)^e \pmod{n} == H(m)$.
- **Why Hash First?** We sign the hash $H(m)$ instead of m for two main reasons:
- 1. **Security:** Prevents forgery attacks. For example, if we signed raw messages, $sig(m_1) \cdot sig(m_2) = (m_1^d)(m_2^d) = (m_1 m_2)^d = sig(m_1 m_2)$. An attacker could forge a signature for $m_1 m_2$. Hashing breaks this: $H(m_1 m_2) \neq H(m_1) \cdot H(m_2)$.
- 2. **Efficiency:** It produces a fixed-size, short input for the RSA operation, regardless of how long the original message is.

5 Lecture 11: Secure Channel with Cryptography

5.1 The CIA Requirement

- The goal is to build a secure channel over an unsecure public network (like the internet) that provides the "CIA" guarantees:
- **Confidentiality** (via Encryption)
- **Integrity** (via MAC)
- **Authentication** (via Digital Signatures)

5.2 Key Exchange and Forward Secrecy

5.2.1 Session Keys

- We must use fresh, temporary **session keys** for each communication session, rather than re-using a long-term key.
- This provides **Forward Secrecy**: If an attacker compromises a long-term private key in the future, they *cannot* decrypt past communications that were encrypted with session keys (which have been destroyed).

5.3 Diffie-Hellman Key Exchange (DHKE)

5.3.1 The Algorithm

- A method for two parties (Alice and Bob) to establish a shared secret over an insecure channel, *without* ever sending the secret itself.
- 1. Alice and Bob agree on public parameters p (prime) and g (generator).
- 2. Alice chooses a private secret a, computes $A = g^a \pmod{p}$, and sends A to Bob.
- 3. Bob chooses a private secret b, computes $B = g^b \pmod{p}$, and sends B to Alice.
- 4. Alice computes $K = B^a \pmod{p} = (g^b)^a = g^{ab} \pmod{p}$.
- 5. Bob computes $K = A^b \pmod{p} = (g^a)^b = g^{ab} \pmod{p}$.
- Both now share the secret key K.

5.3.2 Security (CDH Problem)

- A passive eavesdropper (Eve) sees $g, p, A = g^a$, and $B = g^b$.
- To find $K = g^{ab}$, Eve would need to solve the **Computational Diffie-Hellman (CDH) problem**, which is believed to be computationally hard.

5.4 Man-in-the-Middle (MITM) Attack

- DHKE is vulnerable to an *active* adversary (Mallory).
- 1. Mallory intercepts Alice's $A = g^a$ and sends her own $C = g^c$ to Bob instead.
- 2. Mallory intercepts Bob's $B = g^b$ and sends $C = g^c$ to Alice instead.
- 3. Alice computes a shared key with Mallory: $K_1 = C^a = g^{ca}$.
- 4. Bob computes a shared key with Mallory: $K_2 = C^b = g^{cb}$.
- Mallory can now decrypt all messages from Alice, re-encrypt with K_2, and send to Bob (and vice-versa), reading and modifying everything.

5.5 Authenticated Key Exchange (AKE)

5.5.1 Station-to-Station (STS) Protocol

- To defeat the MITM attack, we must *authenticate* the key exchange.
- In the STS protocol, Alice and Bob use **digital signatures** to sign the values they send.
- Alice sends g^a and a signature on g^a.
- Bob verifies the signature with Alice's public key, confirming it came from her. He then sends g^b and his signature on g^b.
- Mallory cannot forge these signatures, so the MITM attack fails.

5.6 Application: TLS/SSL

5.6.1 Protocol Flow

- TLS (used in HTTPS) is a real-world example of a secure channel.
- **Step 0:** Your browser (Alice) gets the server's (Bob.com) public key via a *certificate*.
- **Step 1:** Your browser and the server perform an Authenticated Key Exchange (like STS) to securely establish shared session keys (e.g., k_1 for encryption, k_2 for MAC).
- **Step 2:** All subsequent communication is protected using the session keys.

5.6.2 Encrypt-then-MAC

- TLS protects messages by first encrypting them, then applying a MAC to the ciphertext.
- $E_{k1}(i||m)||mac_{k2}(E_{k1}(i||m))$

- The sequence number i (e.g., 1, 2, 3...) is included to prevent **replay attacks**, where Mallory re-sends an old, valid message.

6 Lecture 12: Blockchain

6.1 Trust and Decentralization

- Our economy is based on **trust**, often delegated to a trusted third party (e.g., banks, governments, servers).
- A **blockchain** is a technology that provides coordination and enables transactions in a virtual world *without* a single trusted party.

6.2 Core Components of a Blockchain

6.2.1 Linked Ledger

- A blockchain is a time-ordered ledger of all transactions.
- Transactions are grouped into **blocks**.
- Each block is "chained" to the previous one by including a **hash pointer** (the cryptographic hash) of the previous block's header.
- This chain makes the ledger **tamper-resistant**: changing any data in an old block would change its hash, which would break the link to the next block, and so on.

6.2.2 Transactions

- **Identity:** A user's identity is their **public key**, also known as their "address".
- **Transactions:** To transfer a token (e.g., Bitcoin), the owner (Alice) creates a transaction and proves ownership by signing the transaction with her **private key**.

6.3 Proof of Work (PoW)

6.3.1 The Hash Puzzle

- In a decentralized system, who gets to add the next block?
- **PoW** is a mechanism that forces "miners" to compete. To create a block, a miner must find a random number (a **nonce**) such that:
- $H(\text{nonce}||\text{prev\_hash}||\text{transactions}||...) < \text{Target}$
- The target is a very small number, so finding a valid nonce is extremely difficult and requires many, many hash computations (i.e., "work").

6.3.2 Properties

- **Difficult to Compute:** Requires immense computational power. The probability of a miner finding the next block is proportional to their share of the total hash power.

- **Parameterizable Cost:** The network automatically adjusts the "Target" to ensure the average time between blocks remains constant (e.g., 10 minutes for Bitcoin).
- **Trivial to Verify:** Once a miner finds a valid nonce, it's very easy and fast for all other nodes to verify it (just one hash computation).
- This defends against **Sybil attacks**, where an attacker creates many fake identities. In PoW, identities don't matter; only computational power does.

6.4 Consensus

6.4.1 The Longest Valid Chain Rule

- What happens if two miners find a block at the same time? This creates a "fork" (two competing chains).
- The **consensus rule** in PoW blockchains is simple: all nodes accept **the longest valid chain** as the single source of truth.
- Miners are incentivized to build on the longest chain they see, so forks quickly resolve as one chain outgrows the other.

6.4.2 Incentives

- Miners are incentivized to participate honestly.
- **Block Reward:** The miner who finds a valid block is rewarded with newly created coins (e.g., Bitcoin).
- **Transaction Fees:** Miners also collect fees from the transactions they include in their block.
- These rewards are only valuable if the block becomes part of the longest chain, so miners are strongly motivated to follow the rules. This creates an implicit consensus.

6.5 Security and Attacks

- **Modify Transaction Contents:** This is prevented by cryptography. A malicious miner does not know Alice's private key, so they cannot modify her transaction and re-sign it. Any change would invalidate the signature.
- **Deny a Transaction (Censorship):** A malicious miner can refuse to include a transaction in their block. However, as long as *other* honest miners include it, the transaction will eventually get processed. This only becomes a problem in a **51% Attack**.

- **Double-Spending Attack:** An attacker tries to spend the same coin twice on two different forks. This is defeated by waiting for **confirmations** (i.e., waiting for more blocks to be added on top of the transaction). After 6 confirmations, reversing the transaction (by building a longer, alternate chain) becomes computationally infeasible.
- **51% Attack:** If a single entity controls more than 50% of the network's hash power, they can outpace all other miners, create the longest chain at will, and potentially reverse transactions (double-spend) or censor others.

6.6 Private Key Management (Wallets)

- In blockchain, your private key *is* your ownership. If you lose it, your coins are gone forever.
- **Mnemonic Phrases:** A human-readable backup (e.g., 12 or 24 words) that can be used to regenerate all private keys from a master "seed".
- **Hardware Wallets:** A physical device that stores private keys offline. Transactions are signed *inside* the device, so the key never leaves it.
- **Secret Sharing:** Splits a key into N pieces, such that any K pieces are required to reconstruct it. For example, by defining a polynomial $f(x)$ of degree $K-1$, where the secret is $f(0)$, and the N pieces are $f(1), f(2), ..., f(N)$.
- **Multisig:** An address that requires M-of-N (e.g., 2-of-3) *different* private keys to sign a transaction, providing distributed control.