CS 5594: Blockchain Technologies

Cryptographic Primitives

Peng Gao
Assistant Professor

Spring 2022

Part of materials are derived from Prof. Thang Hoang’s SP 2021 course
Outline

• What Is Cryptography? Why Cryptography?
• Cryptographic Hash Functions
• Hash-Based Primitives
  – Digital Signature
  – HMAC
  – Hash Chain, Hash List
  – Merkle Tree
• Public Key Cryptography
  – RSA
  – Digital Signature Algorithm (DSA)
  – Elliptic Curve Digital Signature Algorithm (ECDSA)
Outline

• What Is Cryptography? Why Cryptography?
• Cryptographic Hash Functions
• Hash-Based Primitives
  – Digital Signature
  – HMAC
  – Hash Chain, Hash List
  – Merkle Tree
• Public Key Cryptography
  – RSA
  – Digital Signature Algorithm (DSA)
  – Elliptic Curve Digital Signature Algorithm (ECDSA)
What Is Cryptography

• **Cryptography** is the study of techniques for secure communications in the presence of adversaries
  – Based on mathematical theory

• Security is not the same as cryptography

*Introduction to modern Cryptography (3rd edition)*
– Jonathan Katz and Yehuda Lindell
What Is Cryptography

- **Cryptographic algorithms** are designed around **computational hardness assumptions**
  - Assuming adversary is **computationally bounded**
    - Problem cannot be solved efficiently (in polynomial time)
    - E.g., integer factorization: given integer \( n \), solve for \( p, q: p \cdot q = n \)
      - Hard when \( n \) grows large
    - E.g., discrete log problem: given integers \( n, g, \) and prime \( p \), solve for \( x: n = g^x \mod p \)
      - Hard when \( p \) grows large
  - Theoretically breakable
  - Hard to break in practice with current computing power
  - Need to reevaluate with better theory and faster computing
- **Examples**
  - Hash functions: MD5, SHA-1, SHA-256
  - Symmetric encryption: DES, AES
  - Asymmetric encryption: RSA, ECDSA
What Is Cryptography

• **Information-theoretically secure** schemes
  – No computational assumptions about the adversary
  – Theoretically unbreakable even with *unlimited* computing power
  – Hard to use in practice

• Example information-theoretic encryption scheme: **one-time pad**
Information-Theoretic Encryption: One-Time Pad

Terminology
- $M$: plaintext
- $K$: encryption key
- $C$: ciphertext

Message: $M = \{0,1\}^n$
Key: $K = \{0,1\}^n$
$Enc(M, K) = M \oplus K$

Adversary: gets no information from ciphertext

Using a new random secret key (as long as the plaintext) for each encryption
Problems With One-Time Pad

• **Conditions** for one-time pad to be theoretically unbreakable
  – The key is at least as long as the plaintext
  – The key is random (independent of the plaintext)
  – The key (in whole or in part) is used only once
  – The key is kept completely secret

• **Problems** in **practical** adoption
  – Costly to generate a long, random key every time
  – The sender needs to send the long key securely to the receiver ⇒ If such a method exists, why not just send the plaintext message?
The Role of Cryptography in Information Security

• Can be used to achieve several goals of information security
  – **Confidentiality:** keeping sensitive information private
    ▪ Encryption: plaintext $\rightarrow$ ciphertext
    ▪ Decryption: ciphertext $\rightarrow$ plaintext
  – **Integrity:** keeping information unmodified
    ▪ Cryptographic hash functions
  – **Authentication:** verifying someone/something is who/what it is declared to be
    ▪ Digital signatures, digital certificates
    ▪ Beyond cryptography
      • “Something you are”: biometric (fingerprint, Face ID)
      • “Something you have”: one-time token
      • “Something you know”: PIN, password
      • Multifactor authentication (MFA): password + cell phone
  – **Non-repudiation:** cannot deny having performed a particular action
    ▪ Digital signatures

• (Not limited to encryption)

• **Cryptographic primitives:** low-level cryptographic algorithms used to build cryptographic protocols for security systems (e.g., SHA-256, AES, RSA, PRG, digital signatures)
The CIA Triad for Information Security

- An information security model used to guide an organization’s security procedures and policies
  - **Confidentiality**: keeping an organization’s data private or secret; protecting the data from unauthorized access
    - Cases of compromise: electronic eavesdropping, data exfiltration
  - **Integrity**: keeping an organization’s data correct and reliable; protecting the data from unauthorized changes
    - Cases of compromise: modifying configuration files, changing system logs to evade detection
  - **Availability**: authorized users can have timely, reliable access to resources they need; networks, systems, and applications are up and running
    - Cases of compromise: denial-of-service (DoS) attack, system failure, natural disasters

- Using the CIA Triad to analyze cybersecurity
- Cryptography does **NOTHING** to ensure availability
Why Cryptography for This Course

• Bitcoin is a cryptocurrency
• Crypto is a mandatory building block in Bitcoin/blockchain
  – Cryptographic hash functions and hash-based primitives
  – Public key cryptography

Remark: Blockchain is based on distributed system and cryptography
Outline

• What Is Cryptography? Why Cryptography?
• Cryptographic Hash Functions
• Hash-Based Primitives
  – Digital Signature
  – HMAC
  – Hash Chain, Hash List
  – Merkle Tree
• Public Key Cryptography
  – RSA
  – Digital Signature Algorithm (DSA)
  – Elliptic Curve Digital Signature Algorithm (ECDSA)
Hash Function

• Hash function: a function $H$ with two basic properties
  – **Compression**: $H$ maps an input $x$ of arbitrary length to an output $y = H(x)$ of a fixed length
    ▪ The process is called hashing the data
      • Hash functions are sometimes called compression functions, one-way functions
    ▪ Outputs are called hash values or message digest
      • n-bit output: $y = |H(x)| \ll |x|$, $|y| = \{160, 256, 384, 512\}$ (preferred 256 bits)
  – **Ease of computation**: given $H$ and $x$, it’s easy to compute $y = H(x)$
Three Important Properties of Hash Functions

• Three additional important properties for a hash function to be an effective cryptographic tool
  
  - **Preimage resistance**
    - Computationally hard to reverse the hash function
      - For any \( y \) (in the range of \( H \)) for which a corresponding input is unknown, computationally infeasible to find any input \( x \) s.t. \( H(x) = y \)
    - Protecting against an attacker who only has a hash value and is trying to find the input (preimage attacks)
  
  - **2\textsuperscript{nd} preimage resistance**
    - Given an input and its hash, computationally hard to find a different input with the same hash
      - Given \( x \) and \( y = H(x) \), computationally infeasible to find \( x' \neq x \) s.t. \( H(x) = H(x') = y \)
    - Protecting against an attacker who has an input value and its hash, and wants to substitute a different input value as legitimate (e.g., checksum on files)
  
  - **Collision resistance**
    - Computationally hard to fine two different inputs that result in the same hash
      - Computationally infeasible to find \( x' \neq x \) s.t. \( H(x) = H(x') \)
    - Note that it is impossible for a hash function to **NOT** have collisions (due to compression); the property only says it is **HARD** to find
    - Protecting against collision attacks (finding two inputs producing the same hash value)
Relationships Between Properties

• Does collision resistance imply $2^{\text{nd}}$ preimage resistance?
  – Yes
  – **Proof by Contraposition:**
    ▪ Assume the $2^{\text{nd}}$ preimage resistance property does not hold. Repeatedly pick a random $a$, perform the attack that exhibits a $b$ with $a \neq b$ and $H(a) = H(b)$, until it succeeds. This requires a feasible amount of work by our assumption. Once exhibited, the pair $(a, b)$ proves that the collision resistance property does not hold.
  – $2^{\text{nd}}$ preimage resistance is a weaker version of collision resistance

• Does $2^{\text{nd}}$ preimage resistance imply collision resistance?
  – No

• Does preimage resistance imply $2^{\text{nd}}$ preimage resistance?
  – No
Relationships Between Properties

- Does 2nd preimage resistance imply preimage resistance?  
  - No

- Does collision resistance imply preimage resistance?  
  - No
  - **Proof by Construction:**
    - Let \( G \) be a hash function which is collision resistant and maps arbitrary-length inputs to \( n \)-bit outputs. Consider function \( H \):
      - \( H(x) = 1 \parallel x \), if \( x \) has bit length \( n \)
      - \( 0 \parallel G(x) \), o.w.
    - Is \( H \) collision resistant? (yes)
    - Is \( H \) preimage resistant? (no)

- Does preimage resistance imply collision resistance?  
  - No

- Different applications need different properties
Building Collision-Resistant Hash Functions

• Merkle-Damgard (MD) Construction
  – Starting from a collision-resistant one-way compression function $H$

- Iterating it
  ▪ Length padding: padding the input message to make its length a multiple of a fixed number (e.g., 512)
  ▪ IV: initialization vector (a fixed value)
  ▪ Breaking input into blocks and processing each block with the compression function
    • Hash value of the first message block becomes an input to the second hash operation
    • Avalanche effect: a slight change of input results (e.g., 1 bit flip) in a significant change of output
  – Property: If the compression function is collision resistant and an appropriate padding scheme (MD-compliant padding) is used, the constructed hash function is also collision resistant
Popular Hash Functions

- **MD5** (Message Digest)
  - 128-bit output
    - e.g., MD5 checksum for transferred files to provide integrity
  - Designed by Ron Rivest, 1991
  - Previous versions (MD2, MD4) have serious weaknesses
  - Broken:
    - Xiaoyun Wang et. al. found collision in one hour using IBM p690 cluster, 2004
    - Klima found collision within one minute on a notebook computer using tunneling technique, 2006
  - Too weak to be used for critical applications. No longer recommended for use

- **SHA-1** (Secure Hash Function)
  - 160-bit output
    - e.g., used for SSL (Secure Socket Layer) certificates
  - Designed by NSA, adopted by NIST, 1993
  - Broken: Xiaoyun Wang et. al. found attack on SHA-1, 2005 (CRYPTO’05)
    - Collision found in $2^{69}$ hash operations, much less than brute-force of $2^{80}$ operations

- Recommended
  - SHA-2 family: **SHA-256** (256-bit output), SHA-384, SHA-512
  - BLAKE-256/512 (good for embedded devices)
Applications of Hash Functions

• Password storage for websites
  – Server logon processes store “hash(password)” in file (list of <user_id, $H(P)$> entries)
  – New user logon:

  ![Diagram](Image)

Collision resistance $\Rightarrow$ cannot guess a random password that works

Preimage resistance $\Rightarrow$ cannot derive the password from the leaked hash
Applications of Hash Functions

- Data file integrity check using checksum
  - Detect any changes made to a data file
Birthday Paradox

• Why do we need 160 (e.g., SHA-1) or 256 (e.g., SHA-256) bits in the output of a hash function?
  – If it is too long: unnecessary overhead
  – If it is too short: loss of strong collision resistance property => vulnerable to **Birthday attack**

• **Birthday paradox**
  – The probability of a shared birthday in a group of 23 people exceeds 50%
  – The probability of a shared birthday in a group of 30 people is around 70%
Birthday Paradox

A classroom of $k$ students.

Probability of at least two students having the same birthday ($= 1 - Pr(\text{all students have different birthdays})$):

- $k = 1$: \[1 - \frac{365}{365}\]
- $k = 2$: \[1 - \frac{364}{365}\]
- $k = 3$: \[1 - \left(\frac{364}{365} \times \frac{363}{365}\right)\]

Thus, for $k$ students:

\[
1 - \left(\frac{364}{365} \times \frac{363}{365} \cdots \frac{365 - k - 1}{365}\right)
= 1 - \frac{365!}{(365 - k!) \cdot (365^k)}
\]
“Birthday of a person” can be thought of as a function that maps from an arbitrary person identifier to a fixed degree output from 1-365

- For 25 (different) people, around 50% probability to find a collision
- For 30 people, around 70% probability to find a collision
- For 366 people, 100% probability to find a collision

Generally, for a function \( H \) with \( n \) possible outputs, for \( k \) different inputs:

\[
\Pr(\text{no collision}) = \left(1 - \frac{1}{n}\right) \times \left(1 - \frac{2}{n}\right) \times \cdots \times \left(1 - \frac{k - 1}{n}\right)
\]

When \( x \) is small \( 1 - x \approx e^{-x} \Rightarrow \left(1 - \frac{1}{n}\right) \approx e^{-\frac{1}{n}}, \ldots \)

\[
\Pr(\text{no collision}) \approx e^{-1/n} \times e^{-2/n} \times \cdots \times e^{-\frac{k-1}{n}}
\]
\[
\approx e^{-\frac{k(k-1)}{2n}} \approx e^{-k^2/2n}
\]

\[
\Pr(\text{at least one collision}) = 1 - e^{-k^2/2n}
\]

Suppose the probability for finding a collision is at least \( p \). Then:

\[
k \approx \sqrt{2n \times \ln\left(\frac{1}{1 - p}\right)}
\]

In general, for a function with \( n \) possible outputs, on average \( \sqrt{n} \) inputs are required to find a collision
Birthday Attack

• A cryptographic attack that exploits the birthday paradox to find collisions for a hash function

• For a hash function $H$ of output length $m$ bits
  
  $\sqrt{2n \times \ln\left(\frac{1}{1-p}\right)} \approx 1.174\sqrt{n} \approx \sqrt{2^m} = 2^{m/2}$

  Thus, if we hash $2^{m/2}$ random inputs, it is likely that two messages will have the same hash outputs

  Conclusion: choose at least $m \geq 160$, preferably $m = 256$
Birthday Attack Example

- Example: $H(x) = \text{first_byte}(\text{SHA256}(x))$
  - Collision: $H(\text{"GOOD"}) = H(\text{"EVIL"}) = 1a$

Salted Hashing

- Deterministic hashing is not sufficient for storing passwords
  - Vulnerable to rainbow table attack
    - A type of pre-computation attack
    - Rainbow table: pre-computed tables/databases of hashes and their plaintext passwords
    - Hashes can be searched for and immediately reversed into plaintexts
    - Using less processing time and more space than brute-force attack
      - Basic brute-force attack: exhaustive search for every possible combination for passwords
      - Dictionary attack: a type of brute-force attack that uses a predefined list of passwords that would have a higher probability of success
      - Traditional brute force attacks store no pre-computed data and compute each hash at run time ⇒ using minimal space but taking a long time (space-time tradeoff)

- Salted hashing
  - **Salt**: a random unique token passed to the hash function along with the password plaintext to generate a unique hashed password
  - Goal of salting: making it infeasible to construct pre-computed databases to crack password hashes
    - Infeasible to generate rainbow tables for every possible salt
  - The salt should be unique per password
    - For global salt (same salt for all passwords), pre-computed databases can still be used. --- though they have to be computed for specifically for the application’s salt
  - The salt can be stored right next to the salted and hashed password in the database
    - Need both the salt and the hashed password to validate a password. Thus, they should be “close” in architectural sense
    - Salts are NOT assumed to be secret
  - For extra security, can use a second piece of salt (called “**pepper**”), which is the same for all users and stored outside the database (e.g., environment variable, configuration file, code)
Keyed Hashing

• Collisions are impossible to eliminate completely

• Keyed hash functions
  – An algorithm that uses a cryptographic key and a cryptographic hash function to produce a message authentication code that is keyed and hashed
  – Also known as hash-based message authentication code, or HMAC
    ▪ Used to detect when an attacker has tampered with a message (cryptographic checksums)

• Comparison to salted hashing
  – The salt is usually stored along with the hashed password and is not assumed to be secret, but the key is
  – The salts are supposed to vary but the key is shared between instances
Is Encryption a Good Hash Function?

• **Block-based encryption** as hash
  – Encryption block size may be too short (e.g., DES has 64-bit output)
    ▪ Birthday attacks
Outline

• What Is Cryptography? Why Cryptography?
• Cryptographic Hash Functions
• Hash-Based Primitives
  – Digital Signature
  – HMAC
  – Hash Chain, Hash List
  – Merkle Tree
• Public Key Cryptography
  – RSA
  – Digital Signature Algorithm (DSA)
  – Elliptic Curve Digital Signature Algorithm (ECDSA)
Hash-Based Primitives

• Authentication
  – Digital signature
  – Hash-based message authentication code (HMAC)
  – Authenticated data structures
    ▪ Hash chain
    ▪ Merkle tree

• Proof of work (PoW)
Digital Signature

• A cryptographic mechanism used to verify the **authenticity** and **integrity** of digital data
  – A code that is attached to a message
• Cryptographic hash functions + public key cryptography (PKC)
• **Hash-then-sign** paradigm
  – First shorten arbitrary-long message using hashing, \( d = H(M) \) --- hash function
  – Then sign the digest using the signer’s private key, \( s = \text{Sign}(d) \) --- PKC
• The recipient of the message can **verify** if the signature is valid by using the signer’s public key (“public” means everyone can access)

**Signature generation**
- Message \( m \)
  - Hash
  - \( H(m) \)
  - Sign
  - Signature \( s \)

**Signature verification**
- Bob’s public key \( pk \)
  - Message \( m \)
    - Hash
    - \( H(m) \)
    - Verify
    - Valid/invalid
- Bob’s private key \( sk \)
Digital Signature: Important Notes

• Different message have different digital signatures, though a handwritten signature can be used across messages

• Both public and private keys are generated by the sender of the message, but only the public key is shared with the receiver

• Private key needs to be kept in secret by the sender. Otherwise, anyone who gets the key can pretend to be the sender
  – E.g., attacker can use Alice’s private key to sign a transaction to move or spend her Bitcoins

• Hashing is not a must for producing a digital signature
  – Blockchains prefer hashing as the fixed-length digests facilitate the whole process

• Digital signature is not the same as encryption
  – Encryption:
    ▪ Public key to write message and private key to read it
    ▪ Provides confidentiality
  – Signing:
    ▪ Private key to write message’s signature and public key to verify it
    ▪ Provides message authentication and non-repudiation

• Digital signature provides:
  – **Data integrity**: any modification will produce a completely different signature
  – **Authenticity**: can use public key to confirm the signature was created by the corresponding sender
  – **Non-repudiation**: the sender cannot deny having signed it
Digital Signature: Security Aspects

• Relying on \textbf{collision resistance} property
  – If $H(M) = H(M')$, then $s$ is a signature for both $M$ and $M'$

• Attacks on MD5, SHA-1 threaten current signatures
  – MD5 attacks can be used to get bad CA certificate [Stevens et al. 2009]

• Well-worth the attacker’s effort
  – One collision $\Rightarrow$ forgery for any signer

• Improvement: adding a random token
  – To sign $M$, choose fresh random salt $s$, return $Sign(s, H(s, M))$
Outline

• What Is Cryptography? Why Cryptography?
• Cryptographic Hash Functions
• Hash-Based Primitives
  – Digital Signature
  – HMAC
  – Hash Chain, Hash List
  – Merkle Tree
• Public Key Cryptography
  – RSA
  – Digital Signature Algorithm (DSA)
  – Elliptic Curve Digital Signature Algorithm (ECDSA)
Hash-Based Message Authentication Code (HMAC)

- Using keyed hash functions
  - E.g., using encryption with key and output last $b$ bytes
  - Keyless hash functions: MD5, SHA-1
- Providing integrity and authenticity
- Simple key-prepend construction (MAC)
  - Procedure:
    - Sender/recipient shares secret key $K$
    - For message $M$, MAC: $\text{tag} \leftarrow H(K \parallel M)$
  - If the adversary changes $M$ to $M'$, he/she won’t know how to change the MAC
  - However, susceptible to length extension attacks
    - Pre-condition:
      - MD construction-based hash (e.g., MD5, SHA-1) is used with construction $H(K \parallel M)$
      - Message $M$ and the length of secret $k$ is known
    - Consequence: the attacker can include extra information at the end of $M$ and produce a valid hash without knowing the content of $k$
- Improved construction (HMAC)
  - Two passes of hash computation: for message $M$, tag $\leftarrow H(K \parallel H(K \parallel M))$
  - About as fast as key-prepend for a MD hash
  - Immunity against length extension attacks
- MAC/HMAC does not encrypt the message
HMAC Construction

\[ \text{HMAC}(K, M) = H \left( (K' \oplus \text{opad}) \parallel H((K' \oplus \text{ipad}) \parallel M) \right) \]

\[ K' = \begin{cases} H(K) & \text{if } K \text{ is larger than block size} \\ K & \text{Otherwise} \end{cases} \]

Two “magic numbers” --- related to the security proof

- \text{opad} is the block-sized outer padding, 0x5c5c5c…5c
- \text{ipad} is the block-sized outer padding, 0x363636…36
Outline

• What Is Cryptography? Why Cryptography?
• Cryptographic Hash Functions
  • Hash-Based Primitives
    – Digital Signature
    – HMAC
    – Hash Chain, Hash List
    – Merkle Tree
  • Public Key Cryptography
    – RSA
    – Digital Signature Algorithm (DSA)
    – Elliptic Curve Digital Signature Algorithm (ECDSA)
Hash Chain

• **Definition:** a successive application of a cryptographic hash function $H$ to $x$

  - $H^5(x) = H \left( H \left( H \left( H(H(x)) \right) \right) \right)$ has length 5

• **Key property:** suppose $i < j < k$, given $H^j(x)$, easy to compute $H^k(x)$, but computationally infeasible to compute $H^i(x)$

• **Applications**
  - S/Key one-time password protocol
  - Authenticated data streams
One-Time Password

• **Idea:** generate a long list of passwords, and use each **only one time**

• Attacker gains little/no advantage by eavesdropping on password transmission, or cracking one password

• Disadvantages:
  – Storage overhead
  – Need to remember lots of passwords

• **Alternative:** the **S/Key protocol**
  – Uses one-way hash chain
**S/Key One-Time Password Protocol**

- **Password generation**
  - A secret key $W$ is either generated by the server or provided by the user
  - The server applies $H$ to $W$ for $n$ times: $H(W), H(H(W)), \ldots, H^n(W)$
  - The secret $W$ is discarded
  - The user is provided with $n$ passwords in reverse order: $H^n(W), H^{n-1}(W), \ldots, H(H(W)), H(W)$
  - The server only stores $H^n(W)$ and discards the rest passwords

S/Key One-Time Password Protocol

• **Password authentication**
  - The user provides 
  \[ pwd = H^{n-i}(W) \quad (i = 1, 2, ..., n-1) \]
  - The server computes \( H(pwd) \), 
  and compares it with the one 
  stored on the server (i.e., \( H^{n-i+1}(W) \))
  - If the results match, the 
  authentication is successful. The 
  server then stores the provided 
  \( pwd \) for future use

• **Limitations:**
  - Limited number of passwords; 
  need to periodically regenerate a 
  new chain
  - Does not authenticate the server

[Diagram of S/KEY authentication process]

Chained Hash

- More general construction than one-way hash chains
- Useful for authenticating a sequence of data values $D_0, D_1, ..., D_n$
- $H^*$ authenticates the entire chain
Hash List

- **Definition**: a list of hashes of data blocks in a file
  - A subtree of a Merkle hash tree
  - Top/root hash: *commitment* of the entire hash list

- **Protecting data integrity**
  - Verification is complete after checking the root hash
  - Better than a simple hash of the entire file: only need to redownload the damaged blocks

\[ m_0 = H(k_0) \]
\[ m_0 \ldots m_7 = H(m_0, ..., m_7) \]
Outline

• What Is Cryptography? Why Cryptography?
• Cryptographic Hash Functions
  • Hash-Based Primitives
    – Digital Signature
    – HMAC
    – Hash Chain, Hash List
    – Merkle Tree
• Public Key Cryptography
  – RSA
  – Digital Signature Algorithm (DSA)
  – Elliptic Curve Digital Signature Algorithm (ECDSA)
Merkle Hash Tree

- A **binary tree** over data values
  - Every leaf is labelled with the **cryptographic hash** of a data block
  - Every inner node is labelled with the cryptographic hash of the labels of its child nodes
  - Top/root hash: **commitment** of the entire Merkle tree

- Generalization of hash chain and hash list
  - Protecting data integrity
Merkle Hash Tree

- Allows **efficient** and secure **verification** of the contents of a large data structure
  - Verification is complete after checking the root hash
  - $O(\log(n))$ to verify if a leaf node is part of a tree
  - No need to download the entire tree to verify a data block (need root node + some inner nodes)

- Hashing at the leaf level is **mandatory** to prevent unnecessary disclosure of data values

**Example:**
To authenticate $k_2$, send $(k_2, m_3, m_{01}, m_{47})$
Check $m_{07} = H\left(\left(H\left(m_{01} \parallel H\left(H\left(k_2\right) \parallel m_3\right)\right) \parallel m_{47}\right)\right)$
Merkle Hash Tree

- Authentication of the root is necessary to use the tree
  - Typically done through a digital signature or pre-distribution
- Merkle tree operations: updating, insertion, deletion
Merkle Hash Tree in Blockchain

Hash chain of blocks

Hash tree (Merkle tree) of transactions in each block
Outline

• What Is Cryptography? Why Cryptography?
• Cryptographic Hash Functions
• Hash-Based Primitives
  – Digital Signature
  – HMAC
  – Hash Chain, Hash List
  – Merkle Tree
• Public Key Cryptography
  – RSA
  – Digital Signature Algorithm (DSA)
  – Elliptic Curve Digital Signature Algorithm (ECDSA)
Public Key (Asymmetric) Cryptography

- Cryptographic operations use **different** keys
- Known as **asymmetric key cryptography**, **public key cryptography**
- Asymmetric encryption:
  - Digital signatures
  - Key management
Recall: Information-Theoretic Encryption: One-Time Pad

Terminology
- $M$: plaintext
- $K$: encryption key
- $C$: ciphertext

Message: $M = \{0,1\}^n$
Key: $K = \{0,1\}^n$
$Enc(M, K) = M \oplus K$

Internet ISPs

Adversary: gets no information from ciphertext

Secret key distribution

Key: $K = \{0,1\}^n$
$Dec(C, K) = C \oplus K = M$

Using a new random secret key (as long as the plaintext) for each encryption
Computational Encryption: Asymmetric/Public Key Encryption

• Two keys
  – Private key known only to individual
  – Public key available to anyone
  – Idea for confidentiality:
    ▪ Encrypt using public key, decrypt using private key

Message: $M = \{0,1\}^n$
Enc($M$, Public$_K$) = $C$

Keys:
• Public$_K$
• Private$_K$
Dec($C$, Private$_K$) = $M$
Computational Encryption: Asymmetric/Public Key Encryption

- Two keys
  - Private key known only to individual
  - Public key available to anyone
  - Idea for authentication/integrity (digital signature):
    - Encrypt using private key, decrypt using public key

\[ \text{Dec}(C, \text{Public}_K) = M \]

Keys:
- \( \text{Public}_K \)
- \( \text{Private}_K \)

Message: \( M = \{0,1\}^n \)

Encrypted with the private key, decrypted with the public key.
Computational Encryption: Asymmetric/Public Key Encryption

• **Requirements:**
  - It must be *computationally easy* to encrypt or decrypt a message given the appropriate key
  - It must be *computationally infeasible* to derive the private key from the public key
  - It must be *computationally infeasible* to determine the private key from a ciphertext

• PKC relies on some known **mathematical hard problems**
  - Large integer factorization (e.g., RSA)
    ▪ Given integer \( n \), solve for \( p, q \): \( p \cdot q = n \)
      • Hard when \( n \) grows large
  - Discrete logarithmic (e.g., DSA)
    ▪ Given integers \( n, g \), and prime \( p \), solve for \( x \): \( n = g^x \mod p \)
      • Hard when \( p \) grows large
  - Elliptic curve discrete logarithmic (e.g., ECDSA)
PKC Security Arguments

• Confidentiality
  – Only the owner of the private key knows it, so text encrypted with public key cannot be read by anyone except the owner of the private key

• Authentication
  – Only the owner of the private key knows it, so text encrypted with private key must have been generated by the owner

• Integrity
  – Encrypted letters cannot be changed undetectably without knowing private key

• Non-repudiation
  – Message encrypted with private key came from someone who knew it
Public key cryptosystem harnesses certain algebraic properties in finite field

- **Closure under addition**: 
  \[ a + b = c \in G \]

- **Associativity of addition**: 
  \[ a + (b + c) = (a + b) + c \]

- **Additive identity**: 
  \[ \exists e \text{ s.t. } a + e = e + a = a \]

- **Additive inverse**: 
  \[ \exists b \text{ s.t. } a + b = e \]

- **Commutativity of addition**: 
  \[ a + b = b + a \]

- **Closure under multiplication**: 
  \[ a \times b = c \in G \]

- **Associativity of multiplication**: 
  \[ a \times (b \times c) = (a \times b) \times c \]

- **Distributive laws**: 
  \[ a \times (b + c) = a \times b + a \times c \]

- **Commutativity of multiplication**: 
  \[ a \times b = b \times a \]

- **Multiplicative identity**: 
  \[ \exists e \text{ s.t. } a \times e = e \times a \]

- **No zero divisors**: 
  \[ a \times b = 0 \Rightarrow a = 0 \lor b = 0 \]

- **Multiplicative inverse**: 
  \[ \exists b \text{ s.t. } a \times b = e \]
Outline

• What Is Cryptography? Why Cryptography?
• Cryptographic Hash Functions
• Hash-Based Primitives
  – Digital Signature
  – HMAC
  – Hash Chain, Hash List
  – Merkle Tree
• Public Key Cryptography
  – RSA
  – Digital Signature Algorithm (DSA)
  – Elliptic Curve Digital Signature Algorithm (ECDSA)
Public Key Primitive: RSA

• Rivest-Shamir-Adleman
• Most popular public key method
  – Provide both public key encryption and digital signature
• Based on large integer factorization problem
  – Given integer $n$, solve for $p, q$: $p \cdot q = n$
  – Hard to factorize $n$ in polynomial time
• Variable key length (2048 bits or greater)
• Variable plaintext block size
  – Plaintext block size must be smaller than key size
  – Ciphertext block size is same as key size
RSA Algorithm

• **Euler’s totient function** $\phi(n)$
  
  – Number of positive integers less than $n$ and relatively prime to $n$
    
    ▪ **Relatively prime** means having no factors in common with $n$ 

• If $m$ and $n$ are relatively prime, then $\phi(mn) = \phi(m)\phi(n)$

• **Example:** $\phi(10) = 4$
  
  – 1, 3, 7, 9 are relatively prime to 10

• **Example:** $\phi(21) = 12$
  
  – 1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20 are relatively prime to 21
RSA Algorithm

- **Computing public/private keys:**
  - Find (using Miller-Rabin primality test) large primes $p$ and $q$
  - Let $n = p \cdot q$, then $\phi(n) = (p - 1)(q - 1)$
    - Do not disclose $p$ and $q$
  - Choose $e < n$ such that $e$ is relatively prime to $\phi(n)$
  - Compute $d$ such that $ed = 1 \mod \phi(n)$
    - $d$ is multiplicative inverse of $e$ under modulo $\phi(n)$

- Public key: $(e, n)$
- Private key: $d$
RSA Algorithm

• Let RSA public key \((e, n)\) and RSA private key \(d\)

• Given a plaintext message \(m < n\)

• **Encryption:**
  – Encryption: \(c \leftarrow m^e \pmod{n}\)
  – Decryption: \(m \leftarrow c^d \pmod{n}\)

• **Digital signature:**
  – Signing: \(s \leftarrow m^d \pmod{n}\)
  – Verification: \(m \leftarrow s^e \pmod{n}\)

• What if \(m > n\)?
  – Remark: Hash-then-sign paradigm
  – Hashing: \(t \leftarrow \text{Hash}(m), |t| = 160\) bits \(\Rightarrow t < n\)
  – Signing: \(s \leftarrow t^d \pmod{n}\)
RSA Example: Confidentiality

• Take $p = 7$, $q = 11$, so $n = 77$ and $\phi(n) = 60$

• Alice chooses $e = 17$, making $d = 53$
  – Verify: $ed = 901 = 1 \mod 60$

• Bob wants to send Alice secret message HELLO (07 04 11 11 14):
  – $07^{17} \mod 77 = 28$
  – $04^{17} \mod 77 = 16$
  – $11^{17} \mod 77 = 44$
  – $11^{17} \mod 77 = 44$
  – $14^{17} \mod 77 = 42$

• Bob sends 28 16 44 44 42
RSA Example: Confidentiality

• Alice receives 28 16 44 44 42

• Alice uses private key, \( d = 53 \), to decrypt message:
  \[ 28^{53} \mod 77 = 07 \]
  \[ 16^{53} \mod 77 = 04 \]
  \[ 44^{53} \mod 77 = 11 \]
  \[ 44^{53} \mod 77 = 11 \]
  \[ 42^{53} \mod 77 = 14 \]

• Alice translates message to letters to read HELLO

• **Security property:** No one else could read it, as only Alice knows her private key and that is needed for decryption
RSA Example: Authentication/Integrity

- Take $p = 7$, $q = 11$, so $n = 77$ and $\phi(n) = 60$
- Alice chooses $e = 17$, making $d = 53$
  - Verify: $ed = 901 = 1 \mod 60$
- Alice wants to send Bob message HELLO (07 04 11 11 14) so Bob knows it is what Alice sent (no changes in transit, and authenticated):
  - $07^{53} \mod 77 = 35$
  - $04^{53} \mod 77 = 09$
  - $11^{53} \mod 77 = 44$
  - $11^{53} \mod 77 = 44$
  - $14^{53} \mod 77 = 49$
- Alice sends 35 09 44 44 49 (the signature) in together with HELLO (07 04 11 11 14)
RSA Example: Authentication/Integrity

- Bob receives 35 09 44 44 49 (signature) and HELLO (07 04 11 11 14)
- Bob uses Alice’s public key, $e = 17$, $n = 77$, to verify message:
  - $35^{17} \mod 77 = 07$
  - $09^{17} \mod 77 = 04$
  - $44^{17} \mod 77 = 11$
  - $44^{17} \mod 77 = 11$
  - $49^{17} \mod 77 = 14$
- Bob translates message to letters to verify HELLO
- Security property:
  - Alice sent it as only she knows her private key, so no one else could have “encrypted” it
  - If (enciphered) message’s blocks (letters) are altered in transit, the message would not be “decrypted” properly
RSA Example: Both

• Alice wants to send Bob message HELLO both encrypted and authenticated (integrity-checked)
  – Alice’s keys: public (17, 77); private 53
  – Bob’s keys: public (37, 77); private 13

• Alice encrypts HELLO (07 04 11 11 14):
  – \((07^{53} \mod 77)^{37} \mod 77 = 07\)
  – \((04^{53} \mod 77)^{37} \mod 77 = 37\)
  – \((11^{53} \mod 77)^{37} \mod 77 = 44\)
  – \((11^{53} \mod 77)^{37} \mod 77 = 44\)
  – \((14^{53} \mod 77)^{37} \mod 77 = 14\)

• Alice sends 07 37 44 44 14
RSA Security

- At present, 1024-bit keys are considered secure, but 2048-bit keys are **recommended**

- **Tips** for making $n$ hard to be factorized
  - $p$ and $q$ lengths should be similar (e.g., ~500 bits each if key is 1024 bits)
  - Both $p - 1$ and $q - 1$ should contain a “large” prime factor
  - $\gcd(p - 1, q - 1)$ should be “small”
  - $d$ should be larger than $n^{0.25}$

- **Some attacks on RSA**
  - Mathematical attacks (factor $n$, compute $d$ from $e$) → extremely difficult
  - Brute force
  - Probable-message attacks
  - Timing attacks

- **How to prevent attacks?**
  - Large key
  - Random padding (RSA + OAEP)
  - Message blinding
Announcements

• Initial project report: due March 6, 2022
  – 2-3 pages, double-column ACM format, PDF report prepared using LaTeX
  – The report should be well-formatted and include:
    ▪ Title, authors, abstract
    ▪ Introduction (problem motivation, key ideas, proposed contributions)
    ▪ Approach overview (short summary of the approach)
    ▪ Related work (comprehensive survey of related papers and tools, their pros and cons, why your proposed approach/solution is better)
    ▪ Project timeline and expected milestones
    ▪ References

• Midterm project progress report: due April 3, 2022

• Weekly/biweekly paper reading assignments start very soon
Outline

• What Is Cryptography? Why Cryptography?
• Cryptographic Hash Functions
• Hash-Based Primitives
  – Digital Signature
  – HMAC
  – Hash Chain, Hash List
  – Merkle Tree
• Public Key Cryptography
  – RSA
  – Digital Signature Algorithm (DSA)
  – Elliptic Curve Digital Signature Algorithm (ECDSA)
Digital Signature Algorithm (DSA)

- Useful only for digital signing (no encryption or key exchange)
- Components
  - SHA-1 to generate a hash value (some other hash functions also allowed now)
  - Digital Signature Algorithm (DSA) to generate the digital signature from this hash value
- Designed to be fast for the signer rather than verifier
- Based on discrete logarithmic hard problem
  - Given $y_M$, hard to find $x_M$ s.t. $y_M = g^{x_M} \mod p$
DSA Public Parameters

Announce public parameters used for signing

Pick \( p \) as a prime with \( \geq 1024 \) bits

Pick \( q \) as a 160-bit prime such that \( q | (p-1) \)

Choose \( g \equiv h^{(p-1)/q} \mod p \),

where \( 1 < h < (p-1) \) such that \( g > 1 \)

Note: \( g \) is of order \( q \mod p \) (i.e., smallest \( q \) s.t. \( g^q \equiv 1 \mod p \))

\[
\begin{align*}
p &= 103 & q &= 17 \, \text{(divides 102)} \\
\text{powers of 64 mod 103} &= 64, 79, 9, 61, 93, 81, 34, 13, 8, 100, 14, 72, 76, 23, 30, 66, 1
\end{align*}
\]

17 values
**DSA Key Generation and Signing**

**Key Generation**
- Alice generates a long-term **private** key $x_M$
  - Random integer $0 < x_M < q$
- Alice generates a long-term **public** key $y_M$
  - $y_M = g^{x_M} \mod p$
- Alice randomly picks a private key $k$ such that $0 < k < q$, and generates $k^{-1} \mod q$

**Signing phase**
- Signing message $M$
  - Public key $r = (g^k \mod p) \mod q$
  - Signature $s = (k^{-1}(H(M) + x_M \cdot r)) \mod q$
- Send $(M, r, s)$

**Examples**
- $x_M = 13$
- $y_M = 64^{13} \mod 103 = 76$
- $k = 12; k^{-1} = 12^{-1} \mod 17 = 10$
- $H(M) = 75$
- $r = (64^{12} \mod 103) \mod 17 = 4$
- $s = (10 \cdot (75 + 13 \cdot 4)) \mod 17 = 12$

$(M, 4, 12)$
Verification

- **Public parameters**: \( g, p, q, y_M \)
- Received from signer: \( M, r, s \)

\[
\begin{align*}
  w &= (s)^{-1} \mod q \\
  u_1 &= [H(M)w] \mod q \\
  u_2 &= (r * w) \mod q \\
  v &= [(g^{u_1} \cdot y_M^{u_2}) \mod p] \mod q
\end{align*}
\]

- \( p = 103, q = 17, g = 64, y_M = 76 \)
- \( M, 4, 12 \)
- \( H(M) = 75 \)
- \( w = 12^{-1} \mod 17 = 10 \)
- \( u_1 = 75 \cdot 10 \mod 17 = 2 \)
- \( u_2 = 4 \cdot 10 \mod 17 = 6 \)
- \( v = (64^2 \cdot 76^6) \mod 103 \mod 17 = 4 \)

If \( v = r \), then the signature is verified
**DSA Security**

- Given $y_M$, it is difficult to compute $x_M$
  - $x_M$ is the discrete log of $y_M$ to the base $g$, mod $p$ (i.e., $y_M = g^{x_M} \mod p$)

- Similarly, given $r$, it is difficult to compute $k$

- Cannot forger a signature without $x_M$

- Signatures are not repeated (used once per message) and cannot be replayed

- **Faster at signing than RSA, but slower at verifying than RSA**

- Key lengths of 2028 bits and greater are also allowed
Outline

• What Is Cryptography? Why Cryptography?
• Cryptographic Hash Functions
• Hash-Based Primitives
  – Digital Signature
  – HMAC
  – Hash Chain, Hash List
  – Merkle Tree
• Public Key Cryptography
  – RSA
  – Digital Signature Algorithm (DSA)
  – Elliptic Curve Digital Signature Algorithm (ECDSA)
Why Elliptic Curve Cryptography (ECC)?

- **Shorter key size** than conventional PKCs (RSA, discrete logarithm-based (e.g., DSA))
  - Because the elliptic curve discrete logarithm problem (ECDLP) is much harder
- **Lower computation overhead**
  - Due to shorter key
  - Less data => faster transactions (important for blockchain)
- **ECDSA**: an elliptic curve implementation of DSA
  - Signature scheme used in Bitcoin; every Bitcoin address is a cryptographic hash of the ECDSA public key

<table>
<thead>
<tr>
<th>Security level (bits)</th>
<th>RSA/DL-based key size (bits)</th>
<th>ECC key size (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>512</td>
<td>112</td>
</tr>
<tr>
<td>80</td>
<td>1024</td>
<td>160</td>
</tr>
<tr>
<td>112</td>
<td>2048</td>
<td>224</td>
</tr>
<tr>
<td>128</td>
<td>3072</td>
<td>256</td>
</tr>
<tr>
<td>192</td>
<td>7680</td>
<td>384</td>
</tr>
<tr>
<td>256</td>
<td>15360</td>
<td>512</td>
</tr>
</tbody>
</table>
Elliptic Curve Cryptography

An elliptic curve (EC) consists of all elements \((x, y) \in \mathbb{F}\) satisfying

\[
y^2 = x^3 + ax + b
\]
Elliptic Curve Cryptography

- **Point addition:** Let P and Q be two EC points
  \[ P + Q = R = (x, -y), \]
  \[(x, y) = -R := \text{intersection of EC and PQ-line} \]

- **Point negation:** \( P + (-P) = O \)
  - O: identity point at infinity (not on the curve)
  - P: (x, y); -P: (x, -y)

- **Point doubling:** \( R = P + P = (x', -y') \),
  \[(x', y') = -R := \text{intersection of EC and tangent line of P} \]

- **Point multiplication:** achieved via double-and-add
  - Similar to multiply-and-square trick
  - e.g., \( Q=7P, 7 = (111)_2, Q = 0, R=P \)
    - \( Q += R \& R*2; Q+=R \& R*2; Q+=R \& R*2 \)
Elliptic Curve Cryptography

- **The Group Law**: The points on an elliptic curve form an additive group with an identity $O$. 

---

https://en.wikipedia.org/wiki/Elliptic_curve#The_group_law
Elliptic Curve Cryptography

Point addition and point doubling (arithmetic)

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two points in the elliptic curve $y^2 = x^3 + ax + b$.

- $P + O = O + P = P$
- If $x_1 = x_2$ and $y_1 = -y_2$, that is $Q = (x_1, -y_1) = -P$, then $P + Q = P + (-P) = O$
- If $Q \neq -P$ then $P + Q = (x_3, y_3)$ can be calculated by:
  
  
  \[
  \begin{align*}
  x_3 &= \lambda^2 - x_1 - x_2 \\
  y_3 &= \lambda(x_1 - x_3) - y_1
  \end{align*}
  \]

  where

  \[
  \lambda = \begin{cases} 
  \frac{y_2 - y_1}{x_2 - x_1}, & \text{if } P \neq Q \text{ (point addition)} \\
  \frac{3x_1^2 + a}{2y_1}, & \text{if } P = Q \text{ (point doubling)}
  \end{cases}
  \]
Elliptic Curve Cryptography

• In cryptography, we are interested in **elliptic curves modulo a prime** $p$

  $$y^2 = x^3 + ax + b \mod p$$

• Points on an elliptic curve and the infinity point $O$ form a **cyclic group**
  - E.g., $y^2 = x^3 + 2x + 2 \mod 17$
    - $P=(5,1), 2P=(6,3), 3P = 2P+P = (10,6), \ldots, 18P=(5,16), 19P=O, 20P=19P+P=O+P=P, \ldots$
    - EC has order $|E|=19$ as there are 19 points in its cyclic group

• **How many points** in an arbitrary EC?
  Given an elliptic curve $E$ modulo $p$, the number of points on $E$ is bounded by
  $$p + 1 - 2\sqrt{p} \leq |E| \leq p + 1 + 2\sqrt{p} \text{ (Hasse Theorem)}$$

Number of points close to prime $p$
Elliptic Curve Cryptography

- Rely on EC-discrete logarithmic hard problem
  - Given \((G, Y) \in EC\) s.t. \(Y = k \cdot G\) (\(Y\) is \(G\) added to itself \(k\) times), hard to find \(k\)

- ECC key size smaller than RSA and discrete log-based cryptography
  - Attacks on EC groups are weaker than factorizing algorithm or discrete log attacks

- Best known attacks
  - Baby-step, giant step
  - Pollard’s rho algorithm

- ECDSA
  - Asymmetric, based on DSA
  - Used another mathematical approach to key generation
    - Operations on points of EC
ECDSA Public Parameters

Public parameter generation

Pick $p$ as a prime with $\geq 160$ bits

$\text{Pick } a, b \text{ to form an EC}$

Pick an ECC generator $G$ with order $n$

$n \times G = 0$

How to choose $G$ and $n$?

Multiplication of $G \mod p =$

$(5,1)$ $(6,3)$ $(10,6)$ $(3,1)$ $(9,16)$ $(16,13)$ $(0,6)$ $(13,7)$ $(7,6)$ $(7,11)$ $(13,10)$ $(0,11)$ $(16,4)$ $(9,1)$ $(3,16)$ $(10,11)$ $(6,14)$ $(5,16)$ $(0)$

19 points

$(p,a,b,G,n)$ are public parameters

$p = 17$

$y^2 = x^3 + 2x^2 + 2x$ ($a = 2, b = 2$)

$G = (5,1), n = 19$
ECDSA Key Generation and Signing

**Key Generation**

Alice generates a long-term **private** key \( d_A \)

Random integer \( 0 < d_A < n \)

Alice generates a long-term **public** key \( Q_A \)

\[ Q_A = d_M \times G \mod p \]

**Signing phase:** To sign message \( M \)

Select an ephemeral key \( k \) from \([1, n - 1]\)

Compute an EC point \((x_1, y_1) = k \times G\)

Compute \( r = x_1 \mod n \) (choose other \( k \) if \( r = 0 \))

Compute \( s = k^{-1} (z + r \cdot d_A) \mod n \) (choose other \( k \) if \( s = 0 \))

Signature \( \sigma = (r, s) \)

Send \((M, \sigma)\)
ECDSA Verification

Verification

Public parameters: \(a, b, G, n, Q_A\)
Received from signer: \(M, r, s\)

\[
\begin{align*}
\text{Compute EC point } (x_1, y_1) &= u_1 \times G + u_2 \times Q_A \\
\text{If } (x_1, y_1) &= 0, \text{ invalid signature} \\
\text{If } r &\equiv x_1 \mod n, \text{ valid signature. Invalid otherwise}
\end{align*}
\]

\[
\begin{align*}
u_1 &= z \cdot s^{-1} \mod n \\
u_2 &= r \cdot s^{-1} \mod n \\
M, 10, 12 &\quad z = H(M) = 5 \\
(9, 16) &\quad a = 2, b = 2, p = 17, n = 19, G = (5,1), Q_A = (9,16) \\
10 \cdot 12^{-1} \mod 19 &= 4 \\
5 \cdot 12^{-1} \mod 19 &= 2 \\
x_1 = 10, r = 10 \\
(6, 3) + (5, 1) = (10, 6)
\end{align*}
\]
Some Popular ECs

**Curve25519** (Montgomery curve)
\[
y^2 = x^3 + 486662x^2 + x
\]
\[
p = 2^{255} - 19
\]

**Secp256k1** (used in Bitcoin)
\[
y^2 = x^3 + 7
\]
\[
p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1
\]
Other ECC-Based Primitives

• ECC replaces modular arithmetic operations in conventional PKC by operations defined over the elliptic curve

• ECC primitives can be easily constructed by making analogous changes to the corresponding conventional PKC
  – ECC encryption from ElGamal encryption
  – ECC-DH key exchange from Diffie-Hellman key exchange
  – ECC-DSA signature from DSA signature
Summary

• Cryptography enables secure/private communications in the present of adversaries

• Other important topics we haven’t covered
  – Pseudorandom number generator (PRG)
  – Symmetric encryption, stream cipher, block cipher, DES, AES
  – Key negotiation, Diffie-Hellman Key Exchange (DHKE)
  – Key management, public key infrastructure (PKI), digital certificates
  – …

• Advanced topics in cryptography
  – Private information retrieval
  – Searchable encryption
  – Homomorphic encryption
  – Oblivious ram
  – Zero-knowledge proof
  – Secure multi-party computation
  – …
Thanks!