Security of Systems and Networks

> TextBook: Information Security: Principles and Practice, 3rd edition
>
> Original Slides: http://www.cs.sjsu.edu/~stamp/infosec/powerpoint/
>
> Lecture Notes by Ahmed Nouralla

Chapter 1 - Introduction

- **CIA Triad:**
  - **Confidentiality:** prevent unauthorized reading of information.
  - **Integrity:** detect unauthorized writing of information.
  - **Availability:** data is available in a timely manner when needed.
- **Course Focus:**
  - **Cryptography:** classic cryptography, symmetric ciphers, public key crypto, hash functions, cryptanalysis
  - **Access control:** authentication (passwords, biometrics, ...) and authorization (ACL, capability list, multilevel security, security modeling, covert channels, inference control, Firewalls, IDS)
  - **Protocols:** simple authentication protocols and case studies (SSH, SSL, IPSec, Kerberos, WEP, GSM)

- **Software:** security flaws (buffer overflows, race conditions) and malware (viruses, worms, …), reverse engineering, digital rights management, OS security,

Chapter 2 - Crypto Basics

- **Cryptology:** the art and science of making [cryptography] and breaking [cryptanalysis] “secret codes”
  - Main fields in cryptography: symmetric, asymmetric (public-key), and hash functions.
  - Cryptanalysis techniques are based on given info: ciphertext only, known plaintext, chosen plaintext, …
- **Terminology:**
  - A cipher or cryptosystem is used to encrypt the plaintext
  - The result of encryption is ciphertext. We decrypt ciphertext to recover plaintext
  - A key is used to configure a cryptosystem
    - A symmetric key cryptosystem uses the same key to encrypt as to decrypt
    - A public key cryptosystem uses a public key to encrypt and a private key to decrypt
  - **Kerckhoffs’ Principle:** the security of a cryptosystem must lie in the choice of its keys only; everything else (including the algorithm itself) should be considered public knowledge.
- **Popular cryptosystems:**
  - **Substitution ciphers**: replacing individual letters (e.g., Caesar Cipher: shift by 3 [a -> d, x -> a])
    - **General case:** 26! substitution alphabets possible.
    - **Cryptanalysis:** consider character frequencies in ciphertext.
  - **Transposition ciphers:** change the order of letters
    - **General case:** <= n! plaintext permutations.
  - **One-time Pad:** ciphertext = plaintext XOR key
    - Provably secure: requires truly random keys, no key reuse, key length = plaintext length.
    - Impractical: why not securely pre-share the message itself instead of the key.
  - **Codebook:** individual plain/cipher mapping for words.
- **History:**
  - WWII - golden age of cryptanalysis: Zimmerman Telegram, Japanese Purple, German Enigma
  - Claude Shannon: founded the field of information theory.
    - **Confusion:** obscure relationship between plaintext and ciphertext
    - **Diffusion:** spread plaintext statistics through the ciphertext
  - Data Encryption Standard (DES), 70’s
  - Public Key cryptography, 70’s
  - Crypto conferences, 80’s
  - Advanced Encryption Standard (AES), 90’s

Chapter 3 - Symmetric Key Crypto

Stream ciphers

Generalized one-time pad, with keys stretched into a long keystream

- Processes input continuously given the keystream and is typically faster than block cipher
- Combining key bit and data bit typically uses XOR.
- Not very popular today compared to block ciphers.
- **A5/1:**
  - Based on shift registers, was used in GSM (cellular networks). Efficient in hardware.
  - Initially a secret, became public knowledge and serious weaknesses were identified.
  - Pseudocode

```
# Initially fill three fixed-size binary registers with zeros
X = 19 * "0"
Y = 22 * "0"
Z = 23 * "0"

# Generate a random 64-bit secret key K
K = 64 * either("0", "1")

# Use the key to initialize registers
for i in range(64):
    X[0] ^= K[i]
    Y[0] ^= K[i]
    Z[0] ^= K[i]
    shift_registers()

# Method to potentially shift some registers and prepend a new bit
def shift_registers():
    m = majority(X[8], Y[10], Z[10])
    if m == X[8]:
        X = (X[13] ^ X[16] ^ X[17] ^ X[18]) + X[:-1]
    if m == Y[10]:
        Y = (Y[20] ^ Y[21]) + Y[:-1]
    if m == Z[10]:
        Z = (Z[7] ^ Z[20] ^ Z[21] ^ Z[22]) + Z[:-1]

# Encrypt plaintext using the keystream
plain = "whatever"
cipher = ""
for letter in plain:
  cipher += letter ^ (X[-1] ^ Y[-1] ^ Z[-1])
  shift_registers()
```

- **RC4:**
  - Based on a self-modifying list from which we extract the next keystream byte. Efficient in software.
  - Pseudocode

```
# Start with a sequence of every possible 1-byte char
for i from 0 to 255
    S[i] = i

# Key Scheduling Algorithm, permutes S using a random secret K
j = 0
for i from 0 to 255
    j = (j + S[i] + K[i % len(K)]) % 256
    swap(S[i], S[j])

# Pseudo-Random Generation Algorithm, to obtain the next keystream byte
def generate_byte(i=0, j=0):
    i = (i + 1) % 256
    j = (j + S[i]) % 256
    swap(S[i], S[j])
    t = (S[i] + S[j]) % 256
    return S[t]
```

- Note: first 256 bytes should be discarded. Otherwise, related key attacks exist

Block ciphers

- Generalized codebook, in which a key determines the codebook. Usually implemented in software.
  - Processes input one block at a time, more common and can reuse keys.
  - Plaintext and ciphertext consist of fixed-sized blocks
  - In iterative block ciphers, ciphertext is obtained from plaintext by iterating a *round function*
    - Input to the *round function* consists of key and output of previous round
- **Feistel cipher**
  - A type of block cipher that splits plaintext into left and right halves, then, for n rounds with n subkeys (extracted from the master key), implement the Encryption/Decryption procedures.
- **Data Encryption Standard (DES)**
  - A Feistel cipher with 64bit blocks, 56bit key, 16 rounds, and 48-bit subkeys.
  - DES had no back-door, but exhaustive key search was possible due to short key length.
  - Fiestel function of DES involves four stages
    - Expansion: function that expands the 32-bit half-block into 48-bits.
    - Key mixing: XORing the expanded result with the subkey.
    - Substitution: substituting and shrinking the output using 8 S-Boxes (reduce 6 to 4 bits).
    - Permutation: rearranging output according to a fixed permutation.
  - 3DES uses three keys (or two if K1=K3) to mitigate DES issue, resulting in an effective key-size of 112 or 168 bit, respectively.

$$C = E_{K_3}(D_{K_2}(E_{K_1}(P)))$$
$$P = D_{K_1}(E_{K_2}(D_{K_3}(C)))$$

- **Advanced Encryption Standard (AES)**

- Replaced DES with 128bit blocks, key length varies (128, 192, 256), and 10 to 14 rounds (depending on key length).
- Each round uses 4 functions: ByteSub, ShiftRow, MixColumn, and AddRoundKey.

Block cipher modes

> Check: https://ctf101.org/cryptography/what-are-block-ciphers/

Modes on how to encrypt/decrypt text blocks with a block cipher.

- **Electronic CodeBook (ECB):** encrypt each block independently
  - Problem 1: repeated plaintext is seen in repeated ciphertext, allowing cryptanalysis inferences.
  - Problem 2: no integrity checks, one may substitute or rearrange ciphertext blocks.

Encryption	Decryption
C0 = E(P0, K)	P0 = D(C0, K)
C1 = E(P1, K)	P1 = D(C1, K)
C2 = E(P2, K)	P2 = D(C2, K)

- **Cipher Block Chaining (CBC):** solves both problems of ECB by having the plaintext block XORed with the previous ciphertext block before being passed to the encryption algorithm.
  - First plaintext block is XORed with an Initialization Vector (IV) that's generated randomly and needed for decryption.

Encryption	Decryption
C0 = E(IV ^ P0, K)	P0 = IV ^ D(C0, K)
C1 = E(C0 ^ P1, K)	P1 = C0 ^ D(C1, K)
C2 = E(C1 ^ P2, K)	P2 = C1 ^ D(C2, K)

  - The final cipher block can be used as a Message Authentication Code (MAC) for integrity verification. Keys used for encryption and calculating MAC must be different.
  - A corrupted block C{n} will only affect P{n} and P{n+1}. Corruption won't propagate further as the decryption relies on ciphertext and previous plaintext block.
  - CBC encryption cannot be parallelized, but decryption can.
- **Counter Mode (CTR):**
  - Popular for random access. Encrypt or decrypt by XORing with E(IV + Counter, K).

Encryption	Decryption
C0 = P0 ^ E(IV+0, K)	P0 = C0 ^ E(IV+0, K)
C1 = P1 ^ E(IV+1, K)	P1 = C1 ^ E(IV+1, K)
C2 = P2 ^ E(IV+2, K)	P2 = C2 ^ E(IV+2, K)

Chapter 4 - Asymmetric (Public) Key Crypto

Public key crypto is about the use of two keys (public and private), such that a message encrypted with one can only be decrypted by the other.

- The core concept employs one-way functions (easy to compute, hard to inverse) to create a trapdoor operation that is difficult to reverse by attackers.
- In reality, a hybrid cryptosystem is preferred in which asymmetric crypto is used to share the symmetric key for usage in further communications (as it's more efficient).
- **Main usages:** public key crypto is typically used to create
  - **Cipher text:** by encrypting secrets with the public key of the intended recipient.
  - **Digital Signatures:** by encrypting data (typically message hash) with the private key of its origin.
  - **Digital Certificates:** trusted authorities signing an electronic document that associates an entity (e.g., a person, organization, or website) with their public key.

Important Concepts

1. **Message Integrity:** a message was not altered from some "reference version"
   - **Examples:** Internet/UDP Checksum, CRC, Hash functions (e.g., MD, SHA).
2. **Authenticity (Message Authentication):** the "reference version" came from the someone with knowledge of a shared secret (i.e., either Alice or Bob - assuming no exploits).
   - **Example:** Message Authentication Code (HMAC, CMAC, GMAC)
3. **Non-Repudiation:** One can publicly verify that X authored M (i.e., using X's public key).
   - **Example:** Digital Signature (e.g., with RSA or DSA)

Note that (3) implies (2) implies (1), but not the other way around.

Merkle–Hellman Knapsack Cryptosystem

- One of the earliest public-key cryptosystems, its security relied on the complexity of solving the subset sum problem (a certain instance of the knapsack problem).
  - Given a list of n weights W and a sum S, identify which w_i 's to pick that add up to S.
- **Key generation**
  - Start with a super-increasing sequence W (each weight greater than the sum of all previous weights)
    - This reduces the problem to just looping over W from right-to-left and subtracting w_i from S whenever we can until reaching $S = 0$.
  - Choose coprimes n and m, with $n > \Sigma W$
  - Private key: (W, n, m) and public key: (B, n) where $B = \{(m \cdot w_i)\%n)\}_{i=1}^{n}$
  - Idea: the problem with super-increasing W is solvable, but when multiplying weights by the secret m, it becomes NP-Complete. Only one with knowledge of m can reduce the problem back to the solvable instance.
- **Encryption and decryption**
  - Encrypting n-sized plaintext bits $p_i \in \{0, 1\}$ yields ciphertext (number) $C = \Sigma_{i=1}^{n}(B_i * p_i)$

    - Decryption: solve super-increasing knapsack with $S = C * m^{-1} \bmod n$
        - $m^{-1} \bmod n$ is the multiplicative inverse of m modulo n. Found using the extended euclidean algorithm (check RSA below).
- An attack was found in 1983, using lattice reduction (whatever that is).

Rivest-Shamir-Adleman (RSA)

- The most popular public-key cryptosystem, defines how to create key pairs to establish confidentiality. Another very popular one is Digital Signature Algorithm (DSA).
- Based on the prime-factorization problem: given n, find primes p and q such that $n = pq$.
- **Underlying Mathematics**
    - Let p and q be two large prime numbers and $n = pq$ be the modulus
    - Choose e coprime with $\phi(n) = (p-1)(q-1)$
        - Coprime means they don't share a common factor: their GCD is 1.
    - Find d such that $ed \bmod \phi(n) = 1$ (use Extended Euclidean Algorithm)
        - e and d are multiplicative inverses modulo $\phi(n)$.
        - The algorithm would express $1 = de + k\phi(n)$ for some integer k.
    - Public key is (n, e) and private key is (n, d)
    - Encryption: $C = M^e \bmod n$. Decryption: $M = C^d \bmod n$
    - **Notes:**
        - A nice fact: $x^y \bmod n = x^{y \bmod \phi(n)} \bmod n$. It helps to calculate modular exponents faster.
        - RSA with low e is susceptible to Cube Root Attacks. Commonly used value is $e = 2^{16} + 1$

Diffie-Hellman (DH)

- A key-exchange algorithm used to securely establish a shared secret.
- Based on the discrete logarithm problem: given g, p, and $g^k \bmod p$, find exponent k.
- **Underlying Mathematics**
    - Public parameters are agreed on for a very large prime p and a generator g (e.g., specified in RFC3526) such that $g^i \bmod p$ generates all possible results in {1, ..., p-1} for i in the same range

      ```
      # Example with g=5 and p=23
      >>> sorted([(5 ** i) % 23 for i in range(1, 23)])
      [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
      21, 22]
      ```

    - Alice and Bob independently choose their integer secrets $1 \leq a, b \leq p-2$ and exchange $g^a \bmod p$ and $g^b \bmod p$. They independently compute the shared secret $g^{ab} \bmod p$

```
# Example with a=6, b=15
>>> a, b, g, p = 6, 15, 5, 23
>>> (g ** a) % p
8
>>> (g ** b) % p
19
>>> (g ** (a*b)) % p
2
```

  - Trudy, with knowledge of $g^a \bmod p$ and $g^b \bmod p$ cannot compute $g^{ab} \bmod p$ without solving the discrete log problem (finding a or b).
- **IMPORTANT:** DH is subject to Man-in-The-Middle attacks. Where Trudy exchanges $g^t \bmod p$ with both alice and bob. To mitigate that, one must combine DH with sender/receiver authentication (e.g., via signatures).

Elliptic Curve Cryptography (ECC)

- Elliptic Curve methods provide a different (more efficient) way (using fewer bits, but more complex operations) of doing the math in public-key systems (i.e., EC-RSA and EC-DH).
- The core idea is defining different semantics for addition (and thus, multiplication) of points on an elliptic curve: $y^2 = x^3 + ax + b$
  - Adding two points P, Q yields R: intersection of PQ line with the curve, mirrored by the x-axis.
  - Adding P to itself yields another point 2P: intersection of the tangent line at P to the curve, also mirrored around the x-axis.
  - To get kP, one has to iteratively add P to itself k times.
- EC version of Diffie-Hellman:
  - Use a point P = (x, y) on a certain curve as the public values.
  - As before, math is done modulo some large prime p (also public)
  - Alice and Bob independently compute aP and bP and exchange them.
  - The shared secret is abP (multiplication is commutative so abP = baP).
- Again, ECC advantage is that we can use smaller numbers (faster computations) to achieve the same security. ECC algorithms are popular on IoT and embedded devices.

Sign then Encrypt vs. Encrypt then Sign

- Reminder: Signing = Encryption with Private Key.
- Sign then Encrypt emphasizes confidentiality, sender signature is never leaked.
  - The main concern is that Bob can forward Alice's signed message to Charlie, tricking him that the message was intended for them.
- Encrypt then Sign emphasizes integrity/non-repudiation, yet sender signature is exposed.
  - The main concern is that Trudy can strip Alice's signature and replace it with her own, before forwarding the message to Bob. Of course, Trudy still cannot read message content.
- To avoid issues in both cases, one should add context. That is, perpend sender and receiver identities (e.g., names, email addresses, public keys) to the message content before doing cryptographic operations.

Public Key Infrastructure (PKI)

- The "stuff" needed to securely use Public Key Crypto. This includes
  - **Key generation and management:** generation, storage, exchange, and cryptographic use.
  - **Certification Authorities (CA):** reference trusted vendors that use their private keys to sign public keys of others, effectively verifying that a certain entity owns a certain public key.
  - **Certificate Revocation Lists (CRL)**: published by CAs to revoke or hold previously-issued certs for certain entities before expiry. This may be needed if their entity's private key was compromised, they no longer use the cert, or other mistake from the CA side.
- **PKI trust models:** - not actual technical terms, but accurately represents the idea.
  - **Monopoly**: one universally trusted CA. A single point of trust? No, thanks.
  - **Oligarchy:** few (~80) trusted CAs in browsers in systems, users can customize (add/remove).
  - **Anarchy:** anyone can be a "CA", users decide who to trust. Used in PGP's Web of Trust.

Chapter 5 - Hash Functions++

Hash functions in cryptography

Common use cases include:

- **Message Authentication:** combining a message with a symmetric encryption key to create a MAC.
- **Integrity checks (e.g., CRC):** detecting data corruption on hash mismatch.
- **Data signature efficiency:** signing a long message is costly to compute. It's typical and more secure to sign its hash (computed in blocks) instead.

Crypto hash functions should provide:

- Compression, efficiency, one-way (can't retrieve input given the hash).
- **Collision resistance:** for an n-bit hash, hashing random $2^{n/2}$ values is expected (p>0.5) to find a collision (birthday paradox).
  - **Weak collision resistance:** given an input, it's difficult to deduce a different input sharing the same hash value.
  - **Strong collision resistance (desired):** it's generally difficult to find any pair of different inputs hashing to the same value.
- **Avalanche effect (desired):** change of 1-bit in input affects about half of the output bits.

Sample hashing methods:

- **Adding input bits and taking mod 256:** not secure, easy to find collisions.
- **Adding input bits with a decreasing multiplier (n down to 1), then taking mod 256:** a little harder to find collision, but possible. Used in rsync (non-crypto tool for file transfer).
- **CRC:** good for detecting burst errors, but easy to construct collision. Mistakenly used in WEP.
- **MD5:** 16 byte output, it became easier to find collisions.

- **SHA-1:** 20 byte output, long-time US standard, deprecated as attacks were found.
- **SHA-2**: family of related algorithms providing 32 to 64 bytes. Widely used.
- **SHA-3:** developed via a competition (similar to AES) to prepare for a possible break of SHA-2.

Hash-based Message Authentication Code (HMAC)

- We cannot simply compute HMAC by hashing `key || msg` as it would be vulnerable to length-extension attack (i.e., deducing valid HMAC of `msg || extra` without knowledge of the key).
  - Note: Double-bars indicate concatenation.
- Hashing `msg || key` is better, but HMAC would be the same in case of messages with the same hash.
- Correct way (RFC 2104) for a hash with B byte block length.
  - `HMAC = hash((key ⊕ opad) || hash((key ⊕ ipad) || msg))`
  - `opad = 0x5C` repeated B times and `ipad = 0x36` repeated B times
- Recall the dictionary attack (rainbow table) that involves pre-computing hashes to invert hash functions for common words or small-length data. Recall the salting strategy that mitigates the issue by adding a long secret string to the data before hashing.

Chapter 9 - Authentication Protocols

Security protocol: Communication rules followed in a security application (e.g., SSL, IPSec, Kerberos, ...)

- Flaws (e.g., in WEP, IPSec) and implementation errors (e.g., IE implementation of SSL) can occur.
- Must satisfy precise requirements (e.g., regarding efficiency, robustness, flexibility).
- **Real-life examples:** ATM protocol (verify PIN), Identification Friend of Foe (radar challenging an inbound vehicle)

Authentication Protocols

- **Authentication:** proving identity to the other party, typically to establish a communication session.
  - **On a standalone computer:** simple (e.g., password), yet still subject to attacks (e.g., keylogging)
  - **Over the network:** more difficult and subject to more attacks (eavesdrop, intercept, inject, drop).
  - **Mutual authentication:** performing authentication in both directions (e.g., server-to-client and client-to-server).
- **Replay attack:** reusing information captured from an authentic context to establish a fraudulent one.
  - **Mitigation:** ensure freshness of authentication session using session IDs/tokens.
- **Reflection attack:** attacking mutual authentication by tricking a challenger to respond to its own challenge through a parallel session.

    - **Mitigations:** use a different authentication protocol in different directions, or require the challenged to respond first before presenting its own challenge.
- **Challenge/response:** authentication by requiring an answer to a generated question.
    - **Main goal:** prevent replay by ensuring that only the challenged party can provide the correct response that the challenger can verify.
    - **Example:** challenger sends nonce (number used once) and responder authenticates by replying with hash(password, nonce).
    - **Need to ensure:** strong hash function, unpredictable nonce, securely pre-shared password.
    - **Problems:** no mutual authentication (attacker can disguise as challenger) and subject to offline attacks (attacker captures multiple challenge/response pairs and use them to brute-forces the password).
    - **Solution:** authenticate using cryptographic techniques (e.g., proving knowledge of session key).
        - Session key is a one-time symmetric key used for encrypting communications during a session

Perfect Forward Secrecy (PFS)

- Captured secrets cannot be decrypted later even if long-term server/client keys were compromised.
- Achieved through one-time session keys that are derived independently by both parties
- **Example 1:** Ephemeral Diffie-Hellman in which the symmetric key is used to share public parameters for DH, while private parameters used by communicating parties are forgotten.
- **Example 2:** mutual signing of (server/client nonces + DH public parameters).

Zero-Knowledge Proofs (ZKP)

- A more-sophisticated form of challenge/response authentication
- Allows proving the knowledge of a secret s without revealing information about it.
    - Interactive example (password to door in a cave): the verifier randomly asks the prover to exit from one of two paths (one of which requires knowledge of the password)
    - Repeated success convinces the verifier of the prover's knowledge without disclosing the secret.
    - Probability to fool the verifier is $1/2^n$ gets smaller as n (the number of queries) increases.
- ZKP is used in Microsoft's Next-Generation Secure Computing Base (NGSCB).
- **Fiat-Shamir** creates an authentication protocol based on the interactive ZKP. Mathematics is based on the problem of finding square root modulo n: given $v = s^2 \bmod n$, find s.
    - **Prover:** commits to a random r and sends $m_1 = r^2 \bmod n$ to the verifier (where n is a large prime).
    - **Verifier:** sends a random challenge $c = 0$ or $c = 1$.
    - **Prover:** sends $m_2 = rs^c \bmod n$. In the first case, knowledge of secret is not needed!
    - **Verifier:** verifies $m_2^2 = m_1 v^c \pmod n$. Note that v is public as explained.

Final Remarks

- To achieve confidentiality, use encryption (i.e., with symmetric keys for efficiency)
  - Either use public key crypto (e.g., RSA) to exchange symmetric key.
  - Or (faster), use a dedicated key-exchange algorithm (e.g., Diffie-Hellman) with a KDF.
- To achieve sender/receiver authentication, use digital certificates
  - Trusted CAs vouching that a certain public key belongs to a certain entity.
- To achieve message authentication, use message authentication codes (e.g., HMAC)
  - Value computed using the message and a symmetric key proving that sender had knowledge of the key.
- To achieve non-repudiation, use digital signatures.
  - Signed messages point to their author given the author's certificate (with trusted public key).
- Protocols are designed depending on the desired properties regarding cost, protocol support, data sensitivity, etc.

Special Lecture - Blockchain

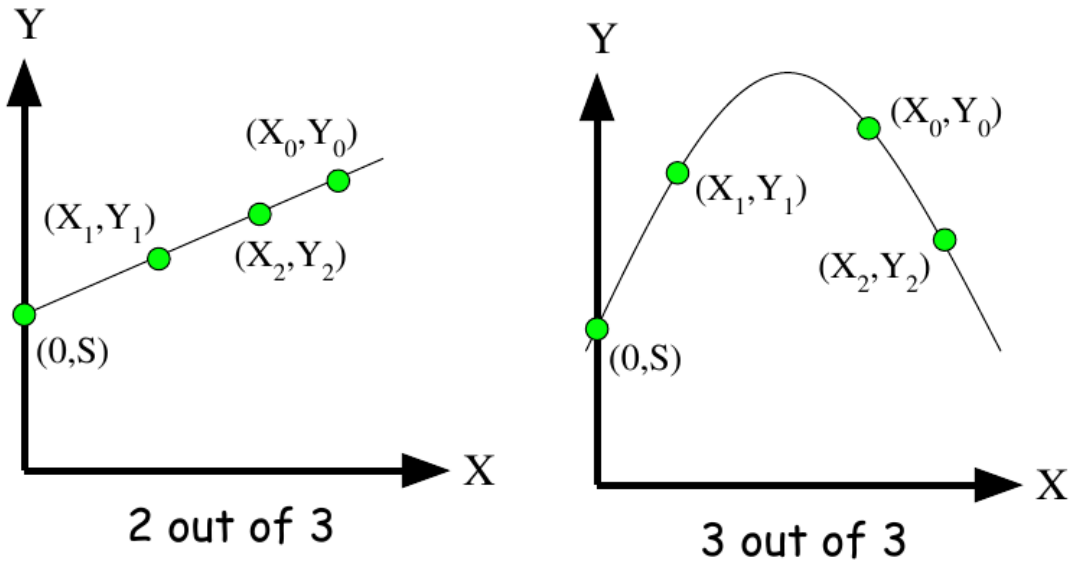
- DigiCash Inc. (1989-1998) was an early provider of anonymous publicly-verifiable electronic payments based on the concept of Blind Signatures (signing a message without viewing its content to maintain privacy).
- The BitCoin white paper introduced a decentralized digital currency that allows direct online payments between parties without relying on a trusted third-party like a bank.
- Main challenge was preventing double-spending without requiring centralized control
  - This is achieved through the blockchain: a public ledger (record) of all timestamped transactions (blocks) where each block references the previous one by hash.
  - The cluster with the largest computing power (hopefully composed of honest peers) can collectively approve/reject a transaction and thus, determine the trusted ledger.
- Mining (obtaining bitcoin) is done by computing a proof-of-work (PoW)
  - PoW is obtained by brute-forcing a nonce R such that $\text{SHA-256}(X) < D$, where
    - X is the block header (includes R, D, and other transaction metadata).
    - D is the level of difficulty of the challenge (more difficult -> higher reward).
  - Concept: to find R such that $\text{SHA-256}(R) < 2^n$, one expects 2^{32-n} trials at max.
- Concerns: identity verification through public keys and adjusting rewards as the computing power increases so money supply doesn't grow indefinitely.

Paper - How to Share a Secret

- The paper introduces Shamir's Secret Sharing (SSS) scheme for distributing trust among a group, such that a secret S is obtainable only if at least t members of the group share their individual keys (like a safe with multiple locks).
- The secret is hidden as the constant term of a polynomial of degree $t - 1$. That is,

$$f(0) = S$$

- Shares are unique non-zero points $(x_i, f(x_i))$ for the polynomial.
- Having t of them, one can use Lagrange interpolation to obtain S.
- Showing the number of points needed to find the secret point for a line and a quadratic curve.



- A popular application: seal/unseal process in Hashicorp Vault (Secret Management Software).