First Class: Introduction

Second Class: Cryptography Review

Cryptographic Hash Function

Mathematical Function

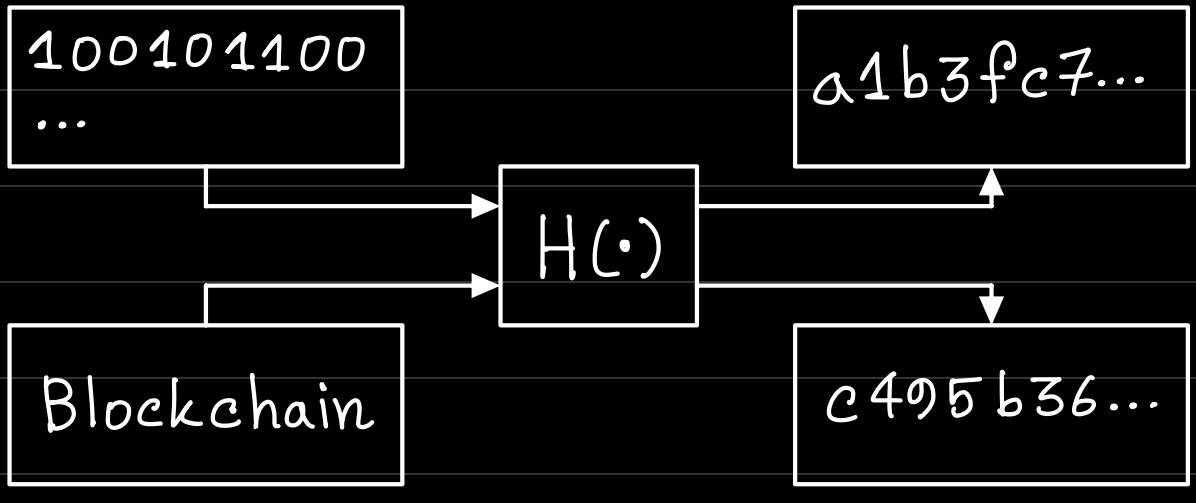
$f: A \rightarrow B$

$f(a) = b;\ a \in A,\ b \in B$

Mathemical function. Properties:

1. Input {any size}
2. Output {fixed size}
3. Time complexity: $O(n)$

Compression: $\{0,1\}^* \rightarrow \{0,1\}^n$



Collision-resistance } must-have
preimage resistance }

hiding } Desirable for certain blockchain system
puzzle-friendliness }

Collision Resistance

$x \neq y$; $H(x) = H(y)$ (must avoid this)

Very difficult to find collision,

So infeasible.

Infeasible: Hard to find a collision, but no, no collisions exist.

input size > output size

{The pigeonhole principle}

To find collision,

Output, $n = 3$ bits

Input, $n_0 = 2^n + 1$ bits

$= 2^3 + 1$ bits

$= 9$ bits

Since input_size > output_size, there must be a collision.

Worst Case: $2^n + 1$ times

Best case: $2^{n/2} + 1$ times

The previous way was brute force.

Some Hash function:

$H(x) = x \bmod 2^{256}$

One collision:
3 and $3 + 2^{256}$

$x \bmod 2^n \approx$ 2 bits
$x \bmod 4$ $\{0, 1, 2, 3\}$

Application:

Message Digest
Hash of any input, bits, random strings, characters, or even files.
Checksum: Hash of files

Pre-Image Resistance:

H: Hash Functions

$H(x)=y$

One way function

For essentially all pre-specified outputs y, it is computationally infeasible to find an x such that $H(x)=y$

$a21b34c5 \longrightarrow H^{-1}(\cdot) \longrightarrow ???$

Low min-entropy and High min-entropy

Sample space: $X=\{h,t\}$

$H(x)=y$

Can attacker find the value of x given y?

Low min-entropy to High min-entropy

Hiding: $H(r||x)$

r is a secret value

Tackles x picking up from a low min-entropy distribution.

Hiding: Application

Commitment Scheme

Who will win?

Argentina

Hiding: Commitment Scheme

com := commit(msg, key)

↓

hash function

verification := verify (com, msg, key)

Security Properties:

1. Hiding
2. Binding

Puzzle-Friendliness

$H(k||x) = y$

k is chosen from a distribution with high min entropy.

Infeasible to find x in time significantly less than 2^n.

Puzzle-Friendliness: Application

A target set Y, for a valid solution $z, z \in Y$

computation: $z = H(\text{puzzle-ID}||x)$

x changes until $z \in Y$

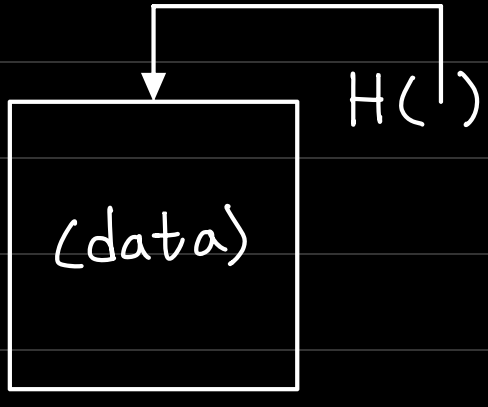
Family of hash functions:

* MD5 (Message Digest 5) → Currently considered broken!
* Secure Hashing Algorithm 1 (SHA1) → Currently considered broken!
* Secure Hashing Algorithm 2/3 (SHA2/3) → Safe to use, SHA3 preferrable.

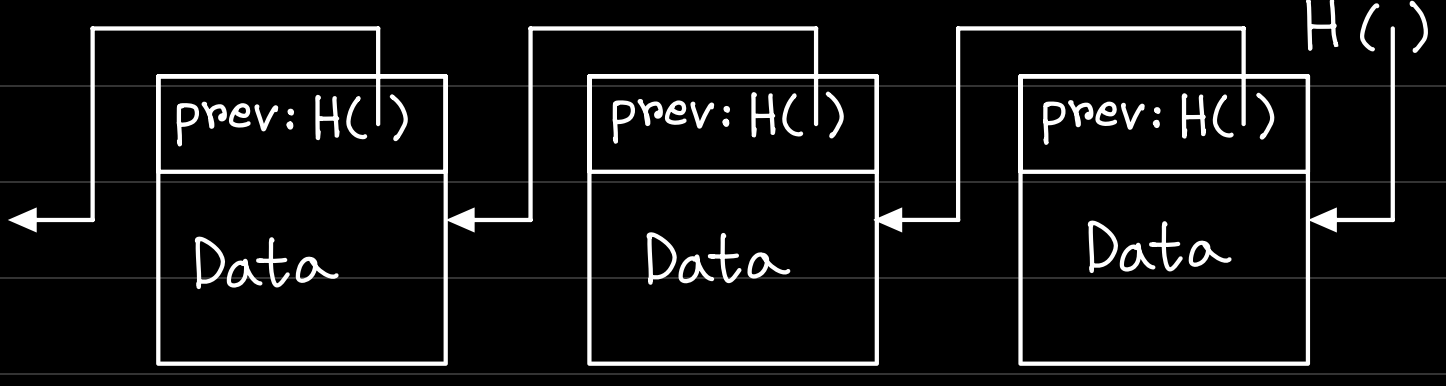
Third Class: Cryptography Review

Hash pointer: A pointer to where some information is stored together with a cryptographic hash of the information.

Pros: To get the info back
To verify that it hasn't changed

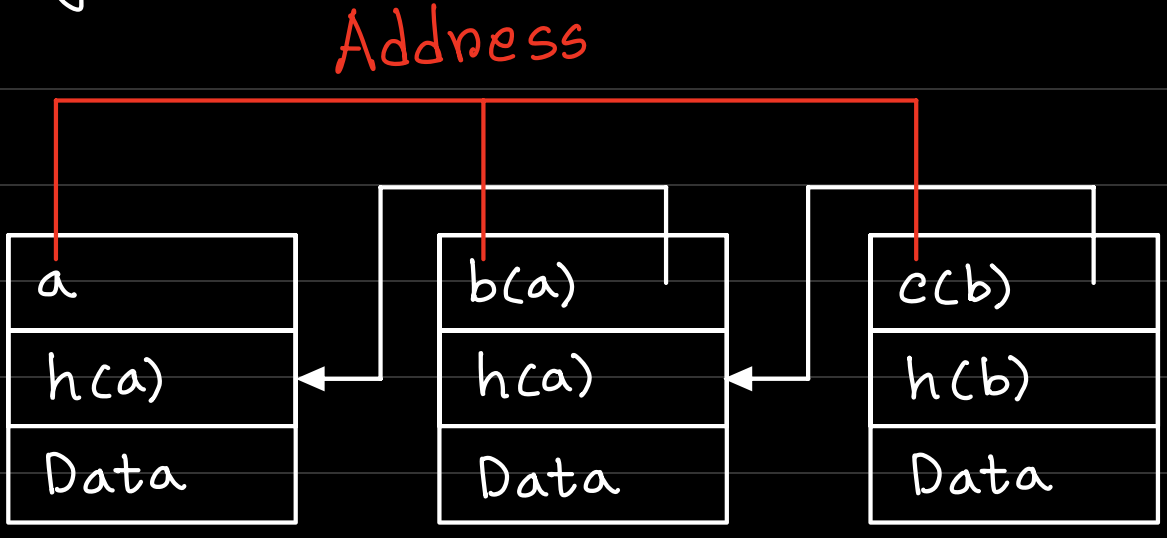


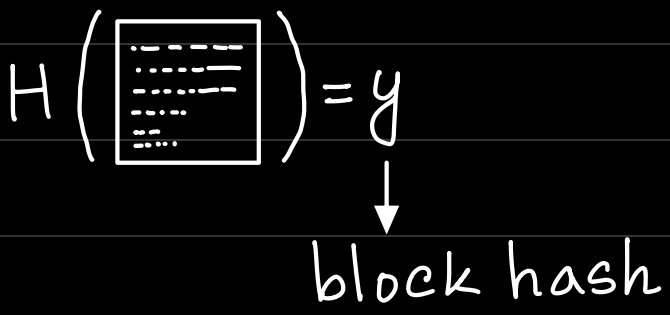
Linked List with Hash Pointer:



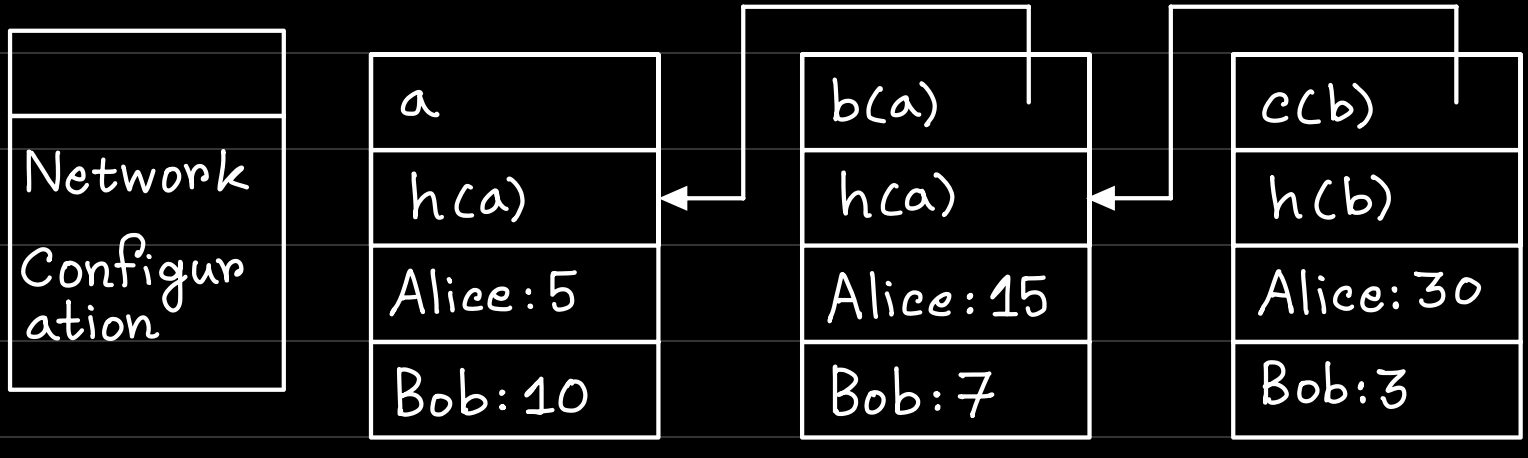
Tamper-Evident Log

$H(x) = y$





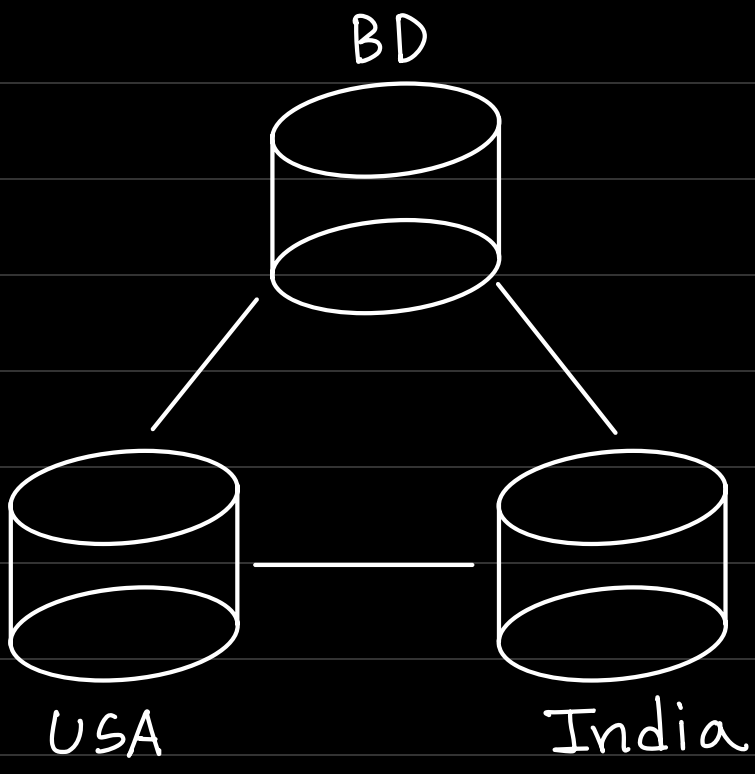
a	b	c
d	e	f
g	h	i



Blockchain!

$H(\begin{smallmatrix}A5\\B10\end{smallmatrix}) = abc65d\ldots$ $H(\begin{smallmatrix}A5\\B1000\end{smallmatrix}) = cba79c\ldots$

$H\binom{A5}{B10} \neq H\binom{A5}{B1000}$



Merkle Tree (Tree Algorithm)

A data structure using cryptographic hashes, basically a binary tree (Almost two leaves) with hash pointer.

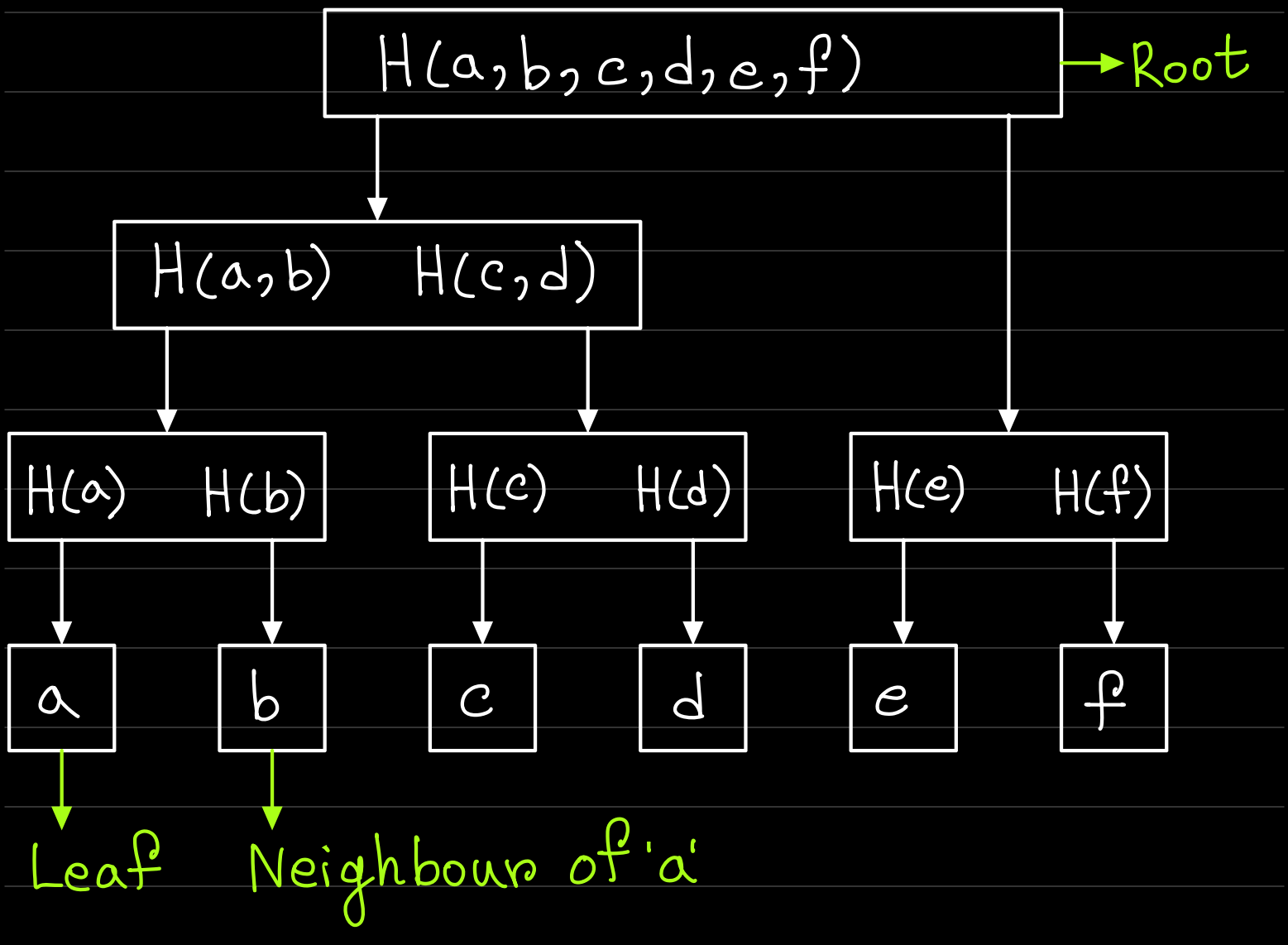
Time/Space Complexity: O(LogN)

Used as an efficient and secure

way to verify large data structure.

Block

$S = \{\underbrace{a, b, c, d, e, f}_{\text{Block}}\}$ Target: (f)



Proof of membership:

1. Proof of membership for the leaf with knowledge of the root.
2. Using the neighbour nodes to verify if it leads to the root.

Bloom filter: a probabilistic data structure which allows to test if an element is a member of set.

Test if $e \in S$

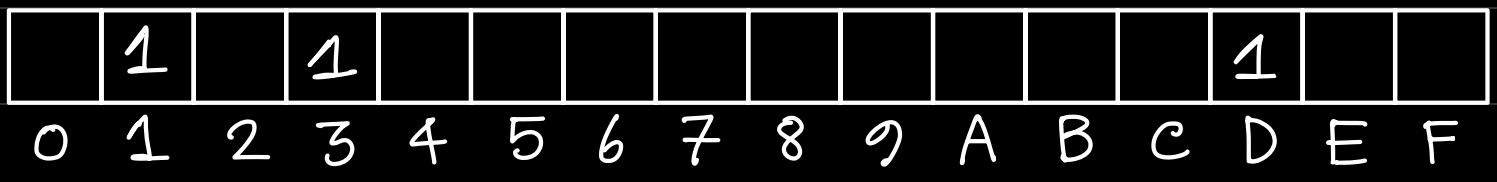
Returns either
True ("e possibly in set S")
or
False ("e definitely not in set S")

Set S: Apple
Element e: Apple

P_1 (add)
P_2 (evaluate)

H_1 (apple) = 1... MD5
H_2 (apple) = D... SHA1
H_3 (apple) = 3... SHA2

For 1D3,

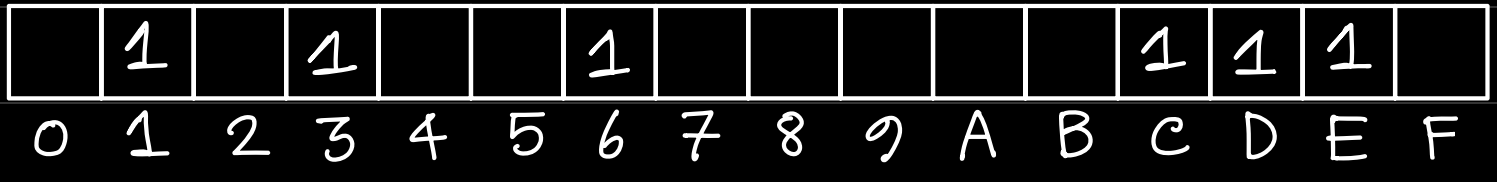


Binary Number: Either 1 or 0.

Set S: Apple, Lime
Element e: Lime

$H_1(e) = 6\ldots$
$H_2(e) = C\ldots$
$H_3(e) = E\ldots$

For 6CE,

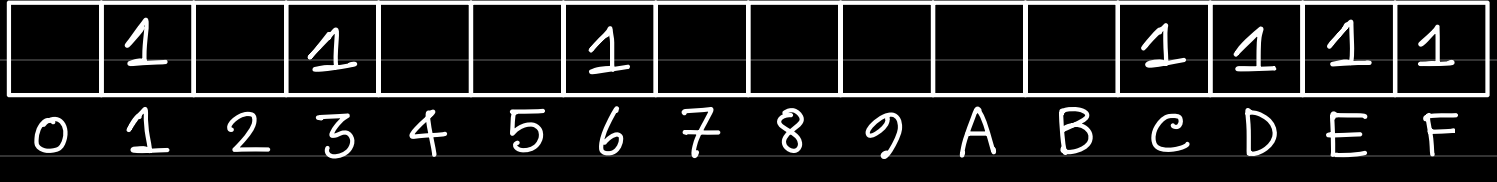


Set S: Apple, Lime, Lemon
Element e: Lemon

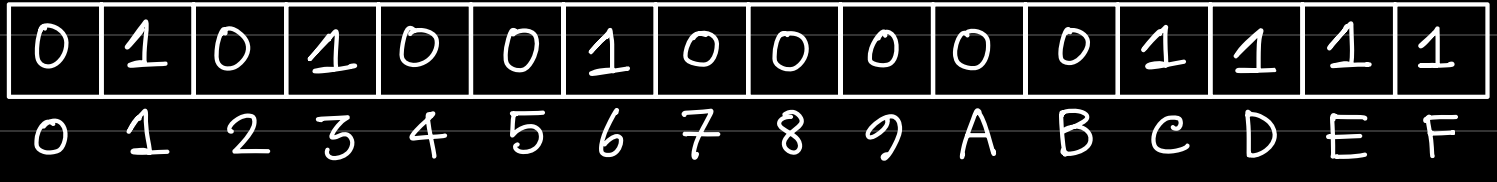
$H_1(e) = 3\ldots$
$H_2(e) = D\ldots$
$H_3(e) = F\ldots$

For 3DF,

✗ ✗ ✓



Finally,

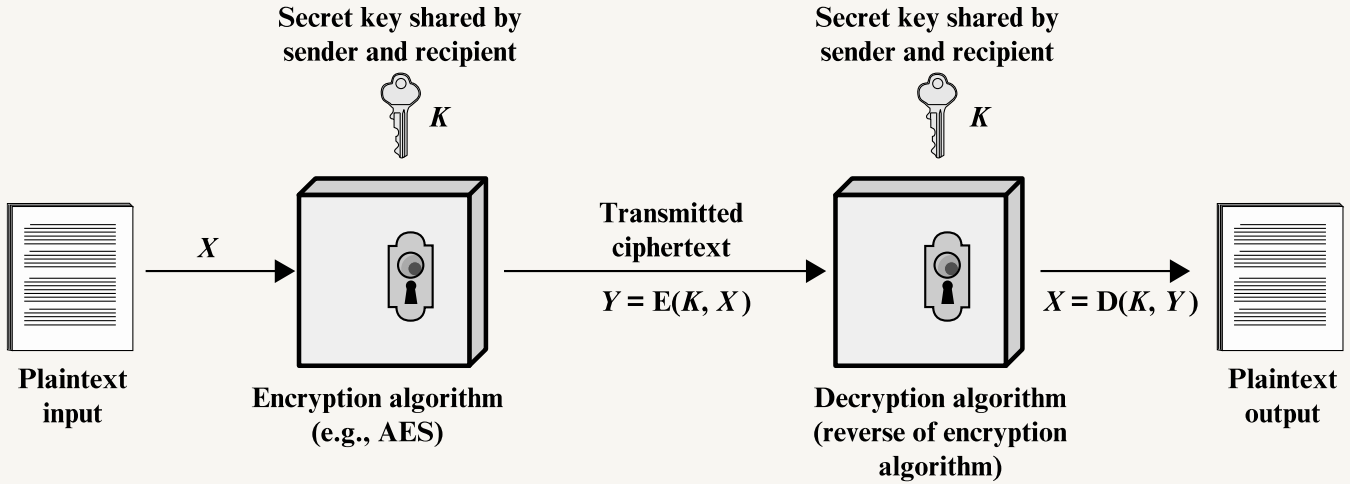


Probabilistic Algorithm

Mango: A96 ✗✗✓ {All should be True.}

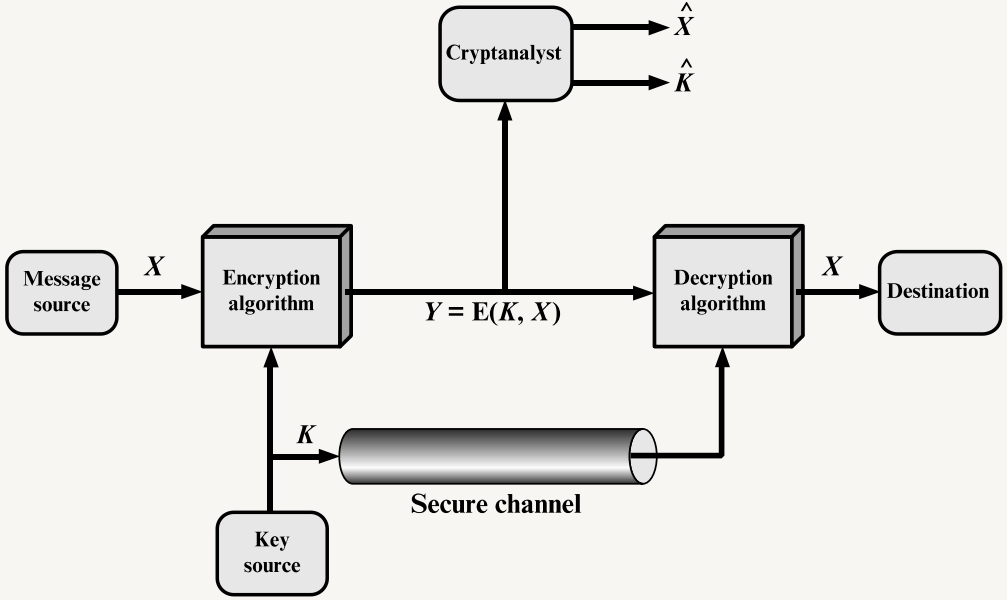
Grape: 6F3 ✓✓✓ {But there is no grape. False Positive.}

Symmetric encryption



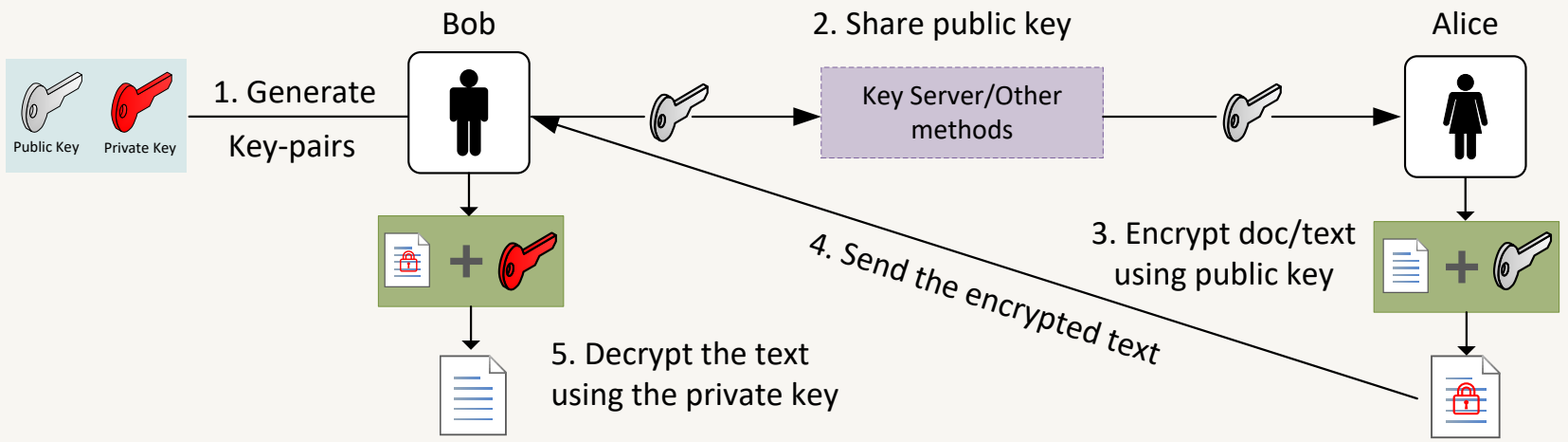
The main issue is key-management!

Symmetric encryption

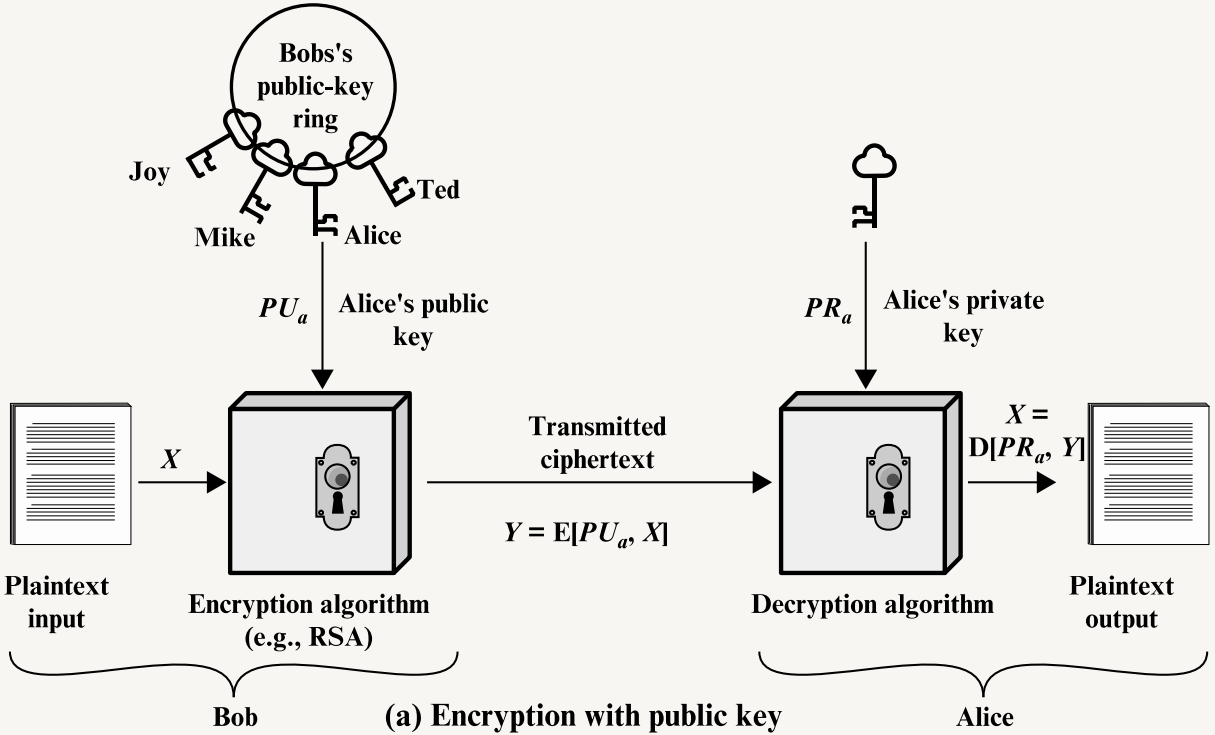


AES is currently the most widely used symmetric encryption

Public key cryptography: encryption



Public key cryptography: encryption



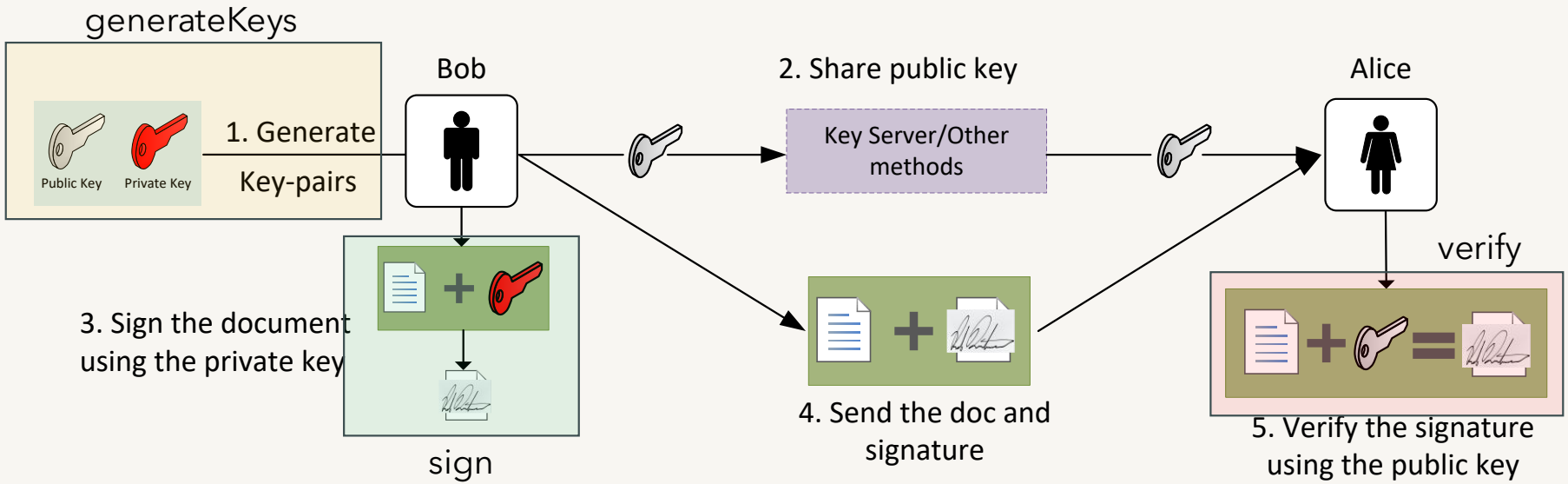
Public key cryptography: digital signature

- Digital signatures are based on asymmetric cryptography algorithms like RSA or ECC
- We need two properties of (analogue) signatures to hold in the digital world:
  - Only an entity is able to create a signature of its own, but everyone can verify it
  - This signature is tied to data that gets signed. A signature cannot be used for different data

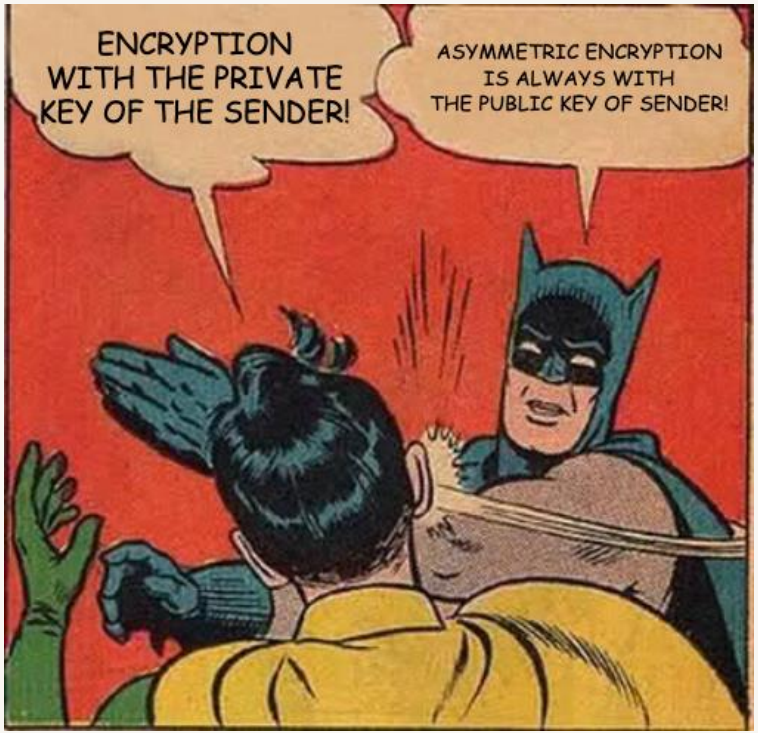
Digital signature: definition

- Three algorithms:
- (sk, pk) := generateKeys(keysize)
  - sk is the secret key and is used to sign messages. pk is the public key and is given to everyone. With the pk, they can verify the signature
- sig := sign(sk, message)
  - The sign method takes the message and the secret key, sk, as input and returns a signature for message under sk
- isValid := verify(pk, message, sig)
  - The verify method takes a message, a signature, and a public key as input
  - It will return true if the signature was generated out of the message and the secret key, otherwise false
- Such that verify(pk, message, sign(sk, message)) == true and signatures are unforgeable

Digital signature: definition



Remember this….



Digital signature algorithms

- Two major digital signature schemes are available
- RSA-based signature schemes, such as RSA-PSS
  - RSA signature was invented 1977 by Rivest, Shamir and Adleman
  - Based on the assumption that the factorisation of large prime number multiplicated is very hard
  - ECC-based signature schemes, such as ECDSA
  - Suggested independently by Neal Koblitz and Victor S. Miller in 1985
  - Based on discrete logarithms
- The BSI recommends following key sizes for asymmetric cryptography
  - RSA: min. 2048 Bit
  - ECDSA: min. 256 Bit
- Due to smaller key sizes in ECC, many blockchain systems use ECC

Cryptography resources

- Introduction to Cryptography: With Coding Theory, by Lawrence C. Washington and Wade Trappe
- Cryptography and Network Security: Principles and Practice, by William Stallings

Third Class: What is distributed system?

Distributed system model

- A distributed system consists of many computers (nodes)
- These nodes might be geographically located at different places, however, they are connected by a communication network
- All nodes are considered autonomous
  - They behave independently of each other
- They communicate with others using the network

Example : Torrent

Distributed system model

- Each node contains a processor, communication network, software and non-volatile storage
- Each processor within a node has volatile memory inaccessible by other nodes
- Each node has a network interface (NIC) through which it is connected to the network
- Software is mainly the OS
- The non-volatile is the storage used to store programs (other s/w) and data

Distributed system model

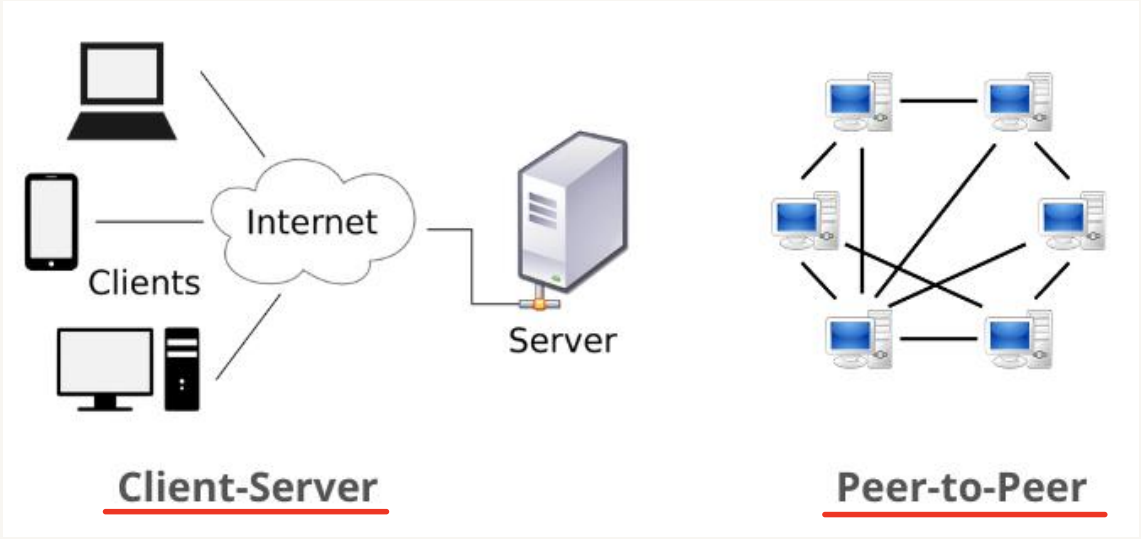
- Often a distributed system can be viewed as a logical construct, from an application viewpoint
- Such applications will be regarded as distributed applications
  - For this course, a distributed system = a distributed application
- A distributed application consists of concurrently executing processes
- A process is the execution of sequential program which is a list of of statements or instructions
- Concurrent processes can be executed in a single processor, each sharing the processor, the *multiprogramming* approach
- However, we are more interested about processes running in parallel in different nodes

Distributed system model

- There are three different types of concurrent processes: independent, competing and cooperating
- Concurrent processes are considered independent if the sets of objects accessed by them are disjoint
- Concurrent processes are considered competing if they share resources but they do not exchange information between them
- Cooperating processes exchange information between them either using shared data objects or via message passing

Distributed system model: network

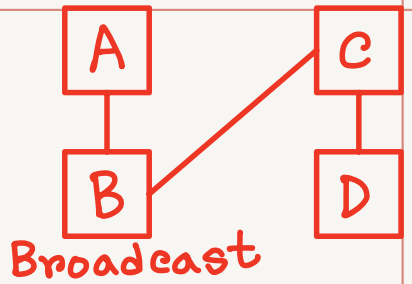
- P2P and client/server Network



https://www.networkstraining.com/wp-content/uploads/2021/06/Client-Server-p2p.png

Even though, we only show nodes here, there could be processes underneath

Network type



- Dwork et al.* categorised three types of networks exhibiting different properties: synchronous, asynchronous, and partially/eventually synchronous
- The latency involved in delivering a message to all nodes in a synchronous network is bound by some time denoted as Δ
  - Any message sent at time T will be delivered by T + Δ
- On the other hand, the latency in an asynchronous network cannot be reliably bound by any Δ, message will eventually be delivered

*C. Dwork, N. Lynch, and L. Stockmeyer "Consensus in the presence of partial synchrony". Journal of the ACM (JACM), 35(2):288323, 1988.

Sent time: T
Delivery time: T+Δ

Network type

- In a partially/eventually synchronous network, it is assumed that
  - the network will eventually act as a synchronous network
  - even though it might be asynchronous over some arbitrary period of time

Fault model

process node
synchronous

- In a distributed system, a node (process) might behave differently for various reasons (e.g. intentional or unintentional corruption)
- When this happens, we call these nodes as faulty nodes
- Up to f nodes out of N (total number) may fail
  - f is usually a function of N, like $f < N/2$ or $f < N/3$
- We mostly look at two types of faults:
  - Crash failure
  - Byzantine failure
- Nodes that do not fail are called “honest” or “correct” nodes

Fault model: crash failure

- The crash failure model deals with nodes that simply fail to respond due to some hardware or software failure
  - E.g. Hardware crash, hard disk bad sector, software crash, etc.
- It may happen any time without any prior warning
- The corresponding faulty node remains unresponsive until further actions are taken

Fault model: BGP {Byzantine General Problem}

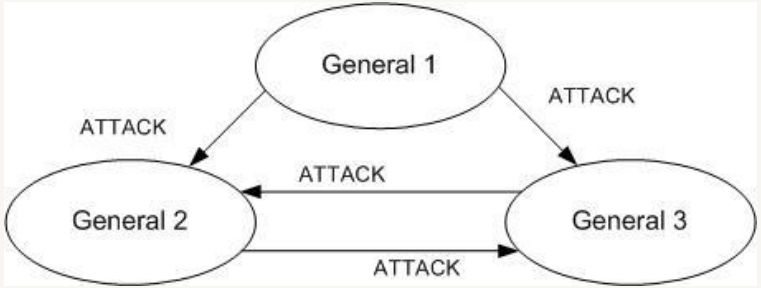
- The Byzantine Generals Problem, by Leslie Lamport, Robert Shostak, and Marshall Pease. ACM TOPLAS 1982
- Byzantine army divisions camped outside the walls of an enemy city
- Each division is led by a general
- Generals decide on a common plan of action

Fault model: byzantine fault

- There are two types of generals: Loyal or Traitor
- Conditions needed to be met:
  - Loyal generals decide upon the same plan of action
  - Small number of traitors should not be able to lead the loyal generals make a bad decision

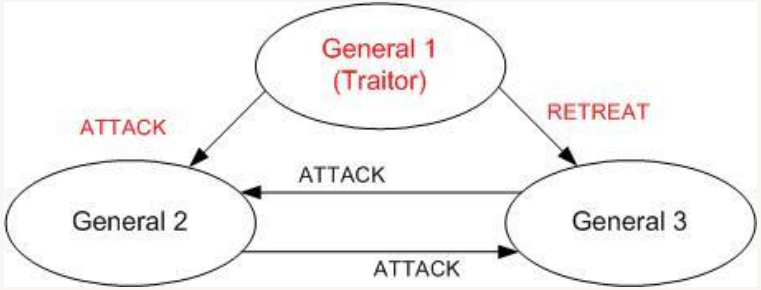
Fault model: byzantine fault

- General 2 receives ATTACK, ATTACK
- General 3 receives ATTACK, ATTACK
- So ATTACK is Not a Bad Decision



Fault model: byzantine fault

- General 2 receives ATTACK, ATTACK
- General 3 receives RETREAT, ATTACK
- Now, ATTACK or RETREAT?



The need for consensus algorithm

Fault model: byzantine fault

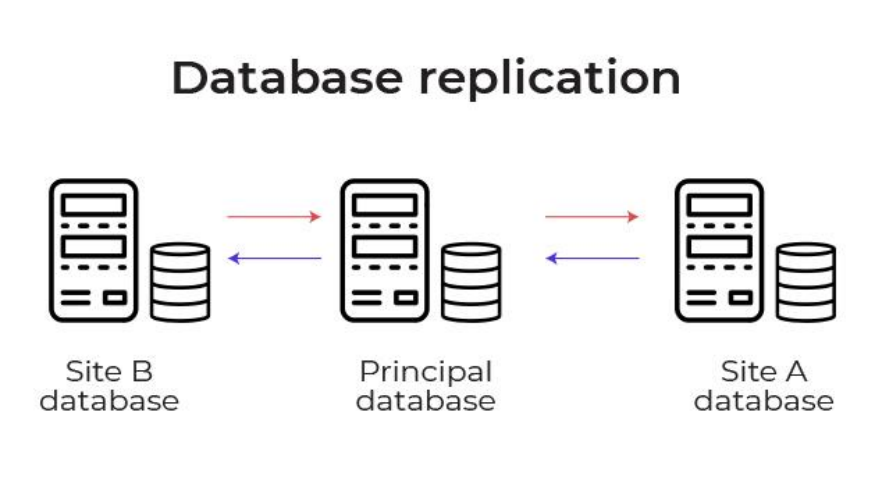
- Coping with failures in nodes not related to crash
- A (faulty) byzantine node sends conflicting information to different parts of system (the byzantine behavior)
  - Non-malicious: Software bugs
  - Malicious reasons: Machine compromised
- P2P Networks:
  - Faulty nodes generate corrupted and misleading messages
  - Good nodes have to “agree to do the same thing” (agreement)
- Agreement in the presence of faults is challenging

Fault model: byzantine failure

- Byzantine failure deals with nodes that misbehave due to some software bugs or because of the nodes being compromised by an adversary
- A Byzantine node can behave maliciously by arbitrarily sending deceptive messages to others
  - This might affect the security of distributed systems
- Hence, such nodes are mostly relevant in application with security implications

The need for consensus algorithm

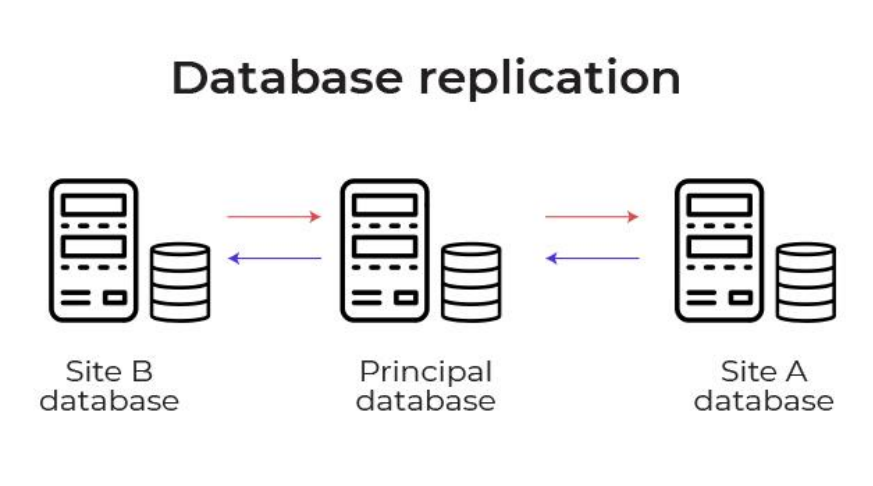
- Consensus is a fundamental problem in distributed applications
  - One use-case is database replication (aka Replicated Database)
- Database replication is the process of storing data in more than one site or node
- It is useful in ensuring resilience against node failures within a network
  - E.g. data are not lost when one or more nodes fail to function in an excepted fashion
  - This improves the availability of data



https://databand.ai/blog/data-replication-the-basics-risks-and-best-practices/

The need for consensus algorithm

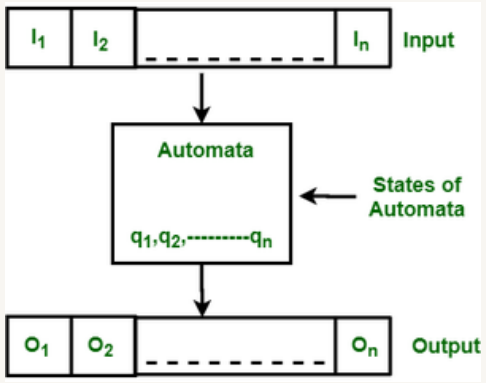
- It is simply copying data from one server to another server
  - So that all the users can share the same data without any inconsistency
- To ensure synchronisation across multiple nodes
  - The mechanism of consensus is used
- Consensus enables all nodes agree to a certain shared state/data among a set of distributed nodes



https://databand.ai/blog/data-replication-the-basics-risks-and-best-practices/

State machine

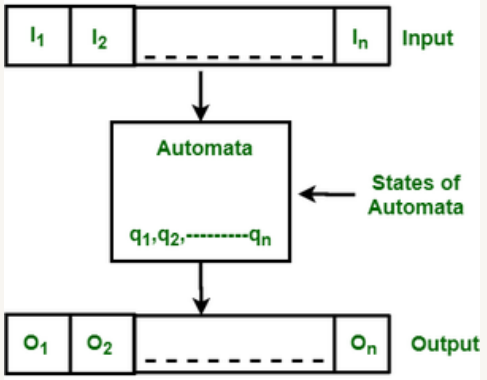
- The finite automata or finite state machine is an abstract machine
  - Simply, it is an abstract model of a digital computer
- It has a set of states
- It contains rules for moving from one state to another but it depends upon the applied input symbol
- It can be deterministic or non-deterministic



https://www.geeksforgeeks.org/introduction-of-finite-automata/

State machine

- A deterministic finite automata is a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$, consisting of
  - a finite set of states Q
  - a finite set of input symbols called the alphabet Σ
  - a transition function $\delta : Q \times \Sigma \rightarrow Q$
  - an initial or start state $q_0 \in Q$
  - a set of accept states $F \subseteq Q$



https://www.geeksforgeeks.org/introduction-of-finite-automata/

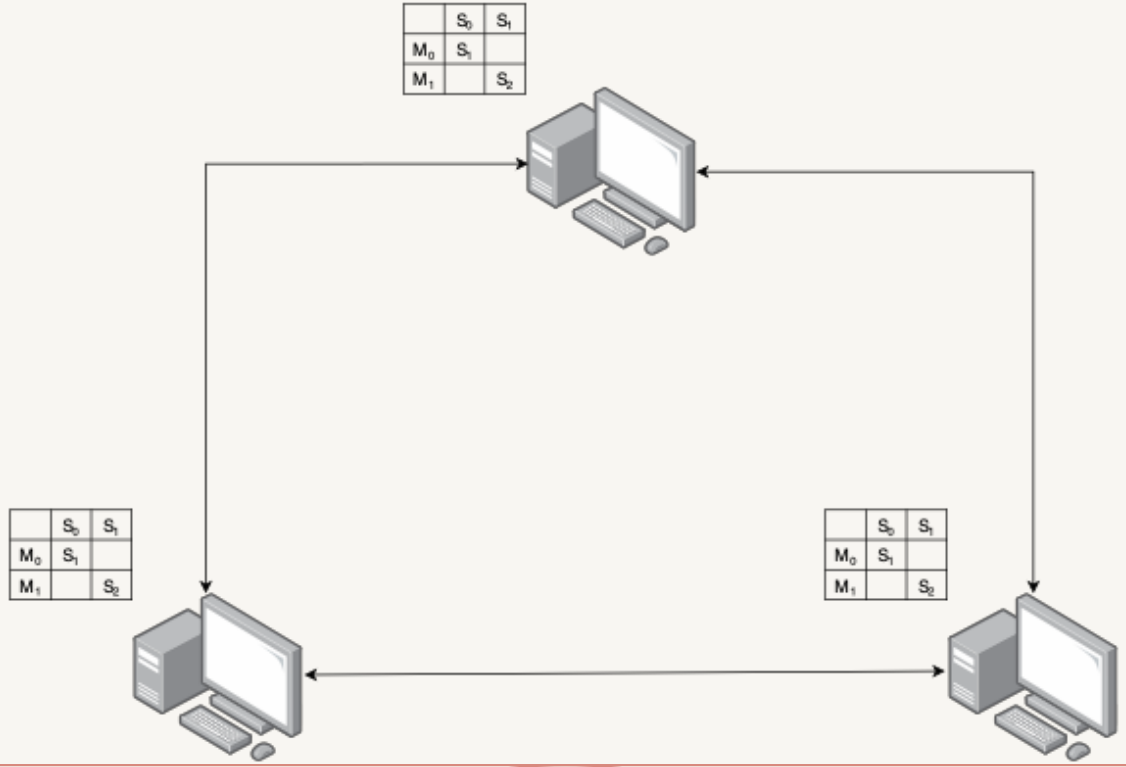
Atomic broadcast

- In fault-tolerant distributed computing, an atomic broadcast or total order broadcast is a broadcast where
  - all nodes receive the same set of input messages in the same order (i.e. the same sequence of messages)

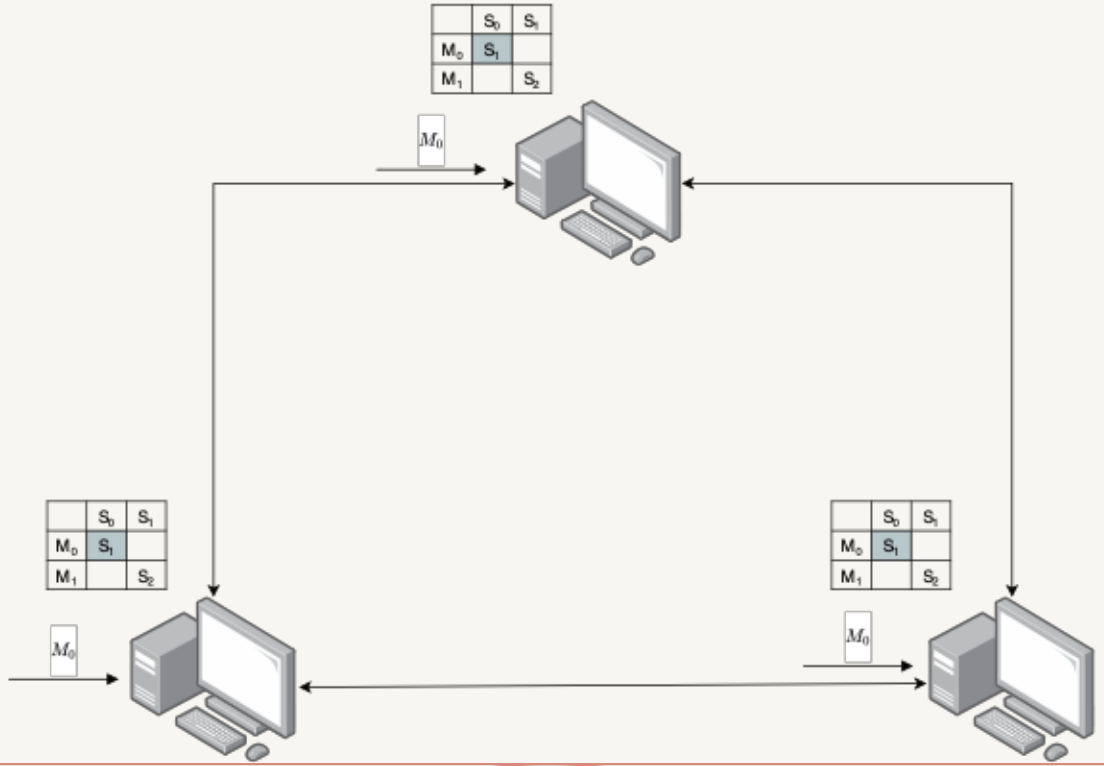
State machine replication

- The notion of the replicated database can be generalised with the concept of State Machine Replication (SMR)
- The core idea behind SMR is that a computing machine can be expressed as a deterministic state machine
- The machine accepts an input message, performs its predefined computation, and might produce an output/response
  - These actions essentially change its state
- SMR conceptualises that such a state machine, with an initial state, can be replicated among different nodes

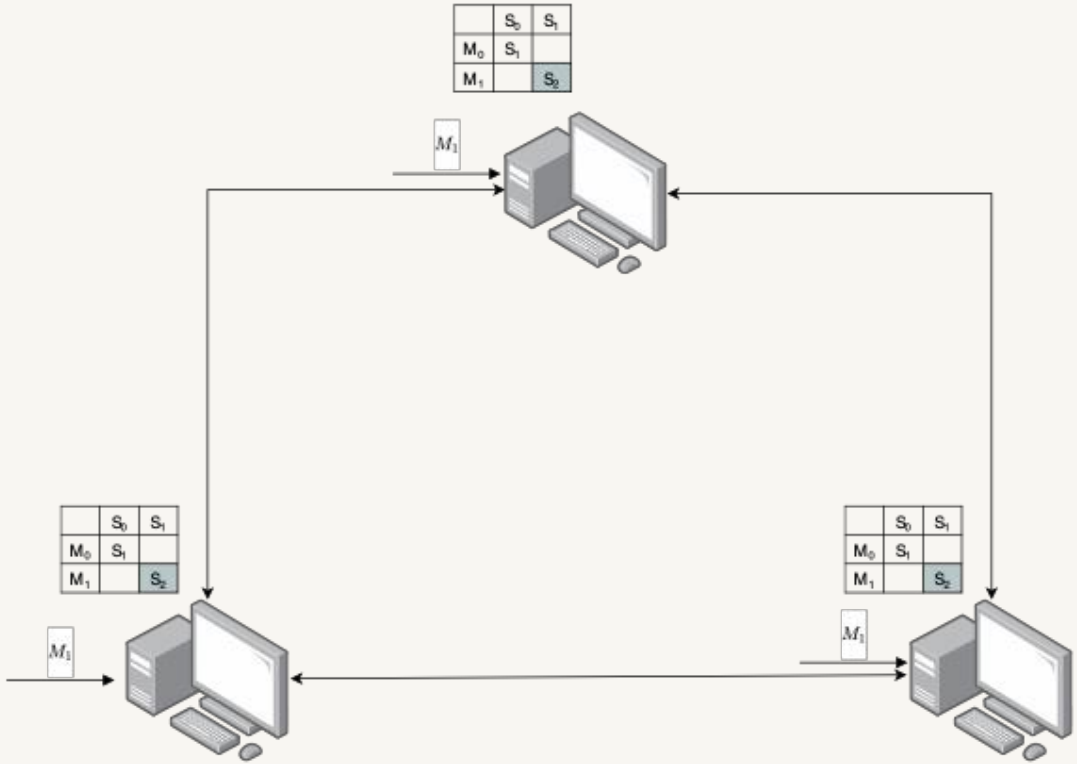
State machine replication



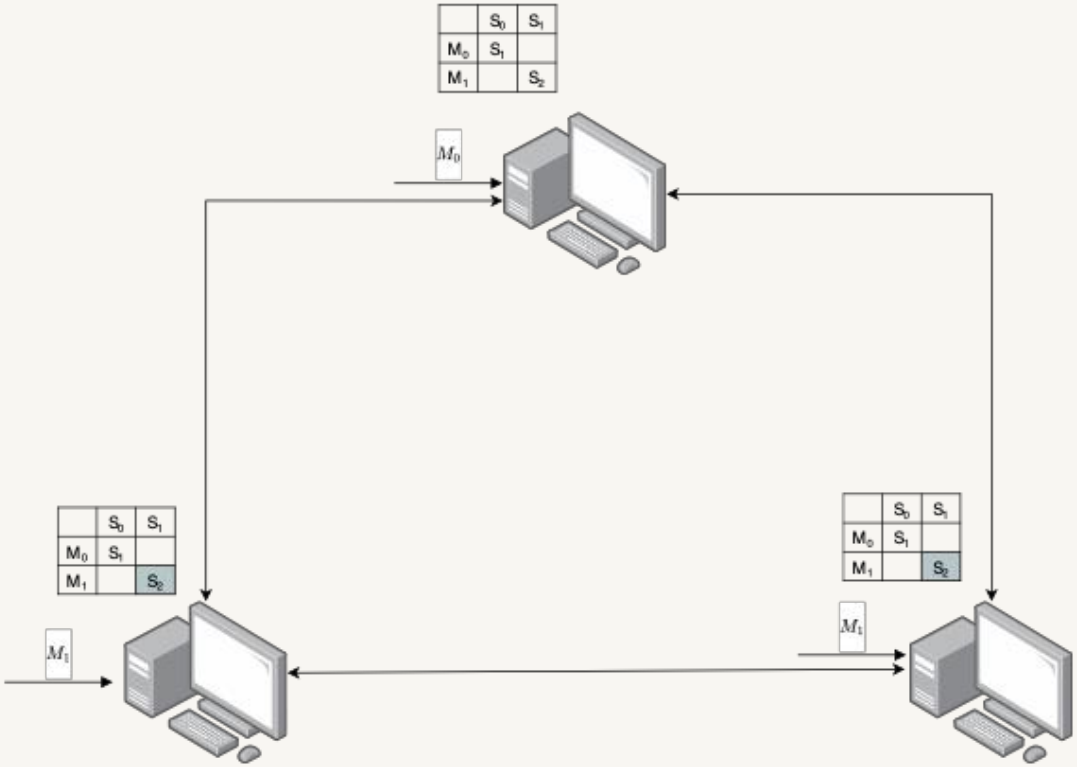
State machine replication



State machine replication



State machine replication



State machine replication

- If we can ensure an atomic broadcast of messages then each node would be able to evolve the states of its state machine individually in exactly the same fashion
  - All nodes behave similarly even though they are different
- This can guarantee consistency and availability regarding the state of the machine (as well as data it holds)
  - For example, if a node fails, other nodes can be used to recover its states or the data
- Once this occurs, it can be said that a distributed consensus has emerged among the participating nodes

State machine replication

- To make this happen, a protocol is required
- The protocol needs to ensure the timely dissemination and atomic broadcast of input messages among the nodes
- Such a protocol is called a consensus protocol