2. Cryptography

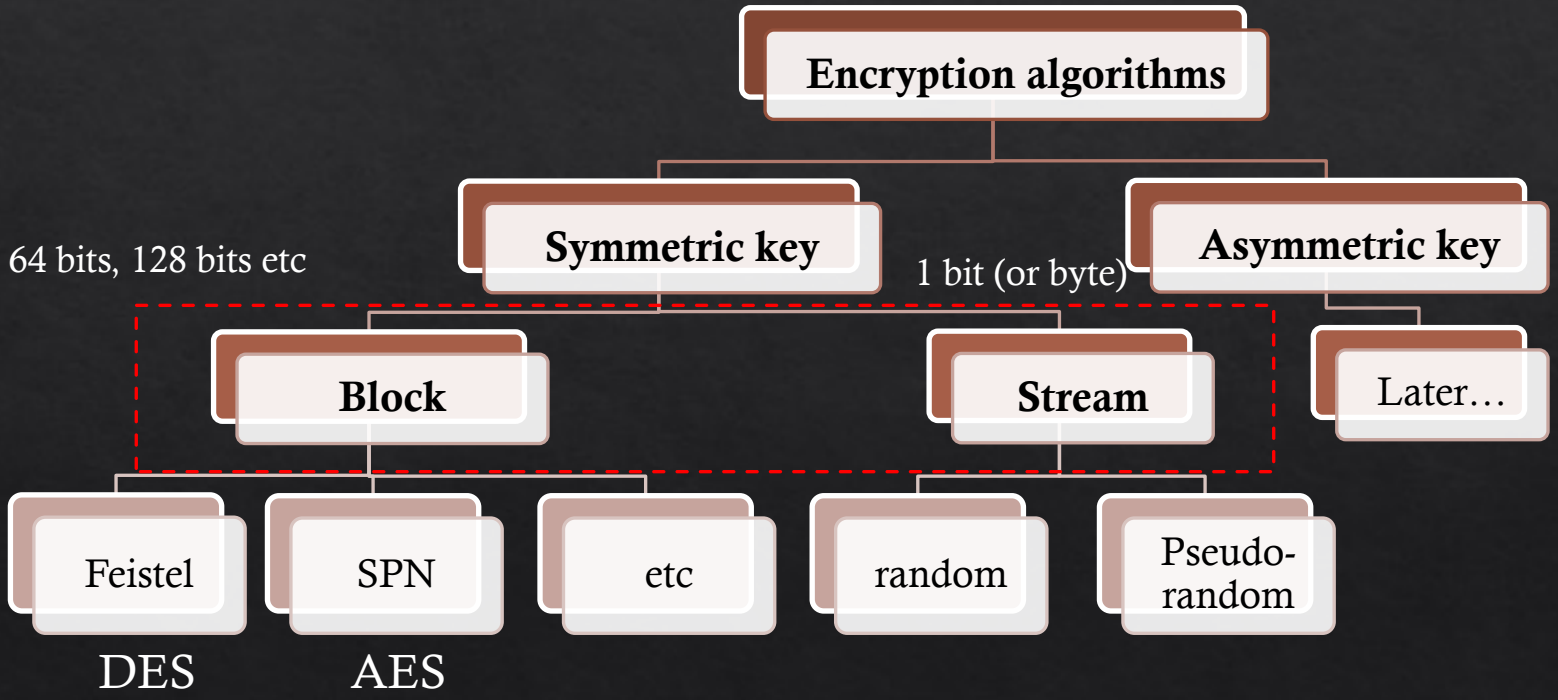
Jin Hong

jin.hong@uwa.edu.au

Cryptography: outline

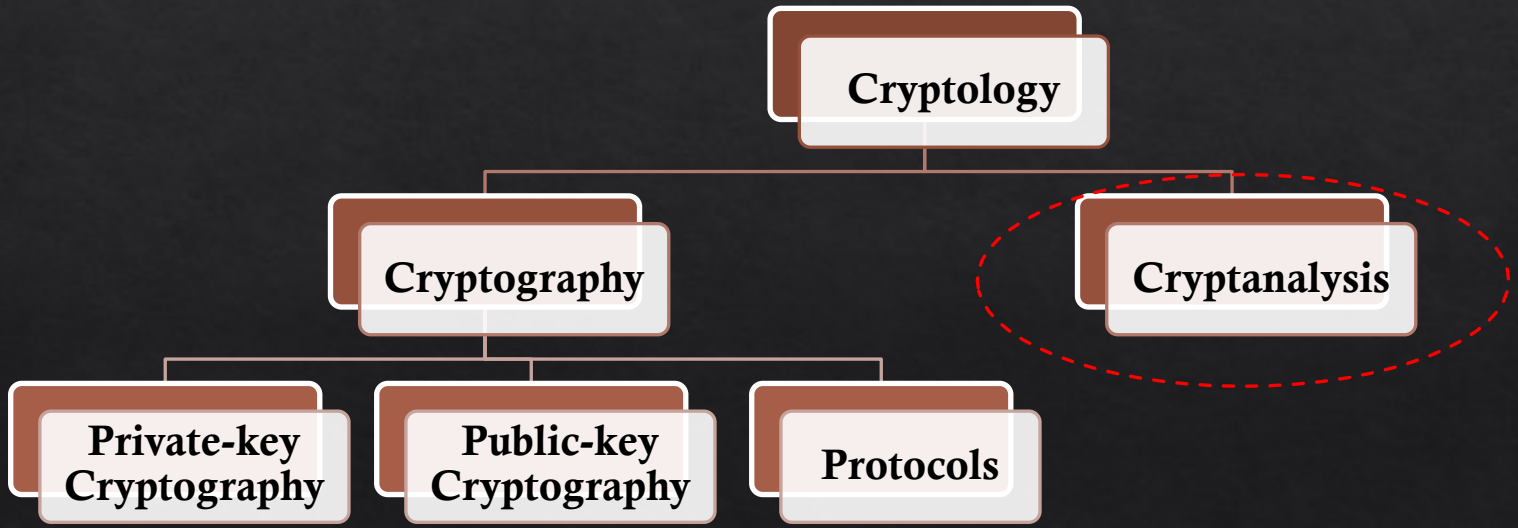
- Cryptology and Cryptography: A recap from CITS1003
- Ciphers
- Cryptanalysis

Classification of Encryption algorithms



Cryptology

- Cryptology is the study of cryptographic techniques and the analysis.



Source: Prof. Baktier's lecture at WPI

Cryptanalysis

- Objective to recover key not just message
- General approaches
  - Brute force attack
  - Cryptanalytic attack

Brute Force attack

- To try every possible way
- Simple to develop the procedure
- Success rate proportional to the key size
- Assume known or recognised plaintext

Key size (bits)	No. of keys	1 decryption/μs	10^6 decryptions/μs
32	$2^{32} = 4.3 \times 10^9$	35 minutes	2 ms
56	$2^{56} = 7.2 \times 10^{16}$	1142 years	10 hours
128	$2^{128} = 3.4 \times 10^{38}$	5.4×10^{24} years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	5.9×10^{36} years	5.9×10^{30} years
26 characters	$26! = 4 \times 10^{26}$	6.4×10^{12} years	6.4×10^6 years

Cryptanalytic Attacks

- **Ciphertext only**
  - Only knows the algorithm and ciphertext
- **Known plaintext**
  - Knows the plaintext & ciphertext
- **Chosen plaintext**
  - Select plaintext and obtain ciphertext
- **Chosen ciphertext**
  - Select ciphertext and obtain plaintext
- **Chosen text (combination of the above two)**
  - Select plaintext or ciphertext to encrypt/decrypt

Cryptanalytic Attacks

- **Ciphertext only**

Known to attacker	**$C_1, C_2, \ldots, C_n$**
Objective	1) $P_1, P_2, \ldots, P_n$
	2) Key K
	3) Algorithm C_{n+1} -> P_{n+1}

Ciphertexts generated using the same key

Find an algorithm that can decrypt any message encrypted using the key K

Cryptanalytic Attacks

- **Known plaintext**

Known to attacker	**$(P_1, C_1), (P_2, C_2), \ldots, (P_n, C_n),$**
Objective	1) Key K
	2) Algorithm C_{n+1} -> P_{n+1}

Attacker **cannot** select these pairs

Cryptanalytic Attacks

- **Chosen plaintext**

Known to attacker	**$(P_1, C_1), (P_2, C_2), \ldots, (P_n, C_n),$**
Objective	1) Key K
	2) Algorithm C_{n+1} -> P_{n+1}

Attacker **can** select the plaintext before attack begins, and **cannot** gain additional pair after the attack started.

Cryptanalytic Attacks

- **Chosen ciphertext**

Attackers **can** select ciphertexts before the attack begins

Known to attacker	**$(P_1, C_1), (P_2, C_2), \ldots, (P_n, C_n),$**
Objective	1) Key K
	2) Algorithm C_{n+1} -> P_{n+1}

This attack exploits public key algorithm, as attackers can generate ciphertexts using the public key

Result of attacks

- **Total break**
  - Found the **key**
- **Global deduction**
  - Did not find the key, but found the **algorithm**
- **Instance deduction**
  - Obtained **some plaintexts** from some ciphertexts
- **Information deduction**
  - Obtained **partial bits** of plaintext

Secureness of ciphers

- **Computationally secure**
  - **Cost** to break is higher than cost to encrypt
  - **Time** taken exceeds the useful lifetime
- **Provably secure**
  - The security mechanism can be **proven** to be equivalent to a hard problem
- **Unconditional security**
  - The attacker **cannot succeed** in cryptanalysing the algorithm given an **infinite** amount of resources
  - Only one-time pad fits into this category

Asymmetric Ciphers

Problems of symmetric key cryptosystem

- DES, AES are all symmetric key cryptosystem
  - i.e., using 1 key to encrypt and decrypt
- In a network, we have many more than 2 users
  - Given n users, how many different keys do we need to establish connected between every pair of users?

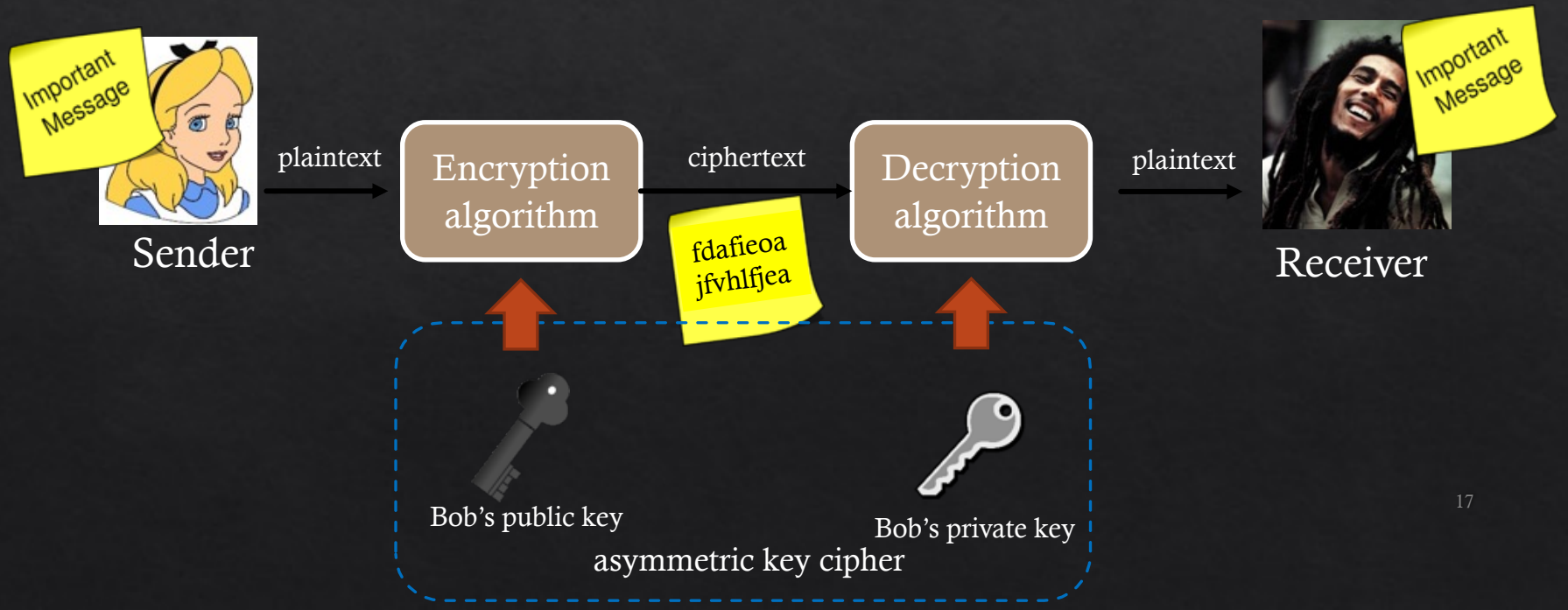
Need to generate $O(n^2)$ number of keys and manage them.

Public Key Cryptography

- In a public key cryptography system, **two different keys** are used – one publicly available and one secret.
- Others who wish to communicate with you can use your **public key** to encrypt their messages.
- Only you can decrypt to reveal the message using your **private key**.
- You only need to provide **1 key** to communicate with anyone!

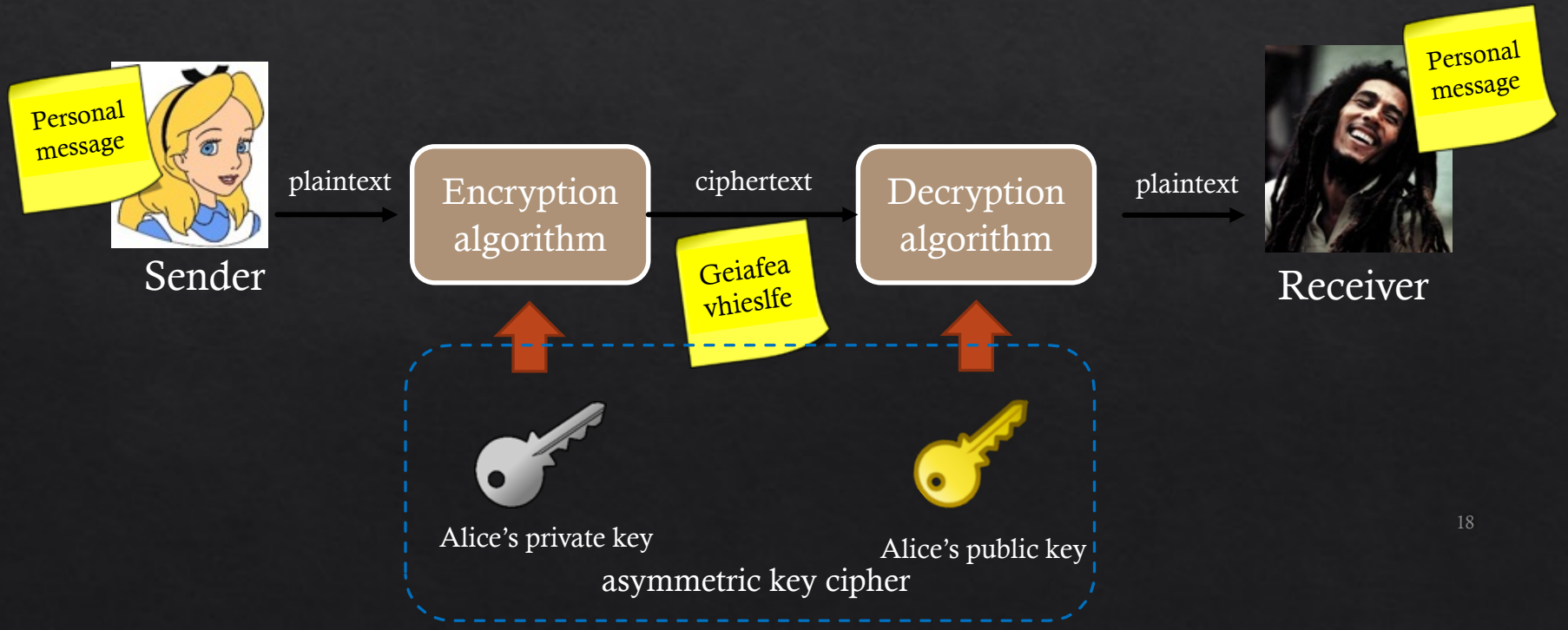
Public Key Crypto

Normal usage



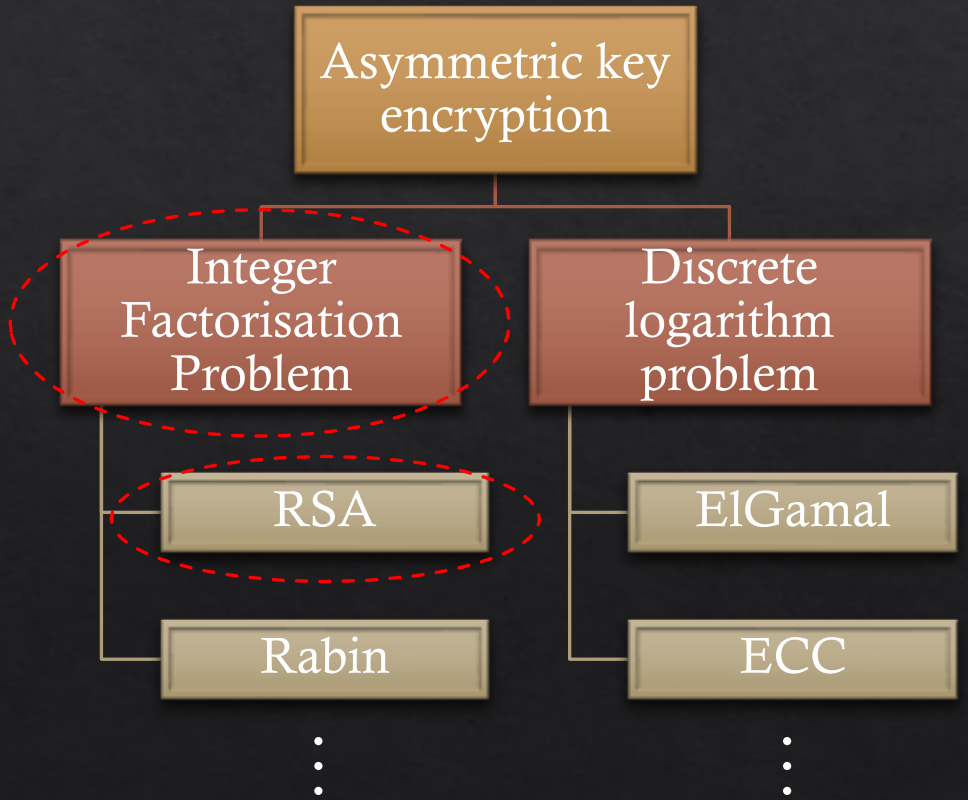
Public Key Crypto

Validating your identity



Asymmetric key ciphers

- Given two large primes p and q
- Compute $n = p \times q$
- Given n, how to get p and q?



Integer Factorisation

Given	Produce	Difficulty
Integers: p, q, r, s, t	Product: N	Polynomial
Integer: N	Factorise: p, q, r, s, t	Non Polynomial

Primes p, q **N = p*q**

Integer Factorisation

- Q: $11^{x} \bmod 13 = 9$, what is x?

RSA

- First public key cryptosystem
- World wide standard for the last 30 years
- RSA named after its creators
  - Ron **R**ivest, Adi **S**hamir and Leonard **A**dleman

Prerequisite for RSA

- Prime number
- GCD (greatest common divisor)
- Euclidean Algorithm
- Relatively prime
- Congruence
- Multiplicative inverse

Prime number

- It is a number that is only divisible by 1 and itself without any remainders

Greatest Common Divisor

- GCD is the largest whole number that can divide both *a* and *b* without a remainder
  - E.g., $45 = 3^2 \times 5$, $18 = 2^1 \times 3^2$
  - $GCD(45, 18) = 2^0 \times 3^2 \times 5^0 = 9$

Euclidean Algorithm

- Given gcd(a, b) = d
  - $a = b*q + r$ (b is smaller than a)
  - $gcd(a, b) \equiv gcd(b, r) = d$

Extended Euclidean Algorithm

- Next step, find x and y such that $a*x + b*y = d$
- We use this later to calculate the multiplicative inverse
- Example
  - $a = 204$, $b = 72$, find $204x + 72y = d$. First, calculate d
  - $204 = 72 * 2 + 60$
  - $72 = 60 * 1 + 12$
  - $60 = 12 * 5 + 0$
  - Therefore, gcd(204, 72) = 12 (the last integer with remainder 0)

Extended Euclidean Algorithm

- gcd(204, 72) = 12, now reverse calculate
- 12 = 72 – 60 * 1
- 12 = 72 – (204 - 72 * 2) * 1
- 12 = 72 * 3 – 204 * 1
- 12 = 204 * (-1) + 72 * (3)

collect terms together

- Hence, x = -1, y = 3

Extended Euclidean Algorithm

- Lets try one more -> GCD(89, 64)
  - 89 = 64*1 + 25
  - 64 = 25*2 + 14
  - 25 = 14*1 + 11
  - 14 = 11*1 + 3
  - 11 = 3*3 + 2
  - 3 = 2*1 + 1
  - 2 = 1*2 + 0
  - ∴ GCD(89, 64) = 1

Extended Euclidean Algorithm

- Now, trace back:
  - 1 = 3 – 2*1
  - 1 = 3 – (11 – 3*3)
  - 1 = (14 – 11*1)*4 – 11
  - 1 = 14*4 – (25 – 14*1)*5
  - 1 = (64 – 25*2)*9 – 25*5
  - 1 = 64*9 – (89 – 64*1)*23
  - ∴ 1 = 64*32 + 89*-23

collect terms together

= 3*4 – 11

= 14*4 – 11*5

= 14*9 – 25*5

= 64*9 – 25*23

= 64*32 – 89*23

89 = 64*1 + 25
64 = 25*2 + 14
25 = 14*1 + 11
14 = 11*1 + 3
11 = 3*3 + 2
3 = 2*1 + 1
∴ GCD(89, 64) = 1

Relatively Prime

- Two numbers *a* and *b* are **relatively prime**, if they have no common divisors > 1.
  - *a* = 8 divisors 1, 2, 4, 8
  - *b* = 15 divisors 1, 3, 5, 15
  - 8 and 15 are **relatively prime**

Congruence

- In modular arithmetic, congruence is
  - Having the same remainder when divided by a specific integer
  - Given $n > 0$, two integers *a* and *b* are congruent modulo *n*
  - a $\equiv b \bmod n$
    - *E.g., $10 \equiv 1 \bmod 3$ -> 10 and 1 are congruent modulo 3*
    - *E.g., $12 \equiv 2 \bmod 5$ -> 12 and 2 are congruent modulo 5*

Multiplicative Inverse

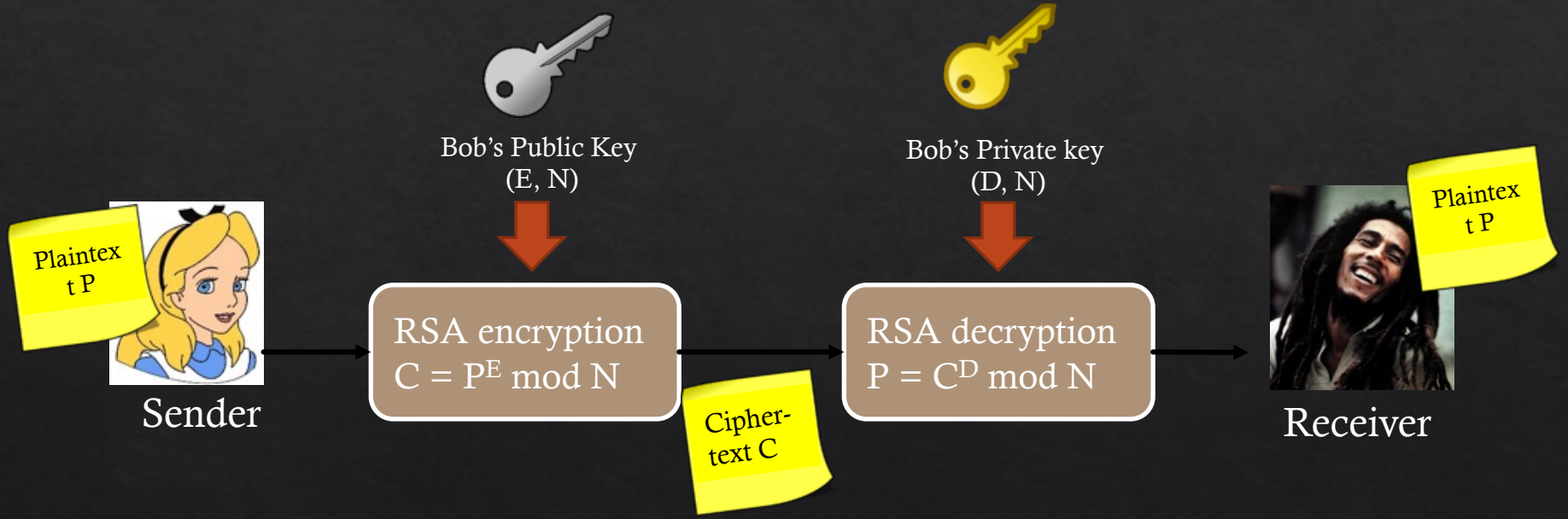
- Find x and y such that $x*y \equiv 1 \bmod n$
- If $n = 10$

1 x 1 = 1	2 x 1 = 2	3 x 1 = 3		9 x 1 = 9
1 x 2 = 2	2 x 2 = 4	3 x 2 = 6		9 x 2 = 8
1 x 3 = 3	2 x 3 = 6	3 x 3 = 9		9 x 3 = 7
1 x 4 = 4	2 x 4 = 8	3 x 4 = 2		9 x 4 = 6
1 x 5 = 5	2 x 5 = 0	3 x 5 = 5	. . .	9 x 5 = 5
1 x 6 = 6	2 x 6 = 2	3 x 6 = 8		9 x 6 = 4
1 x 7 = 7	2 x 7 = 4	3 x 7 = 1		9 x 7 = 3
1 x 8 = 8	2 x 8 = 6	3 x 8 = 4		9 x 8 = 2
1 x 9 = 9	2 x 9 = 8	3 x 9 = 7		9 x 9 = 1

Multiplicative Inverse

- For example, gcd(60, 37) = 1
- What is the multiplicative inverse of 37 mod 60?
  - What is x when 37 * x = 1 mod 60?
- Use the Extended Euclidean Algorithm

RSA: Encryption Decryption



Q: How to create those keys?

RSA: Key Generation

1. Select two large (> 1024 bits) primes p, q
2. Compute modulus $n = p*q$ and
3. Compute $\phi(n) = (p-1)*(q-1)$
   - ϕ is Euler's Totient Function
4. Pick an integer e relatively prime to $\phi(n)$
   - $\gcd(e, \phi(n)) = 1$; $1 < e < \phi(n)$
5. Compute d such that $e*d = 1 \bmod \phi(n)$

- Public key (e, n)
- Private key (d, n)

RSA example

- Key Generation
  1. $p = 5$, $q = 11$
  2. $n = pq = 55$
  3. $\phi(n) = (p-1)*(q-1) = 4 * 10 = 40$
  4. e = 13 (gcd(40, 13) = 1)
     - e = {3, 7, 9, 11, 13, 17, 19, 21, 23, 27, 29, 31, 33, 37, 39}
  5. Compute d, e*d = 1 mod $\phi(n)$, 13*d = 1 mod 40, d = 37

Public Key = {e, n} = {13, 55}

Private Key = {d, n} = {37, 55}

RSA example

- Bob wants to send Alice a secret message: 13 08 16
- Bob uses Alice's public key to calculate the following
  - $13^{13} \bmod 55 = 08$
  - $08^{13} \bmod 55 = 28$
  - $16^{13} \bmod 55 = 26$
- Bob sends 08 28 26

Public Key = {e, n} = {13, 55}

Private Key = {d, n} = {37, 55}

RSA example

- Alice receives 08 28 26
- Alice uses her private key, d = 37, to decrypt message
- $08^{37} \bmod 55 = 13$
- $28^{37} \bmod 55 = 08$
- $26^{37} \bmod 55 = 16$

Public Key = {e, n} = {13, 55}

Private Key = {d, n} = {37, 55}

Is RSA quick?

- Unfortunately, **no**.
- RSA requires many computations to carry out
  - Mostly the power computation

1. Use modular operations to reduce intermediate results
   - $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$
2. Use fast exponent computational algorithms
   - See MATH1721 – Mathematics Foundations

RSA Challenge

- https://en.wikipedia.org/wiki/RSA_Factoring_Challenge
- Different length of subprime numbers
- Challenges are no longer valid
  - ended in 2007
- RSA-829 is the largest subprime solved to date (2020)
- Remember, RSA uses p and q both **1024** bits as basic mode, up to **2048** bits

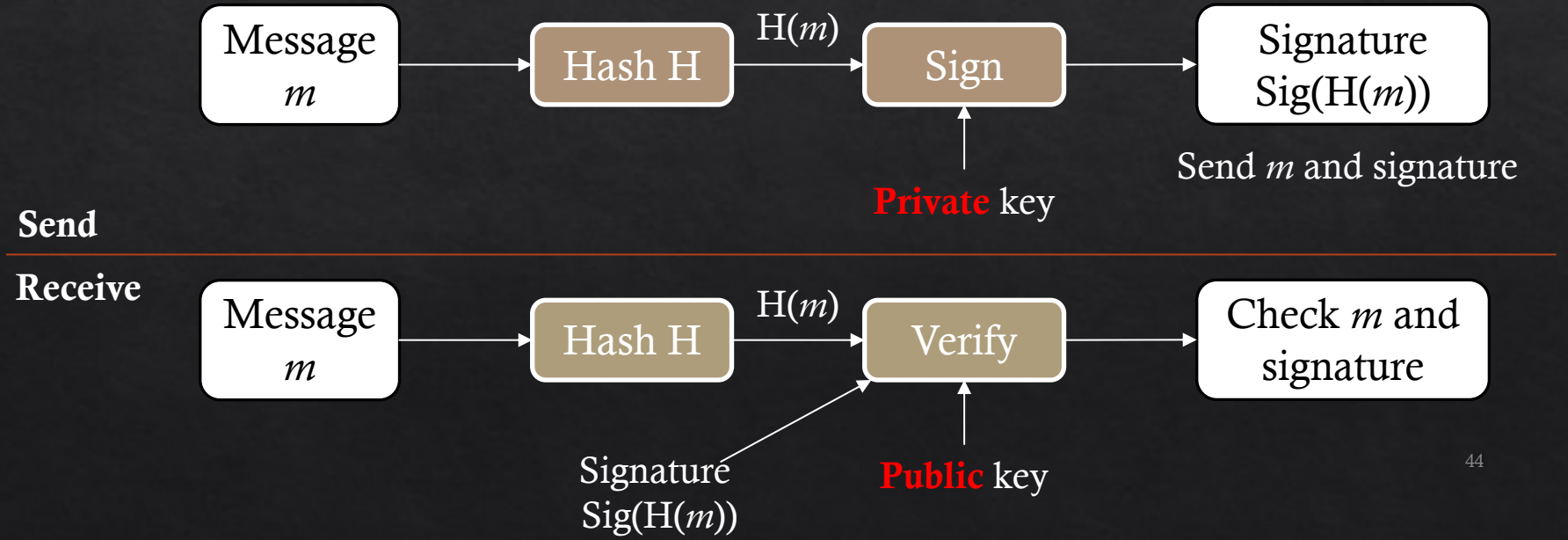
Question so far?

Hashing

- Also known as
  - Hash functions, message digest, one-way transformation etc.
- Generate a message of fixed length given an arbitrary length of the message
  - Typically 128 or 160 bits
- Examples include
  - MD5 (message digest) – 128 bits output
  - SHA-2 (secure hashing algorithm) 256/224, 512/384 bits

Hashing

- The primary goal/application is to generate and verify digital signature



Applications of Hashing

- Password hashing
  - Do not store the raw password
  - Store H(password + salt) and salt
  - Salt makes dictionary attack more difficult
- Message integrity using message authentication code (MAC)
  - Agree on a secret key k
  - Compute H(m|k) and send with m
  - Does not require encryption algorithm
- File integrity (cyber forensics)
- Generating random number

Signing using RSA

- Let us use RSA to sign the message
- Alice first gets the hash of the message **h(*m*)**
- Then sign using her private key d such that
  - s = $(h(m))^d \bmod n$
- Alice sends the encrypted message and the signature to Bob
- The receiver, Bob, can check the authenticity of the message by comparing the message ***m*** and $s^e \bmod n$

d is Alice's private key
e is Alice's public key

Signing using RSA

Using the RSA example 2

Public Key = {e, n} = {3, 55}

Private Key = {d, n} = {27, 55}

- Message m is {06, 14, 11}, C is {51, 49, 11}
- Hash function $H(m) = \sum(3m_i + 1) \bmod 11$
- Then, our hash value is: 19+43+34 mod 11 ≡ **8**
- Sign the Hash: $s = \mathbf{8}^{27} \bmod 55 = 2$
- Alice sends C and **s**: ({51, 49, 11}, 2)
- Bob can decrypt the message, calculate **hash** of the message, and decrypt the **sign** using Alice's public key to check the authenticity
- $H(m) = 2^3 \bmod 55 = \mathbf{8}$
- If the original message is changed (or the hash), then they will **not** match.

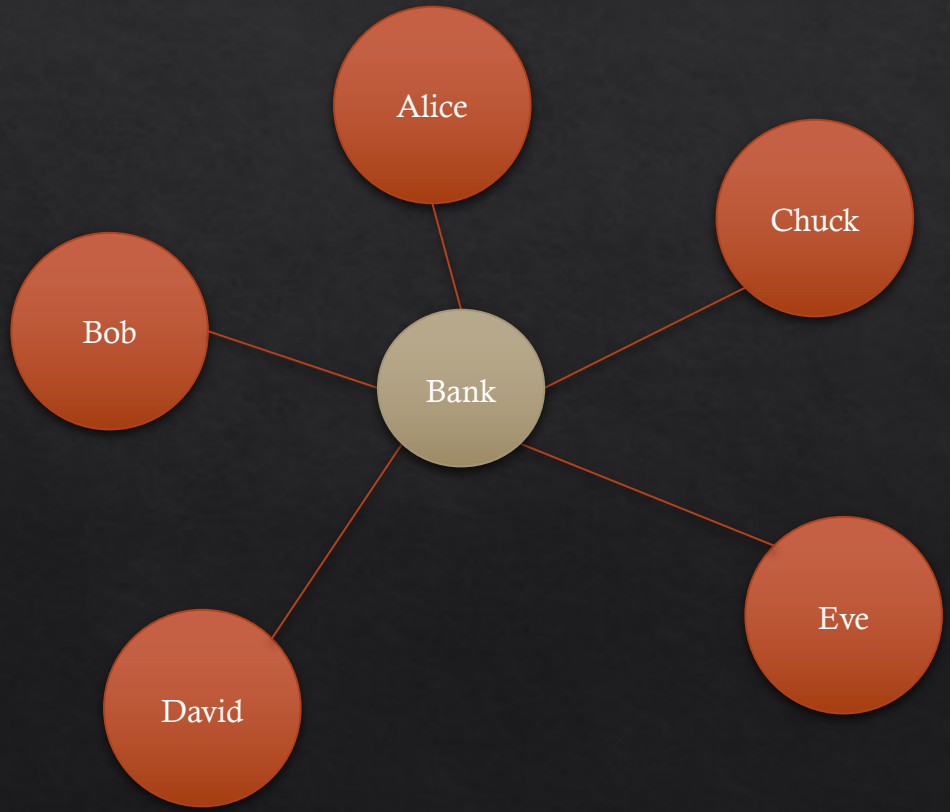
Blockchain

- What is blockchain?

Blockchain

- Why distributed system?

Blockchain

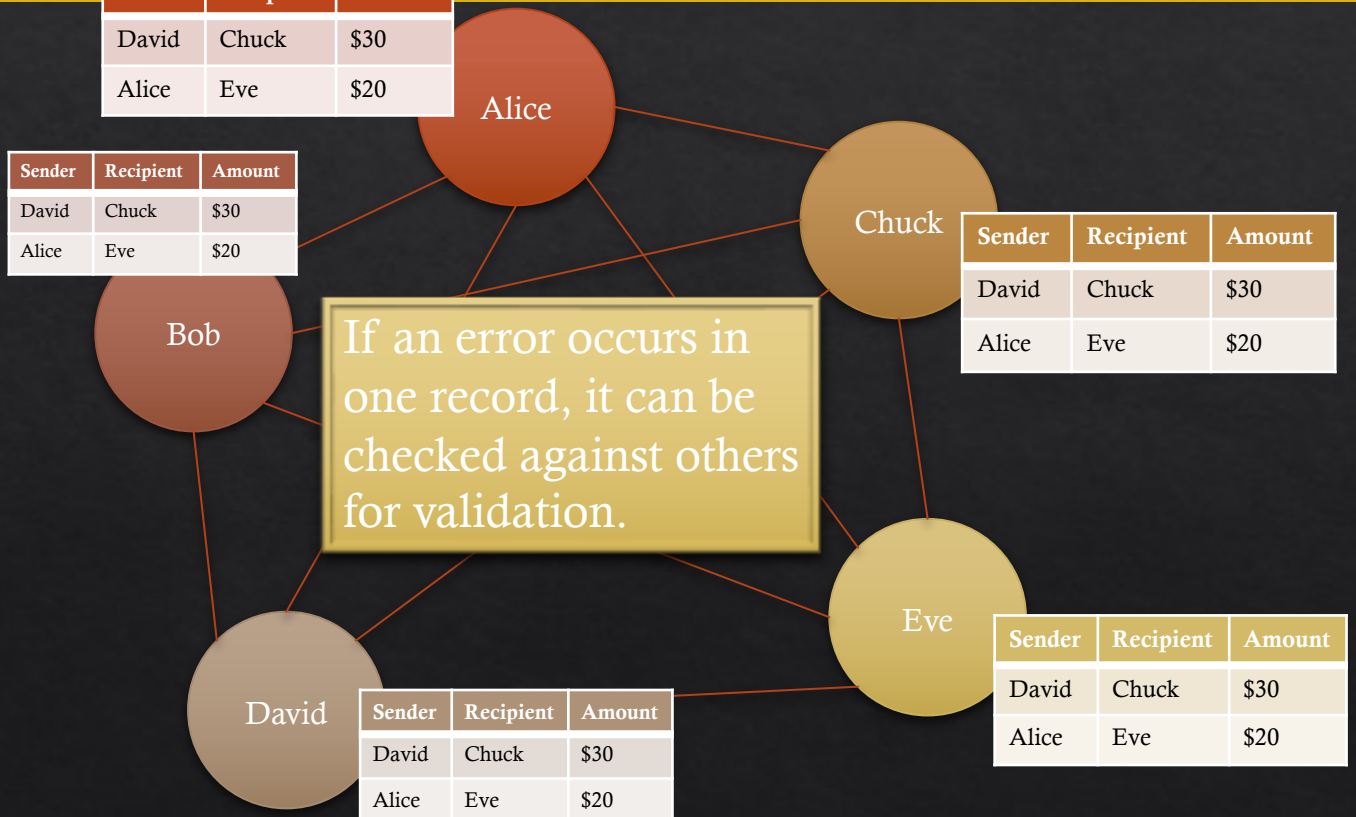


Traditionally, we have a single entity (e.g., Bank) to keep track of the transactions

Sender	Recipient	Amount
David	Chuck	$30
Alice	Eve	$20

Blockchain

Sender	Recipient	Amount
David	Chuck	$30
Alice	Eve	$20



Sender	Recipient	Amount
David	Chuck	$30
Alice	Eve	$20

Sender	Recipient	Amount
David	Chuck	$30
Alice	Eve	$20

Sender	Recipient	Amount
David	Chuck	$30
Alice	Eve	$20

Sender	Recipient	Amount
David	Chuck	$30
Alice	Eve	$20

In Blockchain, everyone keeps the record of transactions

Blockchain - details

- When a transaction happens, **everyone** in the network records the transaction
- Each node records a list of transactions – called ***Block***
- The next block is chained from the previous block hash

Blockchain - details

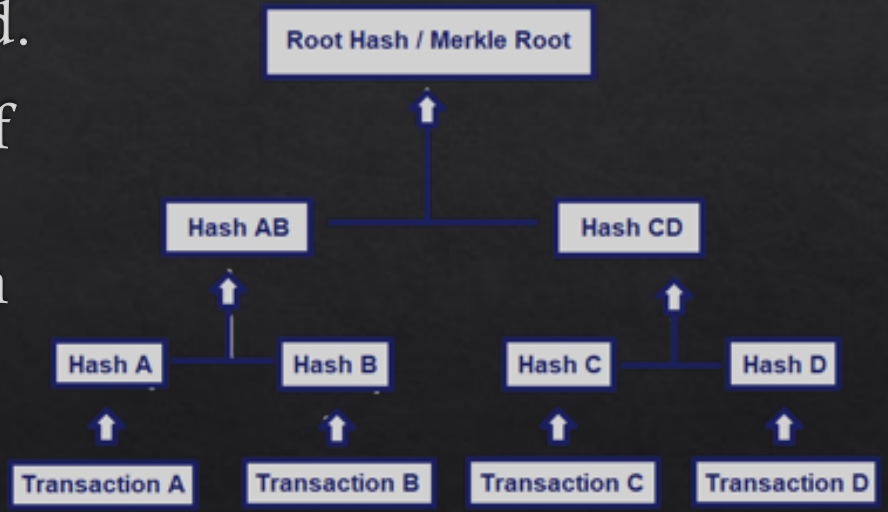
Block 1

Transaction 1
Transaction 2
Transaction 3
…
Transaction n

Hash of Block 1: xxxxxxxxxx….

Blockchain - details

- To calculate the hash value of the block, **Merkle tree** is used.
- Merkle tree is simply a tree of **hashed values** of each transaction computed bottom up

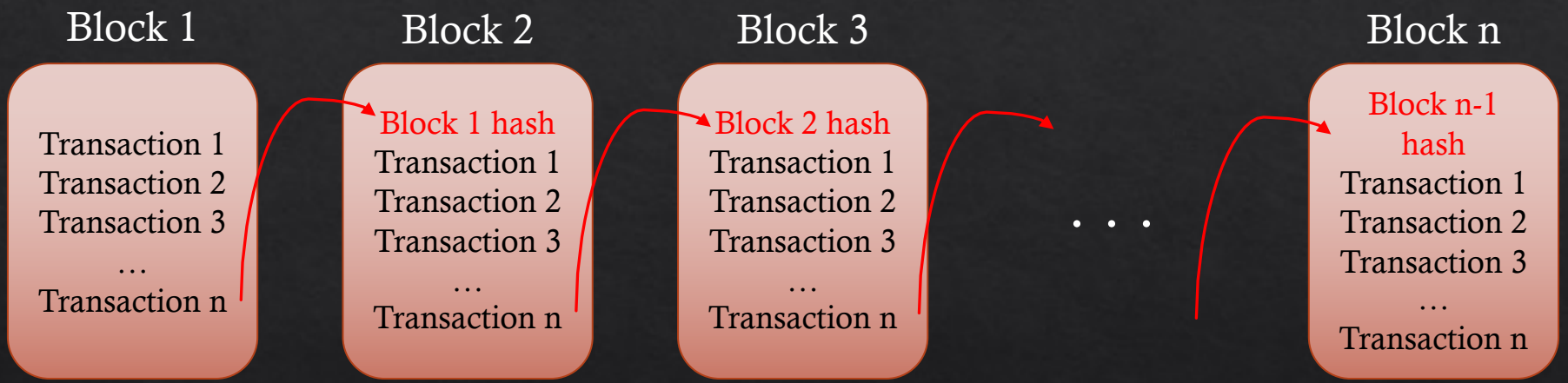


Blockchain - details

- Merkle tree allows you to:
  - **Summarise** all the transactions with a digital fingerprint
  - Quick/simple test to verify a transaction is **included** or not
  - Each branch can be downloaded **on its own** for verification
  - Provide **integrity** and **validity** of data

- What do you do if you have a odd number of transactions?
  - Repeat the last hash value twice

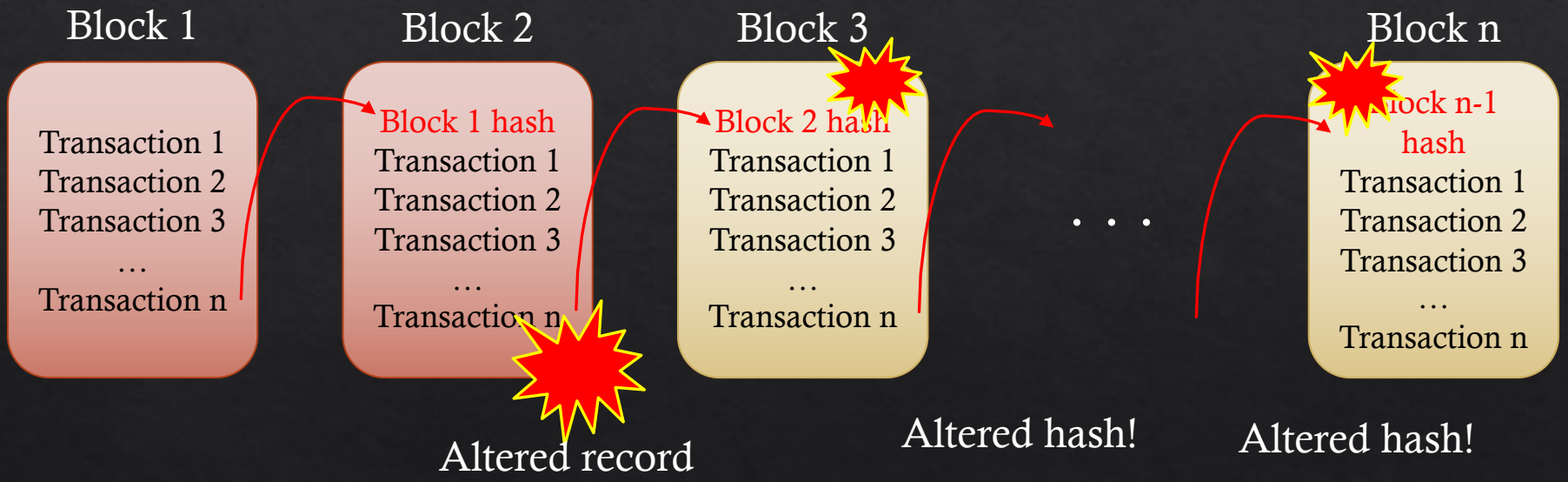
Blockchain - details



Blockchain - details

- Blockchain is just a sequence of hash-chained records
  - These records are immutable, stored in distribution
- When a transaction occurs, the transaction is recorded to peer blocks (can be an arbitrary number) using the hash-chain
  - That is, you cannot modify the transaction

Blockchain - details



All nodes have the same blockchain, so you can easily find the error

Using Blockchain

- Underlying technology for **Cryptocurrency**
  - E.g. a Bitcoin wallet address: 3NT1wrYVYYsPorK1jq9UVLVbzbroSYUaP7
    - Addresses are used only **ONCE** – why?
  - Due to its properties described previously
  - Satoshi Nakamoto created Bitcoin, and also proposing Blockchain (used to be Block ‘space’ chain)
    - Satoshi Nakamoto is an unknown identify

Bitcoin white paper: https://bitcoin.org/bitcoin.pdf

Cryptocurrency

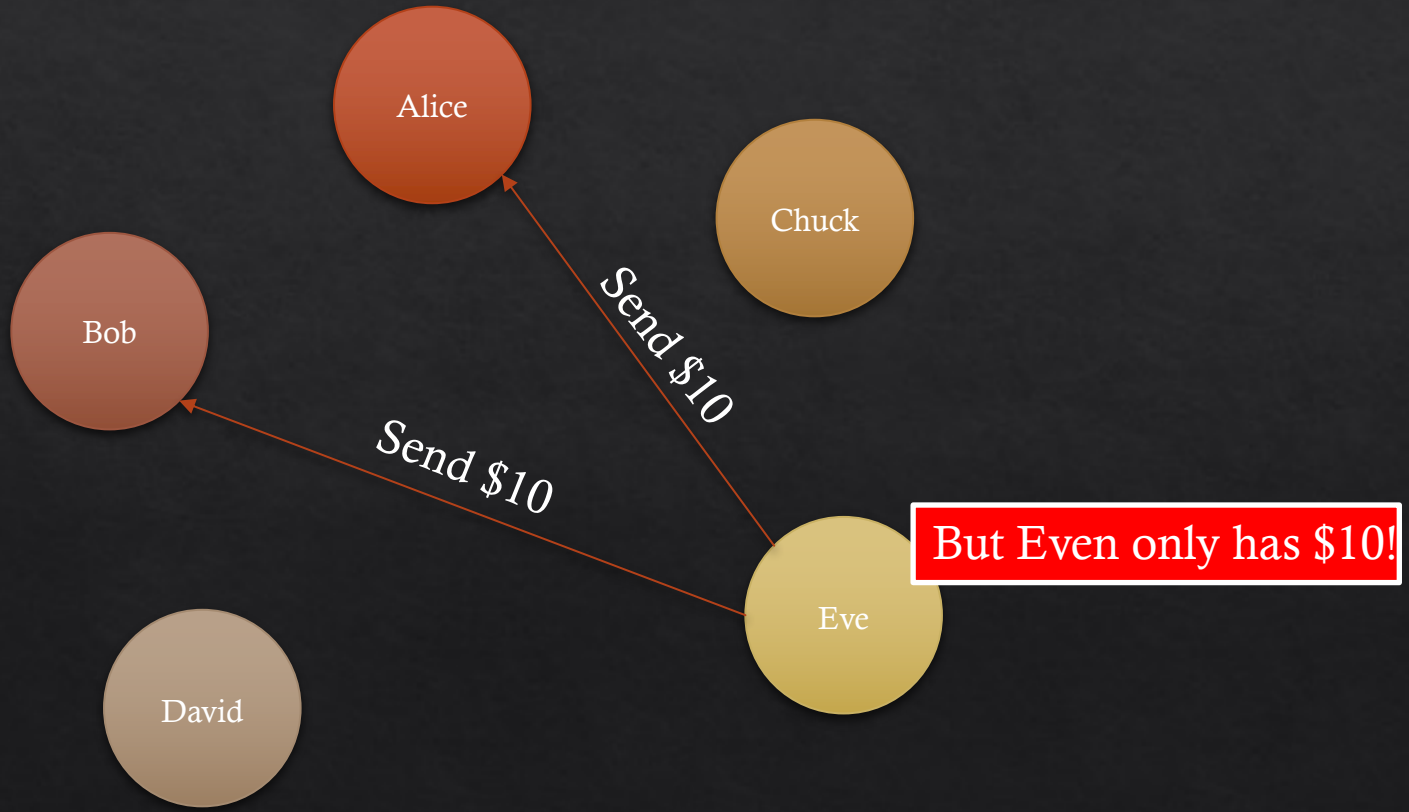
- Cryptocurrency is generated through **mining**
  - This is the only way to release new cryptocurrency
  - E.g. we have 19.6M bitcoins in circulation as in 2024*
- Mining is done by **validating the block**
  - i.e. calculating that hash value for the next block
- To earn cryptocurrency, you have to be the first one to mine the block, as well as be part of the longest link
  - I will show you what it means soon

*https://blockworks.co/price/btc

Cryptocurrency

- Using Blockchain for cryptocurrency has some important issues that needs to be manages
  - What could go **wrong**?

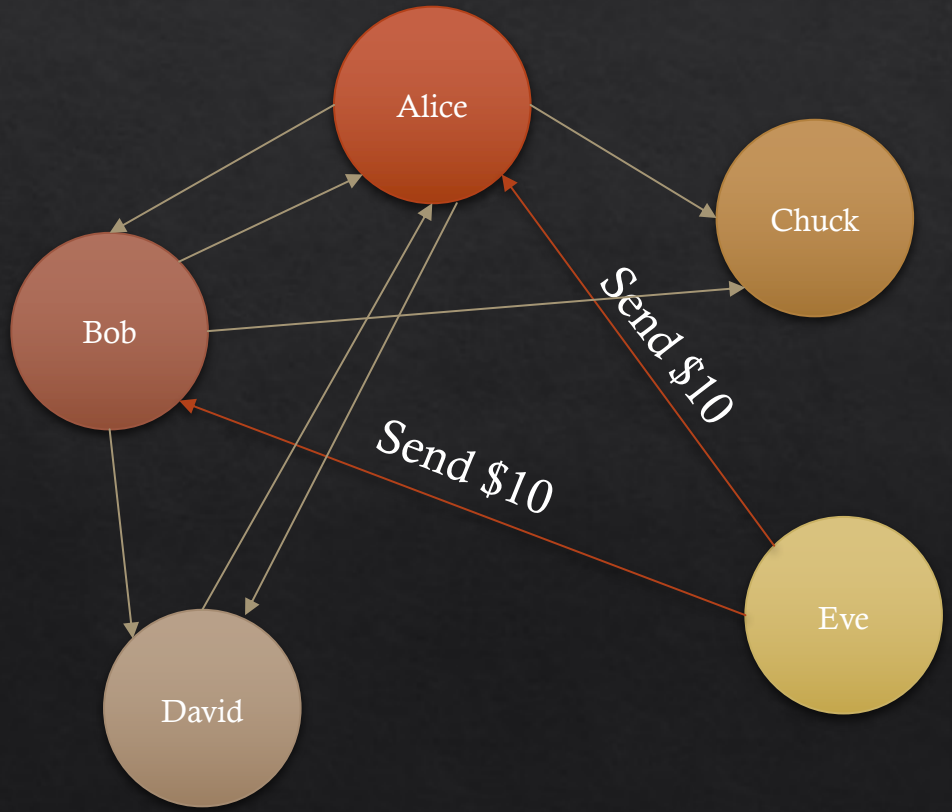
Cryptocurrency



Cryptocurrency

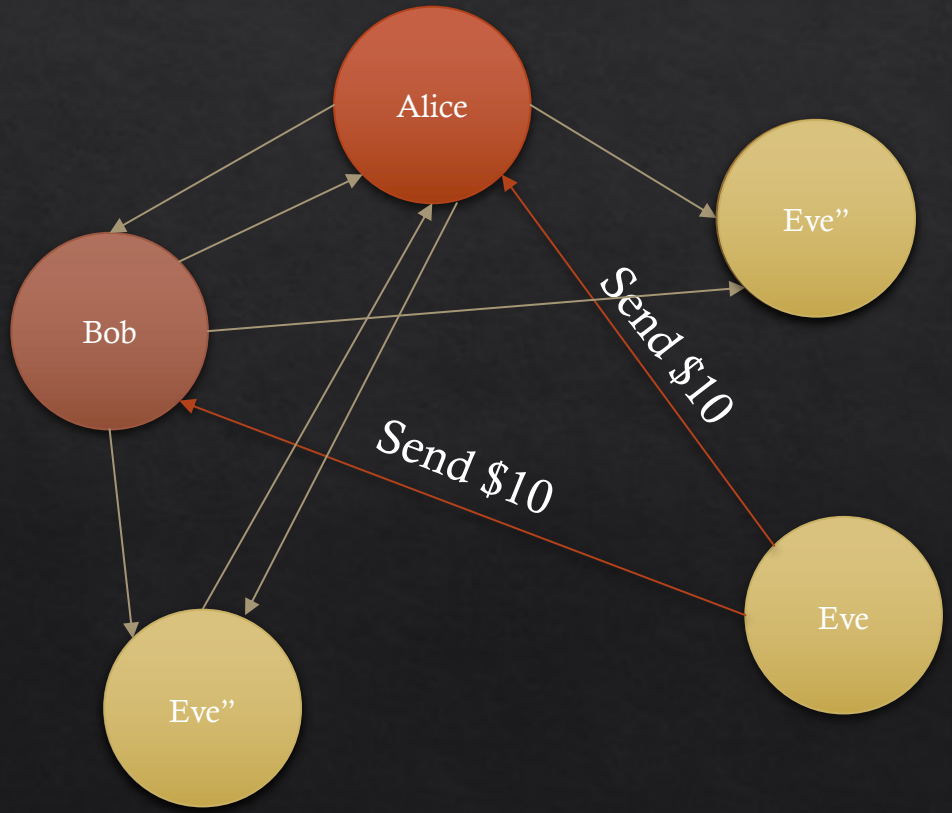
- When a transaction happens, all the nodes will vote.
- The transaction that has the majority of votes will be recorded in the Blockchain

Cryptocurrency



Vote for transaction

Cryptocurrency



Cannot trust every node in the network! This is a Sybil attack

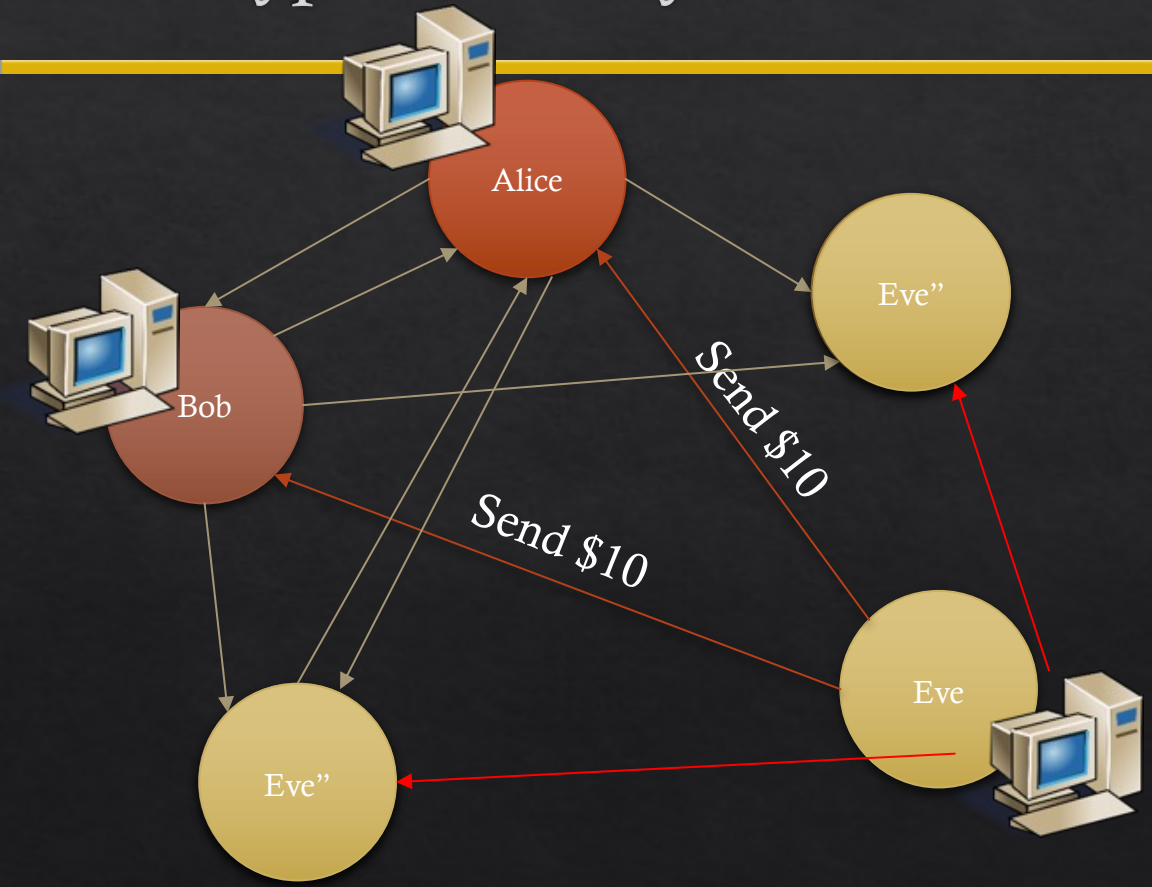
Cryptocurrency

- To overcome this problem, **evidence** is provided to cast a vote
  - Named Proof-of-Work
- Each node will perform some brute force task, which spends resources
  - What is the brute force task?
    - E.g., Certain property of the hash value to be produced

Cryptocurrency

- Brute force task example:
  - Given a hash function f(m) = 31 * m * c + c + 7 mod 79
  - Given message m = 29
  - Find a nonce value c such that f(m) is a single digit

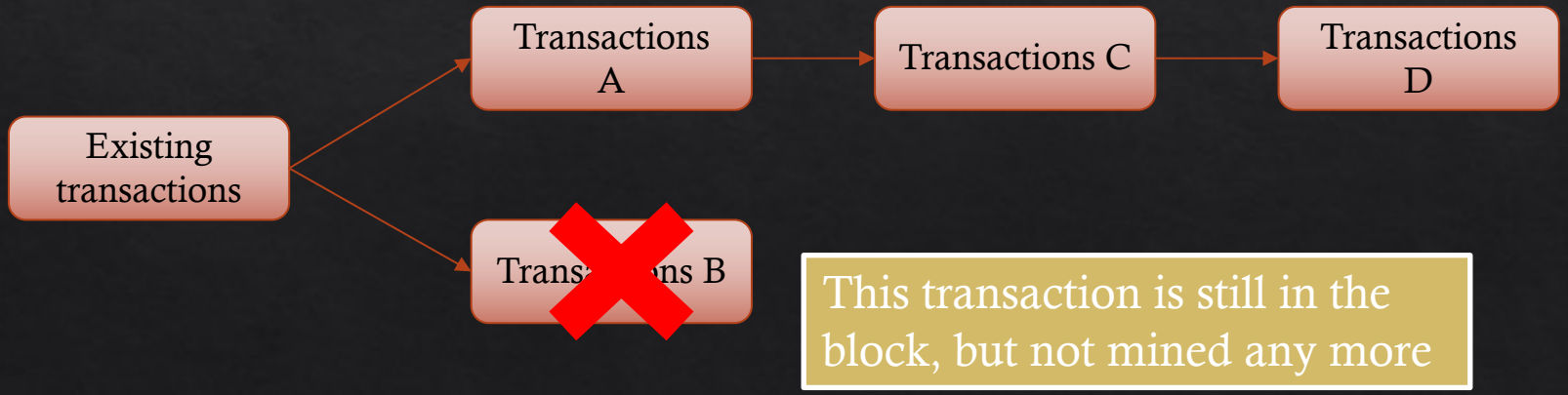
Cryptocurrency



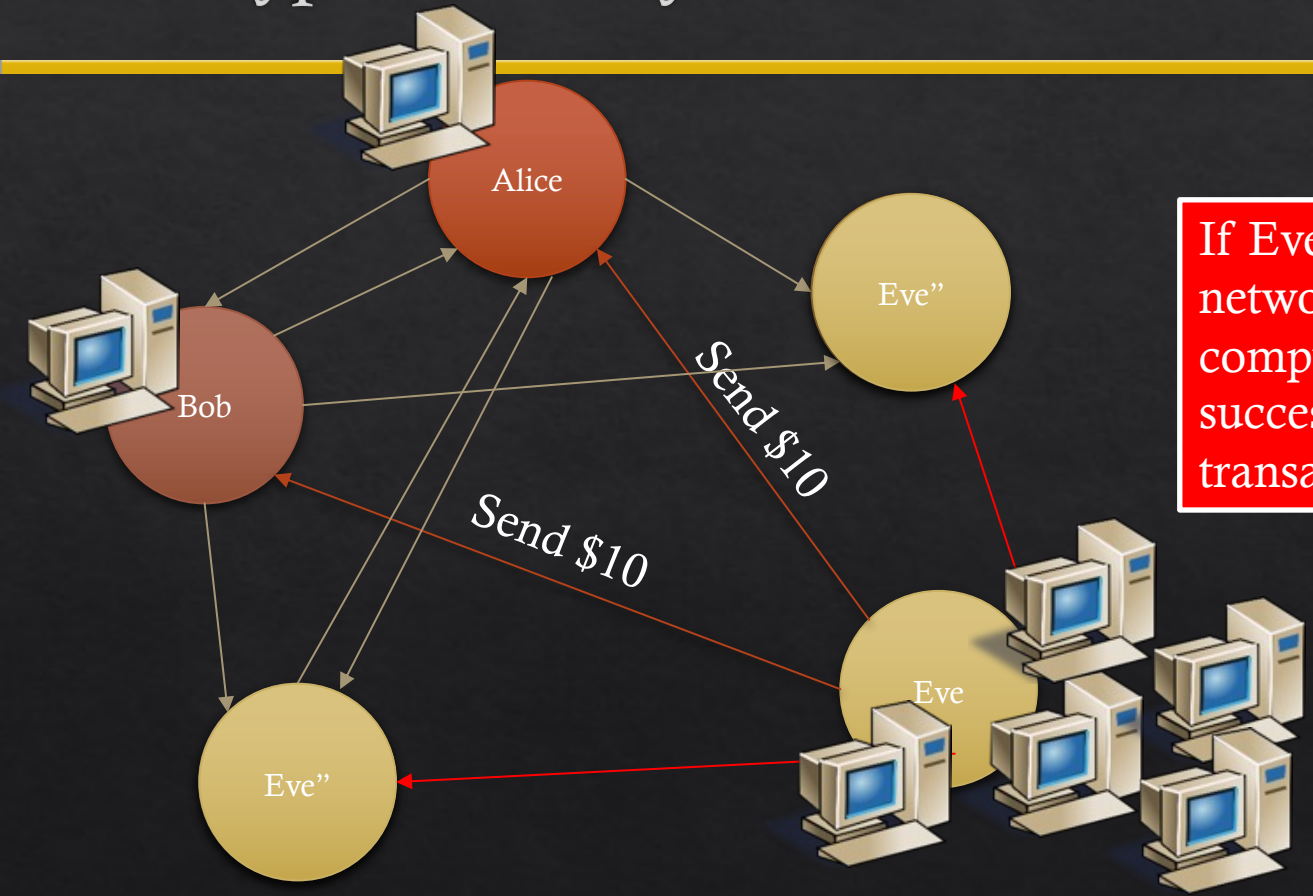
Eve only has one computer to perform 3 Brute force tasks in order to vote!

Cryptocurrency

- What happens to the invalid votes? Or different votes at the same time?
- This scenario is called forks



Cryptocurrency



If Eve can afford > 51% of the network nodes (or equivalent computing resources), Eve can successfully manipulate transactions

Cryptocurrency

- And there are other issues we need to consider
  1. Forks that can be caused by sybil nodes or selfish mining
  2. Selfish mining: miners trying to increase their rewards by keeping blocks private
  3. DNS attacks: sending peers wrong information
  4. Mempool attacks: flooding new blocks with transactions
  5. DDoS attacks: achieving denial of service
  6. Consensus delay: preventing peers from reaching consensus
  7. Theft of wallets – probably one of the biggest issues now

Other uses of Blockchain

- Smart contracts
- Crowdfunding
- Governance
- Auditing
- File Storage
- Intellectual Property
- IoT
- Identity management
- Anti-money laundering
- Etc…

https://blockgeeks.com/guides/what-is-blockchain-technology/

Summary

- Ciphers – Symmetric and Asymmetric
- Hashing
- Blockchain

- Supplementary:
  - Diffie-Hellman
  - ECC

Next Week

- Privacy

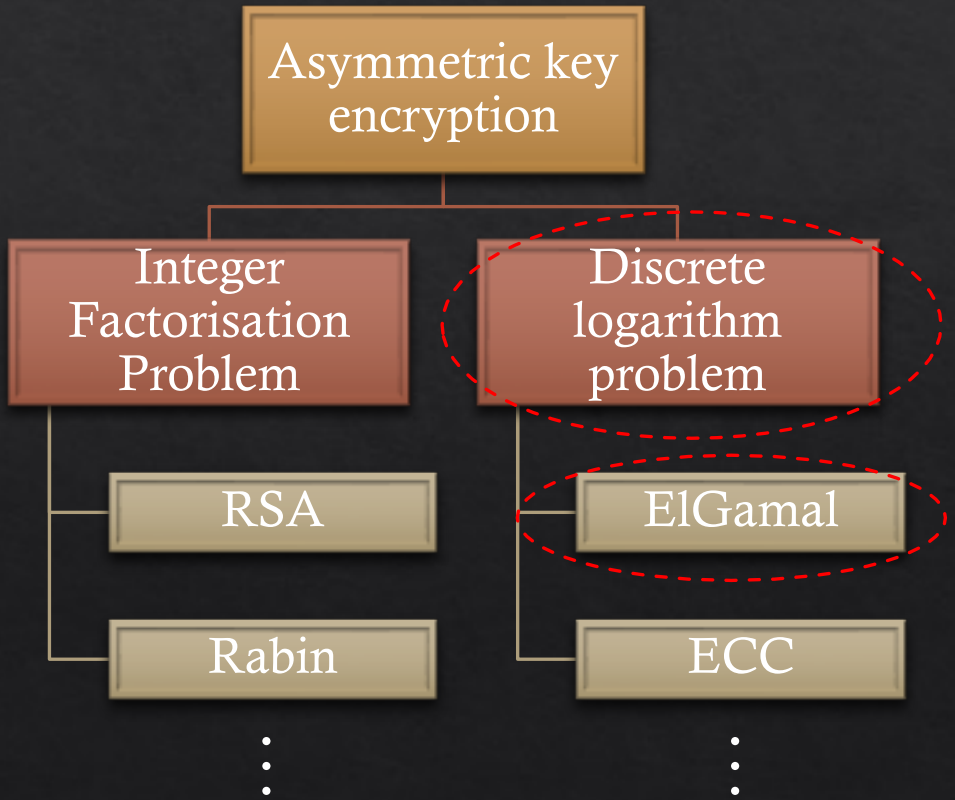
Additional Items

- DES details
  - https://www.tutorialspoint.com/cryptography/data_encryption_standard.htm
- AES encryption animation (mind the music)
  - https://www.youtube.com/watch?v=evjFwDRTmV0
- Block Cipher Modes
  - http://www.crypto-it.net/eng/theory/modes-of-block-ciphers.htm
- DH C code
  - https://www.geeksforgeeks.org/implementation-diffie-hellman-algorithm/
- RSA and ECC tutorial
  - https://www.tutorialspoint.com/cryptography/public_key_encryption.htm
- RSA in more details
  - https://blog.sigmaprime.io/introduction-to-rsa.html
- Basics of ECC
  - https://dl.acm.org/citation.cfm?id=3064818
- Blockchain resources
  - Demo online: https://anders.com/blockchain/
  - Other video/resource from MIT: http://blockchain.mit.edu/
  - Bitcoin developer reference: https://bitcoin.org/en/developer-reference
  - Mining: https://www.investopedia.com/tech/how-does-bitcoin-mining-work/

Supplementary materials

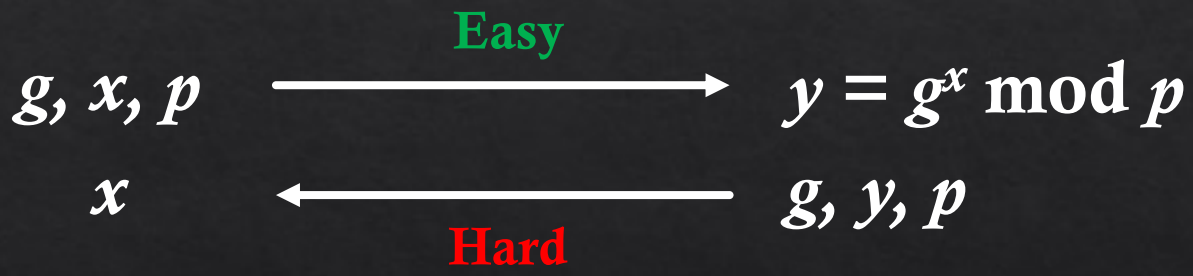
Asymmetric key ciphers

- Given g, x and a large prime p
- Compute y = g^x mod p
- Given *y*, how to get *x*?



Discrete Logarithm problem

- Given g, y and prime p, find an integer x, if any, such that $y = g^x \bmod p$



- Used to construct Diffie-Hellman and ElGamal-type public cryptosystems
  - DH, DSA (Digital Signature Algorithm) etc.

DH Key Exchange

- Diffie-Hellman is a public key distribution scheme
- Proposed by Whitfield Diffie and Martin Hellman in 1976

TCP/IP Model	OSI Model	Protocols
Application	Application	DNS, DHCP, FTP, HTTPS, **SSH** etc.
	Presentation	JPEG, MIDI, MPEG, TIFF etc.
	Session	NetBIOS, NFS, PAP, SQL, ZIP etc.
Transport	Transport	TCP, UDP, **SSL/TLS** etc.
Internet	Network	ICMP, IGMP, **IPSec**, IPv6 RIP etc.
Link	Data Link	ARP, ATM, FDDI, PPP, STP, HDLC etc.
	Physical	Bluetooth, Ethernet, DSL, Wi-Fi etc.

DH key agreement protocol

- Users can exchange a secret key
- Requires **no prior secrets**, can be established in **real time** over untrusted network
- Based on discrete logarithms of large numbers
- Required numbers are:
  - p: one prime
  - g: a primitive root of p (or a base)
  - x: a secret key

$$y = g^x \bmod p$$

DH process

Choose p, a large prime, and g (g < p)

Pick a random number a

Pick a random number b

Sender

Send $g^a \bmod p$

Send $g^b \bmod p$

Receiver

Compute k

$k = (g^b)^a = g^{ab} \bmod p$

Compute k

$k = (g^a)^b = g^{ab} \bmod p$

DH process

- p and g are both publicly available numbers
  - p is at least 512 bits
- Users pick private values a and b
- A common key k is used
- Attacker has g^a, g^b, p, g
  - Has to compute g^{ab} from the above information
  - Discrete Logarithm Problem!

DH Example

- Alice and Bob choose public numbers
  - p = 17
  - g = 6
- Alice picks a = 5, Bob picks b = 3.
  - $g^a \bmod 17 = 6^5 \bmod 17 = 7$
  - $g^b \bmod 17 = 6^3 \bmod 17 = 12$
- Alice and Bob computes $k = g^{ab}$
  - $k = g^{ab} \bmod 17 = (6^5)^3 \bmod 17 = (6^3)^5 \bmod 17 = \mathbf{3}$
- Alice and Bob now use **3** for the symmetric cipher!

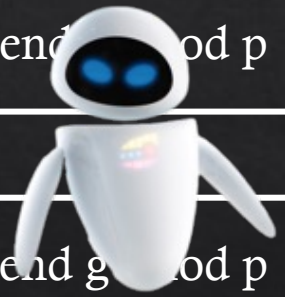
DH MITM attack

- MITM – Man In The Middle attack

Sender

Send g^a mod p

Send g^c mod p



Send g^b mod p

Send g^c mod p

Receiver

Compute k_1

$k_1 = (g^c)^a = g^{ac} \bmod p$

Compute k_1 and k_2

$k_1 = (g^a)^c = g^{ac} \bmod p$

$k_2 = (g^b)^c = g^{bc} \bmod p$

Compute k_2

$k_2 = (g^c)^b = g^{bc} \bmod p$

DH MITM attack

- Attacker can exploit DH with MITM attack
- The problem is the lack of authentication
- To prevent MITM, use authentication
  - E.g., using signatures
  - ISO/IEC 9796 (9796-2:2010 is the current version)

DH MITM mitigation

Sender

A, g^a

$B, g^b, SIG_B(g^a, g^b, \mathbf{A})$

$SIG_A(g^b, g^a, \mathbf{B})$

Receiver

Include the identify in the signature to thwart identity misuse attacks (e.g., MITM)

DH MITM mitigation

Sender

A, g^a

E, g^e, $SIG_E(g^a, g^e, \mathbf{A})$

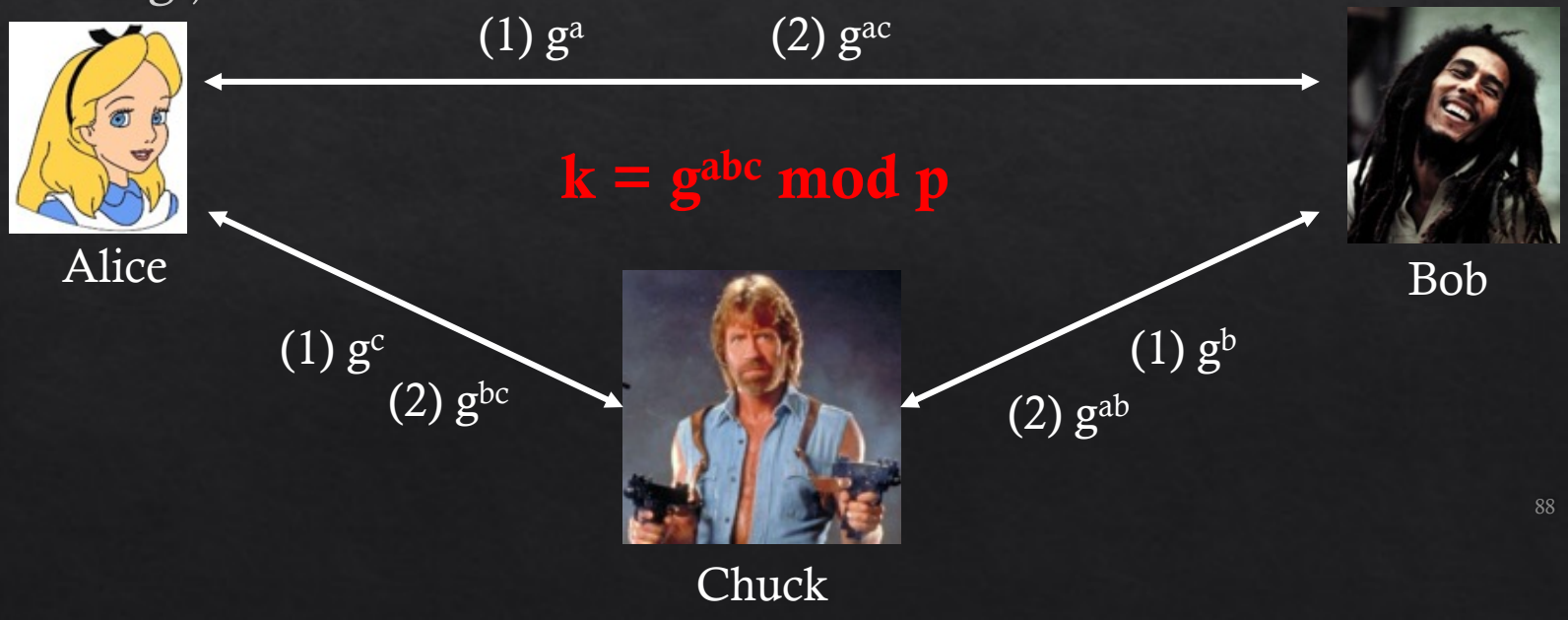
Cannot validate the identify using Bob's public key

Receiver

Even cannot forge Bob's public key without knowing Bob's private key.
We assume Bob's public key is known to Alice.

DH with 2+ Users

- DH can be used more than 2 users at a time
  - E.g., 3 users



RSA vs DH

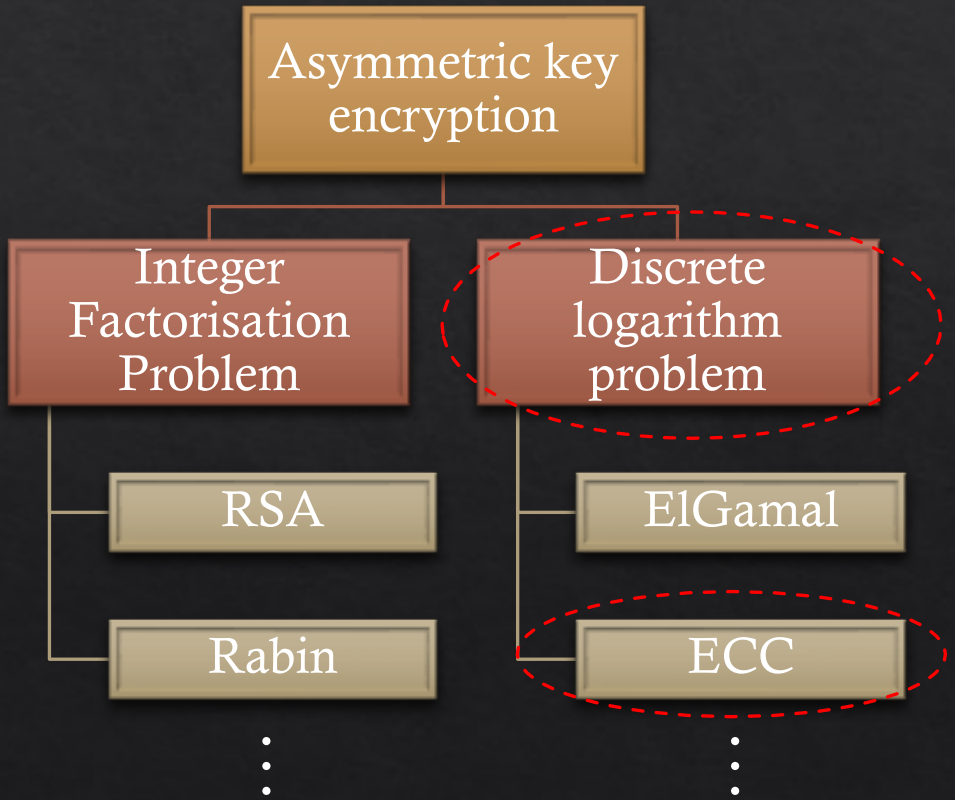
- RSA
  - Can be used for message encryption decryption
  - Can provide authentication
  - Allow anyone to communicate with me
- DH
  - Only exchange shared key, typically for symmetric ciphers
  - Requires authentication to thwart attacks
  - Already identified who I want to communicate with

DH Exercise

- Get into groups of 3
- Choose Alice, Bob and Eve
- Alice and Bob decides p and g (< 20), and show to Eve
- Alice and Bob pick a and b respectively, keep it secret from everyone
- Alice send g^a, and Bob sends g^b vice versa
- Eve will know g^a and g^b
- Alice and Bob calculate $k = g^{ab}$
- Can Eve find k?

Asymmetric key ciphers

- Given g, x and a large prime p
- Compute y = g^x mod p
- Given *y*, how to get *x*?



ECC Intro

- ECC – Elliptic Curve Cryptography
- We will not cover the mathematical details (it is complicated)
  - Recommended to take math courses for further details
- We will look at some basics of ECC

ECC

- Given a set *E* consisting of points (x_i, y_i) in a plane
- There exists a **group operation** (denoted by +) such that given two points p_1 and p_2 in E, $p_1 + p_2 = p_3$ which is also in *E*
- Given point p_i, we can find points $p_i + p_i$, $p_i + p_i + p_i$, $p_i + p_i + \ldots + p_i$ for **k-1** number of group operations
  - We denote this as $\mathbf{k \times p_i}$
- We can easily calculate $k \times p_i$, but finding k given $k \times p_i$ is **difficult**
  - This property is achieved by using an *elliptic curve*

ECC

- Alice chooses an integer X_A, which is the private key
- Alice then generate $Y_A = X_A \times G$, which is the public key
- The same for Bob
- Follow the similar process of DH to exchange secret key
  - Alice calculates: $k = X_A \times Y_B$
  - Bob calculates: $k = X_B \times Y_A$

ECC

- $k = X_A \times Y_B$

$$= X_A \times (X_B \times G)$$

$$= X_B \times X_A \times G$$

$$= X_B \times Y_A$$

- Which is the same as Bob's ($k = X_B \times Y_A$)
- Eve has to find X_A (or X_B) from Y_A, but $Y_A = X_A \times G$!

Advantages of ECC

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 1: NIST Recommended Key Sizes

Applications of ECC

- Due to the smaller size calculated needed compared to RSA, it requires less computational power
- Uses of ECC includes
  - Contactless smartcards
  - Wireless sensor networks
  - RFID
  - Mobile, IoT, vehicle communications etc etc…

Symmetric vs Asymmetric

	Symmetric	Asymmetric
Key type	Shared secret	Public and Private Key pair
Algorithm	Classified or Open	Open
Key distribution	Required	Not required
Strengths	• Fast • Easy to use	• Provides authenticity of the source • Private key is kept secret
Weaknesses	• Shared secret • No authenticity	• Public key management • Slow
Examples	AES, 3DES	RSA, DH, ECC

Best of the both

- Key distribution by Public key cipher
  - E.g., RSA, DH, ECDH-RSA
- Data encryption by symmetric key cipher
  - E.g., AES

- **Most security protocols use this approach**