Basic Cryptography
for
Bitcoin and Blockchain

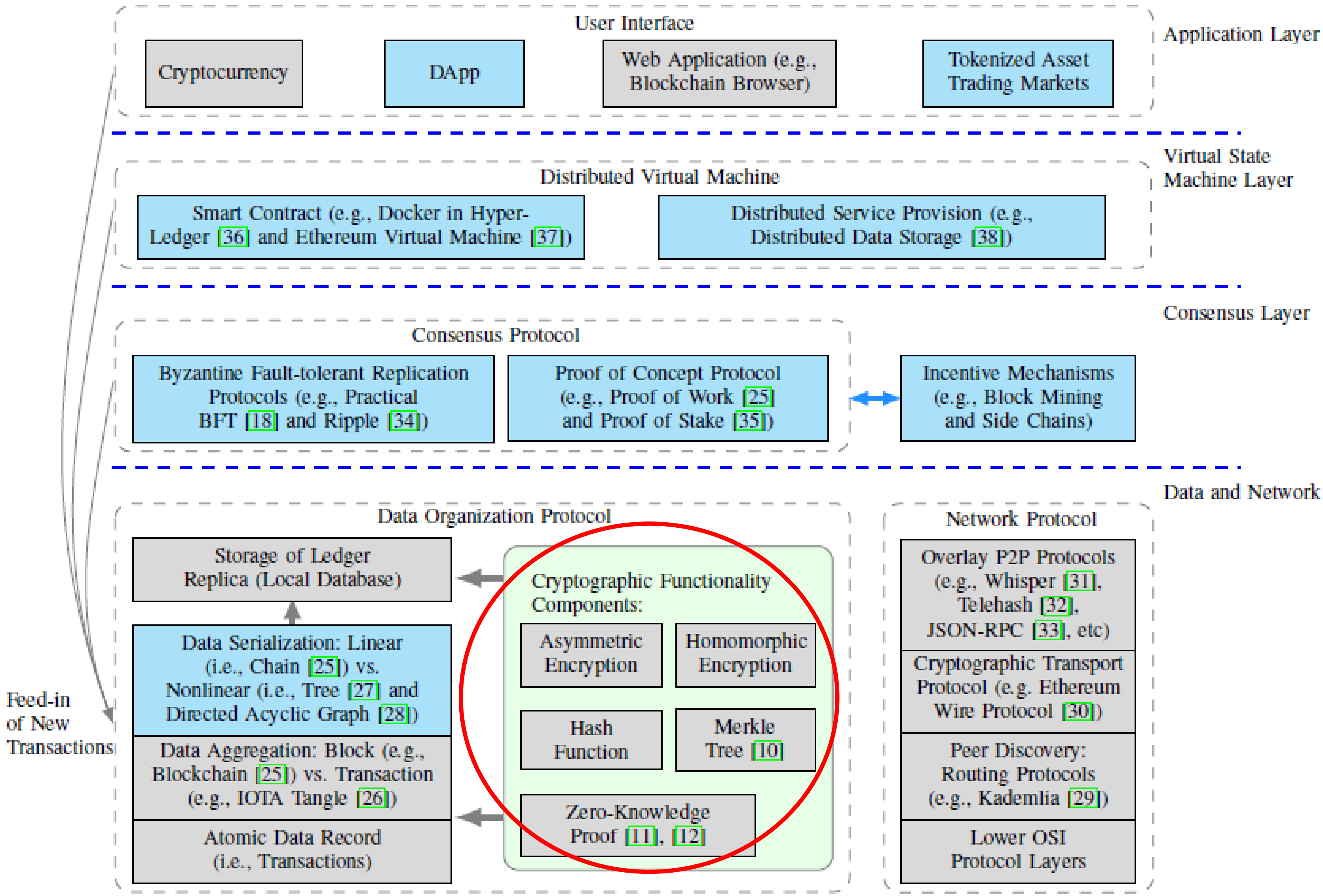
Course Outline

- Introduction
- Money and token economy
- Cryptography
- Blockchain data structure
- Mining and PoW
- Consensus algorithms / Filecoin
- Bitcoin as a platform
- Ethereum and smart contracts
- Distributed applications & enterprise DLT
- Security
- Scalability
- Privacy

Previous Weeks (Review) & Today

- **Blockchain for data, computation, intelligence**
  - Why, How, distributed computing
  - Central vs distributed model
- **Historical perspectives & eco-system**
  - Token economy & crypto industry
  - ICO, STO, stable coin
  - Applications and projects
- **Cryptography (today)**
  - Secret key & public key cryptography
  - Cryptographic hashing
  - Elliptic Curve Cryptography for Bitcoin/Blockchain
  - Secure communication and key distribution

Blockchain Network Implementation Stacks



From “A survey on consensus mechanisms and mining management in blockchain networks” by Wang and Hoang

What is Cryptography?

- Crypto (hidden or secret) graphy (writing): Greek, art of secret writing
- Cryptography = the science (art) of encryption
- Cryptanalysis = the science (art) of breaking encryption
- Cryptology = cryptography + cryptanalysis

- Cryptography can characterize by:
  - type of encryption operations used
    - ✓ substitution / transposition / product
  - number of keys used
    - ✓ single-key or private / two-key or public
  - way in which plaintext is processed
    - ✓ block / stream

Goals of Cryptography

- <u>Confidentiality</u> (secrecy) - only authorized parties are able to understand the data
- <u>Data Integrity</u> - message that arrives is the same as originally sent (not altered in transmission)
- <u>Authentication</u> - Both the sender and receiver need to confirm the identity of other party
- <u>Non-repudiation</u> - An entity is prevented from denying its previous commitments or actions

- <u>Availability</u> - Timely accessibility of data to authorized entities
- <u>Access control</u> - An entity cannot access any entity that it is not authorized to
- <u>Anonymity</u> - The identity of an entity if protected from others

Confidentiality

- Only authorized parties are able to understand the data (authorized from the perspective of the party that encrypted the data).
- It is okay if unauthorized parties know that there is data. It is even okay if they copy the data, so long as they cannot understand it – versus information hiding

Integrity

- **When message is sent over network, the data that arrives is the same as the data that was originally sent. Data is not tampered with.**
- **Technical solutions include:**
  - Encryption (with some keys)
  - Hashing algorithms (no key)

Authentication

- How can we know that a party that provides us with sensitive data is an authorized party?
- How can we know that the party that is accessing sensitive data is an authorized party?
- Two solutions are:
  - Passwords
  - Digital signatures

Nonrepudiation

- Ensuring that the intended recipient actually got the message.
- Ensuring that the alleged sender actually sent the message.
- This is a difficult problem. How do we prove that a person's cryptographic credentials have not been compromised? E.g. man-in-the-middle attacks

Security Attacks

Passive attacks

- Obtain message contents
- Monitoring traffic flows

Active attacks

- Masquerade (wearing masks) of one entity as some other
- Replay previous messages
- Modify messages in transmit
- Add, delete messages
- Denial of service

Cryptographic Attacks

- Ciphertext only: attacker has only ciphertext
- Known plaintext: attacker has plaintext and corresponding ciphertext – each cipher is independent
- Chosen plaintext: attacker can encrypt messages of his choosing
- Distinguishing attack: an attacker can distinguish your cipher from an ideal cipher (random permutation)
- A cipher must be secure against all of these attacks

Kerckhoffs' Principle

- **The security of an encryption system must depend only on the key, not on the secrecy of the algorithm**
  - Nearly all proprietary encryption systems have been broken (Enigma, DeCSS, zipcrack).
  - Secure systems use published algorithms (PGP, OpenSSL, Truecrypt).

Provable Security

- There is no such thing as a provably secure system
- Proof of unbreakable encryption does not prove the system is secure
- The only provably secure encryption is the one time pad: C = P + K, where K is as long as P and never reused
- Systems are believed secure only when many people try and fail to break them

Old Style Cryptography

Old Style Cryptography

- **Shift of alphabet**
  - e.g. Caesar cipher A=D, B=E, C=F
- **Many more sophisticated systems developed from 1500s to mid-20th century**
  - Substitution and transposition of letters
  - Some essentially unbreakable by manual means
- **Made obsolete by computers since 1940**

Example: The Caesar Cipher

K=3

Outer: plaintext

Inner: ciphertext

An Example

- For a key K=3,
  plaintext letter: `ABCDEF...UVWXYZ`
  ciphtertext letter: `DEF...UVWXYZABC`

- Hence
  `TREATY IMPOSSIBLE`
  is translated into
  `WUHDWB LPSRVVLEOH`

Breaking the Caesar cipher

- With the help of fast computers, 99.99% ciphers used before 1976 are breakable
- By trial-and error
- By using statistics on letters
  - frequency distributions of letters

letter	percent
A	7.49%
B	1.29%
C	3.54%
D	3.62%
E	14.00%

....

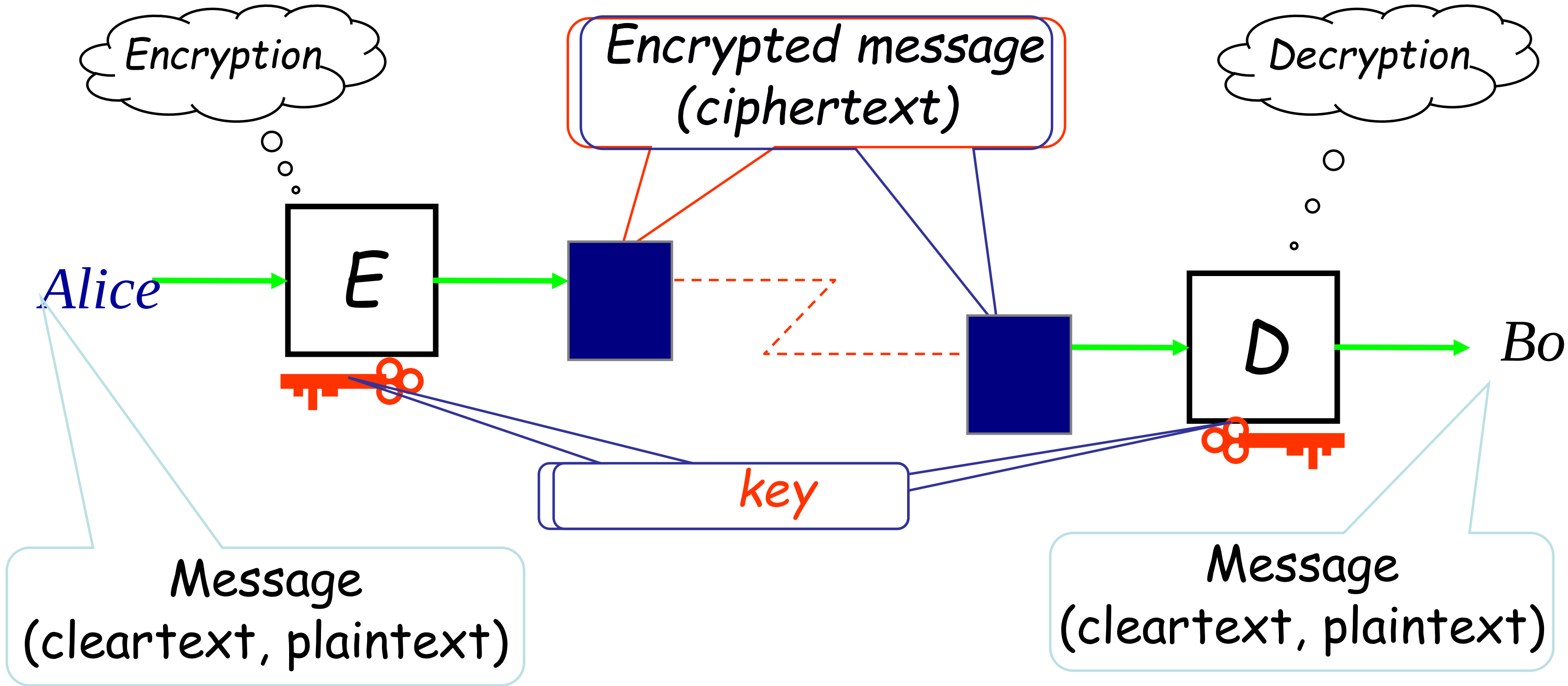
Modern Cryptography

Modern Cryptography

- **Asymmetric Cipher: hard problems in mathematics**
  - Breaking the system requires an efficient algorithm for solving a hard problem – e.g. Factoring large numbers, discrete logarithms
  - Examples: RSA, El Gamal
  - Used in public key systems – Asymmetric Cryptography
  - Slow
- **Symmetric Cipher: information theory**
  - Texts scrambled by repeated application of bit shifts and permutations
  - Examples: DES, AES
  - Used in private key systems – Symmetric Cryptography
  - Fast
- **Hash functions**

Secret Key Cryptography

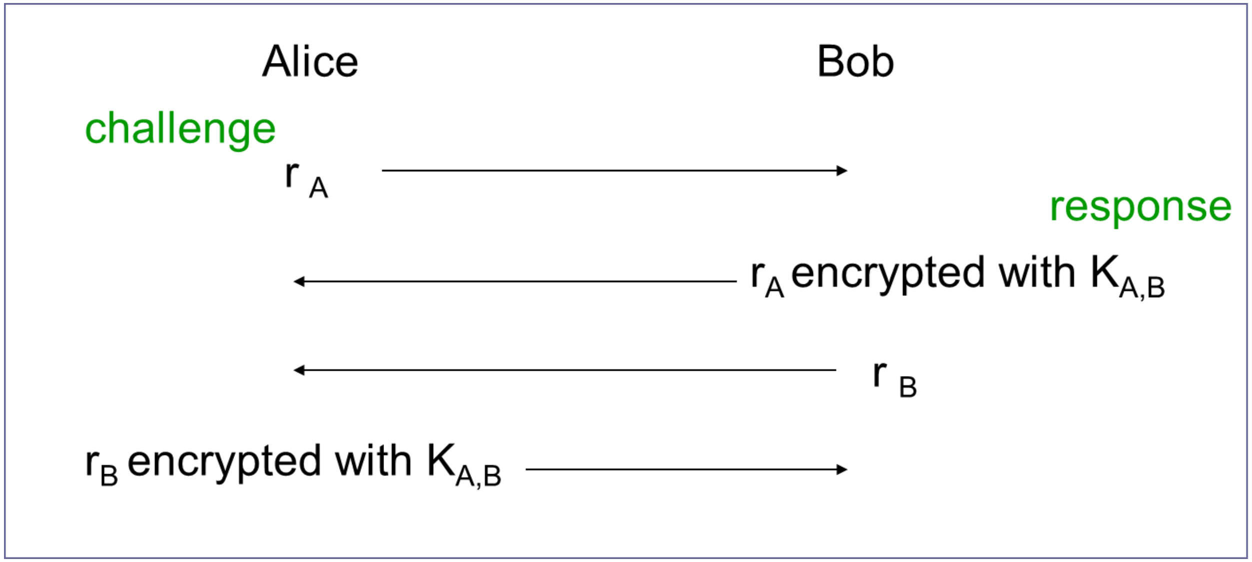
Secret Key Cryptography



- Using a single key for encryption/decryption
- The plaintext and the ciphertext having the same size
- Also called *symmetric* key cryptography

SKC Applications

- **Transmitting over an insecure channel**
  - Message encrypted by sender and decrypted by receiver, w/ the same key
  - Prevent attackers from eavesdropping
- **Secure storage on insecure media**
  - Data is encrypted before being stored somewhere
  - Only the entities knowing the key can decrypt it
- **Authentication**
  - Strong authentication: proving knowledge of a secret without revealing it.



SKC Applications

❑ Integrity Check

- ❖ Noncryptographic checksum
  - ✓ Using a well-known algorithm to map a message (of arbitrary length) to a fixed-length checksum
  - ✓ Protecting against accidental corruption of a message
  - ✓ Example: CRC

- ❖ Cryptographic checksum
  - ✓ A well-know algorithm
  - ✓ Given a key and a message
  - ✓ The algorithm produces a fixed-length Message Authentication Code (MAC) that is sent with the message

Example of Secret Key Cryptography

- Message broken into 64-bit blocks and each 64-bit block of plaintext is encrypted separately

- Encryption: Each plaintext block is exclusive-ored with the key starting from the first byte of the block, repeatedly to the end of the block (the key moves a distance of its size from left to right of the plaintext block).

- Decryption: do the reverse of encryption: the cipher-text is exclusive-ored.

$$
\begin{array}{ccccc}
0 & \oplus & 0 & = & 0 \\
1 & \oplus & 1 & = & 0 \\
0 & \oplus & 1 & = & 1 \\
1 & \oplus & 0 & = & 1
\end{array}
$$

$\oplus$: *exclusive* ***or***

Example of Secret Key Cryptography

The plaintext:

0	1	0	0	0	0	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0

The key:

1	1	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0

The ciphertext:

1	0	0	1	0	0	1	0	0	0	1	1	1	0	0	0	0	1	1	0	1	0	0	1

ciphertext:

1	0	0	1	0	0	1	0	0	0	1	1	1	0	0	0	0	1	1	0	1	0	0	1

XOR'd with key

1	1	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0

yields plaintext

0	1	0	0	0	0	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0

Standard Algorithms are Incredibly Secure

- Encryption algorithm & related key kept secret
- Breaking the system is hard due to large numbers of possible keys
- For example: for a key 128 bits long, there are

$$2^{128} \approx 10^{38}$$

keys to check using brute force.

1 year = 31x10^6 sec; 1 sec = 10^9 operations

- Even with the most powerful computing resources, most of the software developers alive today will be dead before one could break such an encryption

Incredibly Secure (cont.)

- Most security experts believe that 256-bit keys are good for the lifetime of the universe (many billions of years)

- The problem is that encryption is just one link in the chain of security. Encryption is a really strong link in that chain, but one weak link breaks the chain

- It is usually easier for the attacker to hack your machine and steal the plaintext than to break your cipher – social engineering

General Approaches

- There are two general encryption methods: Block ciphers & Stream ciphers

- **Block ciphers**
  - Slice message M into (fixed size blocks) m_1, …, m_n
    - ✓ Add padding to last block
  - Use E_k to produce (ciphertext blocks) x_1, …, x_n
  - Use D_k to recover M from m_1, …, m_n
  - e.g.: DES

- **Stream ciphers**
  - Generate a long random string (or pseudo random) called ***one-time pad***.
  - Message $\oplus$ ***one-time pad*** (exclusive or), e.g.: EC4
  - Cipher achieves perfect secrecy if and only if there are as many possible keys as possible plaintexts, and every key is equally likely (Claude Shannon's result)

Stream Ciphers (One-Time Pad)

Advantages

- **Easy to compute**
  - Encryption and decryption are the same operation
  - Bitwise XOR is very cheap to compute
- **As secure as possible**
  - Given a ciphertext, all plaintexts are equally likely, regardless of attacker's computational resources
  - …as long as the key sequence is truly random
    - ✓ True randomness is expensive to obtain in large quantities
  - …as long as each key is same length as plaintext
    - ✓ But how does the sender communicate the key to receiver?

Disadvantages

- **Key must be as long as plaintext**
  - Impractical in most realistic scenarios
  - Still used for diplomatic and intelligence traffic
- **Does not guarantee integrity**
  - One-time pad only guarantees confidentiality
  - Attacker cannot recover plaintext, but can change it to something else
- **Insecure if keys are reused**
  - Attacker can obtain XOR of plaintexts

Secret Key Encryption Attacks

Attacks on Encryption Algorithms:

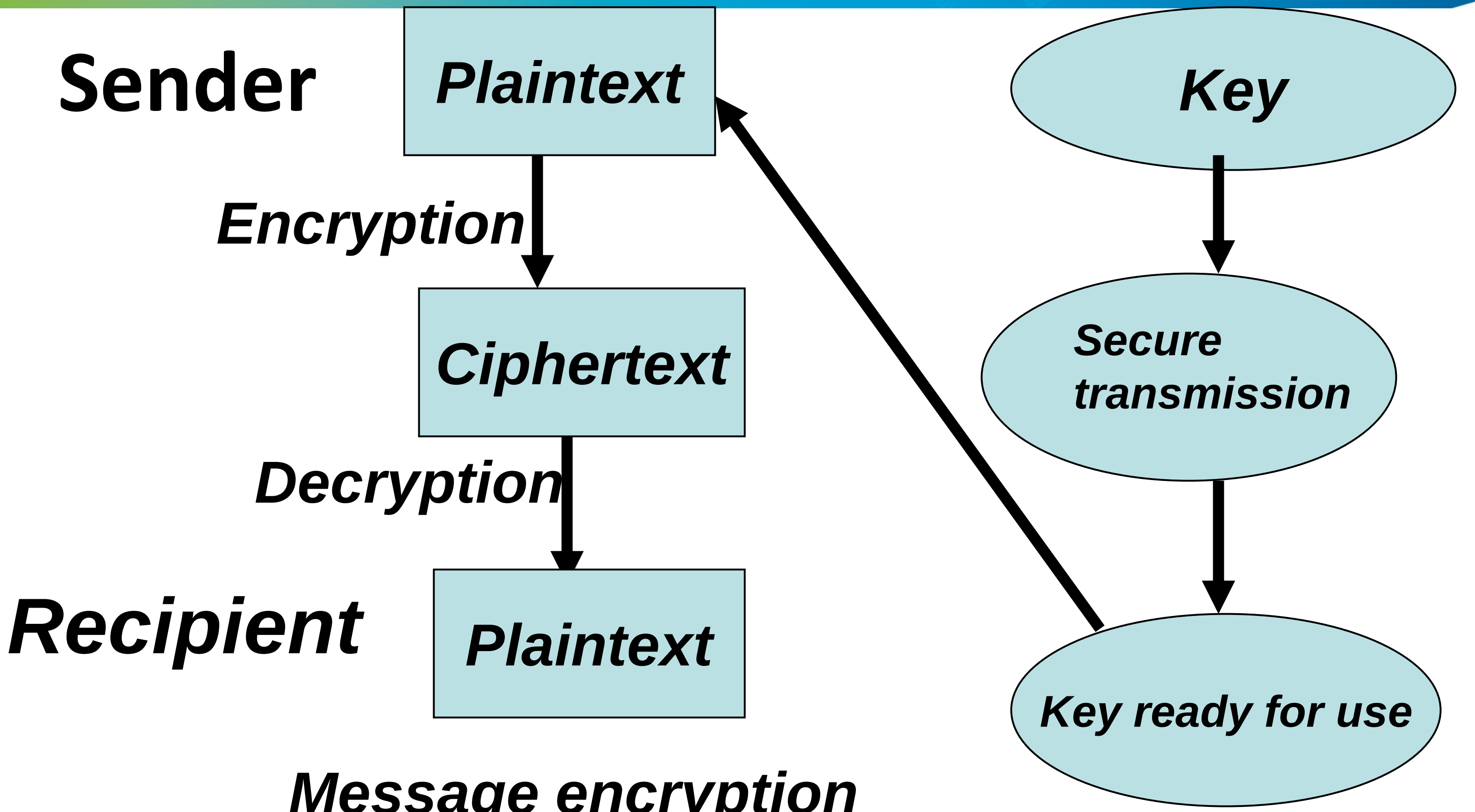
- ❖ Substitution Ciphers: Frequency Attacks
- ❖ One Time Pads are provably secure
- ❖ Modern Attacks:
  - ✓ Linear Cryptanalysis looks for a linear relationship between plaintext and ciphertext. (Known Plaintext Attack)
  - ✓ Differential Cryptanalysis looks at how differences in plaintext cause differences in ciphertext. (Chosen Plaintext Attack)

Secret Key Algorithm Design

Modern Encryption Algorithm Design Techniques

- ❖ Confusion and Diffusion
  - ✓ Diffusion means many bits of the plaintext (possibly all) affect each bit of the ciphertext
  - ✓ Confusion means there is a low statistical bias of bits in the ciphertext
- ❖ Non-Linearity: The encryption function is not linear (represented by a small matrix)
  - ✓ Prevents Linear Cryptanalysis

Secure Key Distribution



Message encryption

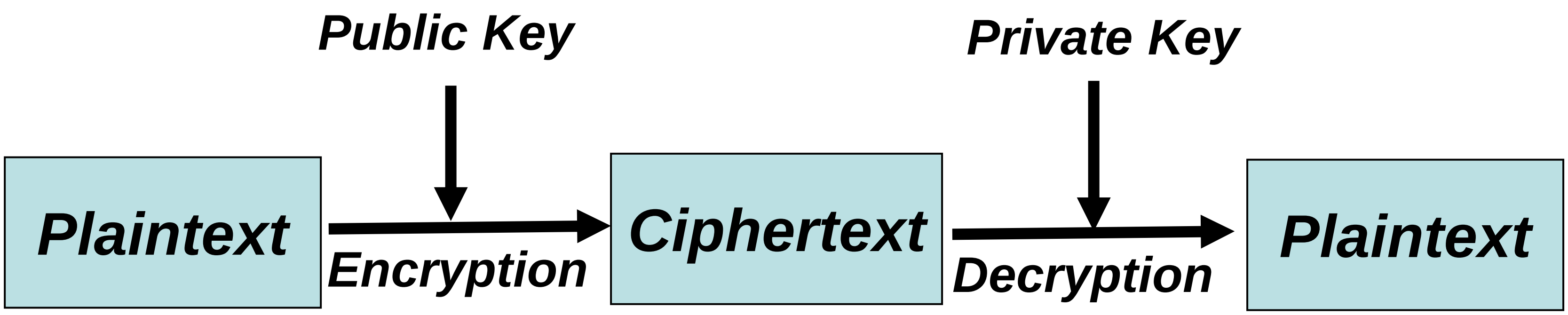
Secure key distribution

Hard Problem for conventional encryption

Public Key Cryptography

Public Key Cryptography

- In 1970s the Public Key Cryptography emerged
- Each user has two mutually inverse keys
- The encryption key is published
- The decryption key is kept secret
- Anybody can send a message to Bob, only Bob can read it



PKC Applications

Transmitting over an insecure channel

Alice		Bob
encrypt m_A using e_B	$\longrightarrow$	encrypt m_A using d_B

Secure storage on insecure media

- Data is encrypted with the public key of the source, before being stored somewhere
- Nobody else can decrypt it (not knowing the private key of the data source)

Authentication

Alice		Bob
encrypt r using e_B	$\longrightarrow$	decrypt to r using d_B
	$\longleftarrow$	r

Public Key Cryptography – Digital Signature

- Generate a digital signature on a message
- Digital signature is a number associated with a message
- Can only generated by someone knowing the private key
- Verification of signature only requires public key
- Non-repudiation: the signing individual can not be denied, because only he knows the private key

Example of Public Key Cryptography

- Definition: The multiplicative inverse of ***x*** with modulo ***n*** is ***y*** such that (x*y) mod n = 1
  e.g.: x=3; n=10, => y=7; since (3*7) mod 10 = 1

- The above multiplicative inverse can be used to create a simple public key cipher: either ***x*** or ***y*** can be thought of as a secret key and the other is the public key. Let ***x*** = 3, ***y*** = 7, ***n*** = 10, and M be the message:

- X=3 is public key; y=7 is private key

  - M = 4 ;
    - ✓ 3*4 mod 10 = 2; (create ciphertext) - encrypting
    - ✓ 2*7 mod 10 = 4 = M ; (recover message) – decrypting

  - M =6 ;
    - ✓ 3*6 mod 10 = 8;
    - ✓ 8*7 mod 10 = 6 = M (message)

Public Key Crypto

- Public key cryptography circumvents key distribution problem completely
- Public key algorithms is incredibly slow relative to symmetric key algorithms, e.g. 100x slower than DES
- In general, encrypting large messages using public key cryptography is not considered practical

Rivest, Shamir, and Adelman (RSA)

- RSA is the most famous public key algorithm
- It picks two HUMONGOUS prime numbers, p and q, containing hundreds to thousands of bits
- Two prime numbers remain secret (private keys)
- $n = p*q$ is the public key - to encrypt the message
- Only someone knows the prime factors can decrypt the message in a reasonable amount of time

RSA: Choosing keys

1. Choose two large prime numbers p, q.

2. Compute n = pq, z = (p-1)(q-1)

3. Choose e (with e<n) that has no common factors with z. (e, z are “relatively prime”).

4. Choose d such that ed-1 is exactly divisible by z. (in other words: ed mod z = 1).

5. Public key is $\underbrace{(n,e)}_{K_B^+}$. Private key is $\underbrace{(n,d)}_{K_B^-}$.

RSA: Encryption, Decryption

0. Given (n,e) and (n,d) as computed above

1. To encrypt bit pattern, m, compute

$$c = m^e \bmod n$$

2. To decrypt received bit pattern, c, compute

$$m = c^d \bmod n$$

Verify This!

$$m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$$

RSA: Why is that $m = (m^e \bmod n)^d \bmod n$

Euler's theory: *If p,q prime and n = pq, then:*

$$m^{\varphi(n)} \equiv 1 \pmod{n} \qquad \phi(n) = (p-1)(q-1)$$

(since "ed " divide by (p-1)(q-1) with remainder 1)

$$
\begin{aligned}
(m^e \bmod n)^d \bmod n &= m^{ed} \bmod n \\
&= m^{1+k \times (p-1)(q-1)} \bmod n \\
&= m^1 \bmod n \\
&= m
\end{aligned}
$$

RSA

- RSA algorithm is based on the difficulty of factoring a product of two large primes.

Easy Problem

Given two large primes p and q compute

$$n = p \times q$$

Hard Problem

Given n compute p and q.

At this point in history, this is a difficult problem.

Factoring a product of two large primes

- The best known algorithm requires the time proportional to:

$$T(n) = \exp[c(\ln n)^{1/3}(\ln \ln n)^{2/3}]$$

For p & q of 65 digits long T(n) is about 1 month using cluster of workstations.

For p & q of 200 digits long T(n) is astronomical.

Issues with RSA

- RSA is still considered secure after twenty years of use.
- The big security problem is that some implementations of RSA have been flawed and had security problems of their own.
- The software developer should use a well-tested and highly-regarded implementation of RSA.

Are there enough primes to use?

- There are huge numbers of large prime numbers.
- There are approximately 10^{151} primes of length 512 bits or less.
- One interpretation is that there are enough primes of up to 512 bits to assign every atom in the universe 10^{74} prime numbers without ever repeating one of those primes.

The Future of RSA

- The future of RSA is hard to predict - depends upon what happens in prime number factoring theory
- Not too many years ago, experts believed it is hard to factor a 128 bit number
- Now, with adequate resources, one can factor a 512 bit number in just a few months
- Recommend to use no less than a 2,048 bit key for data requiring long-term security (ten or more years)
- 1,024 bit numbers may be nearing the end of their usefulness even for short-term security
- The longer the key, the longer it takes to encrypt messages using public key cryptography

Quantum Computing algorithm for factoring

- In 1994 Peter Shor from the AT&T Bell Laboratory showed that in principle a quantum computer could factor a very long product of primes in seconds.

- Shor's algorithm time computational complexity is

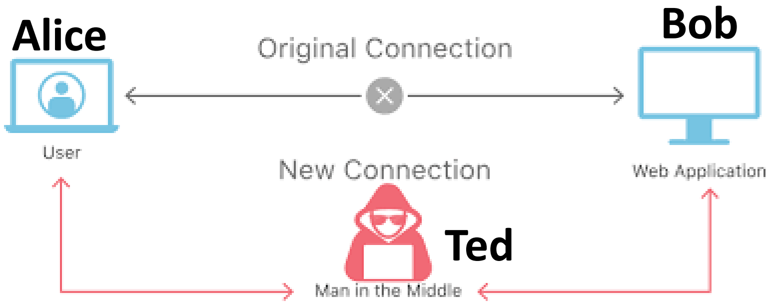
$$T(n) = O[(\ln n)^3]$$

Once a quantum computer is built the RSA method would not be safe

Public Key Crypto Vulnerabilities

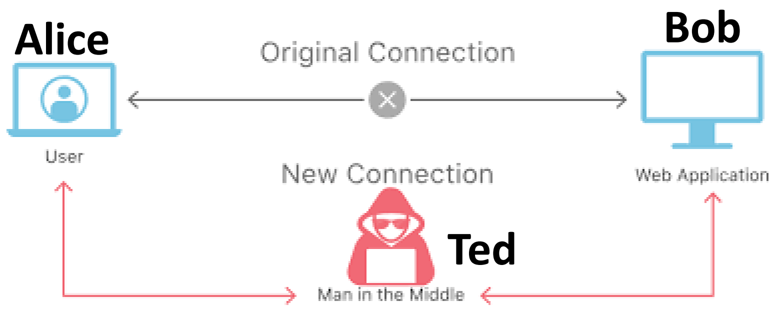
"Man-in-the-Middle" Attacks

- ❑ Alice tries to send a message to Bob
- ❑ Ted pretends to be Bob when communicating with Alice, and pretends to be Alice to Bob
- ❑ Ted sends Alice his public key, misrepresenting it as Bob's public key
- ❑ Ted sends Bob his public key, misrepresenting it as Alice's public key
- ❑ Ted is intercepting all traffic between Bob and Alice.



Man-in-the-Middle (cont.)

- When Alice sends Bob a message, she encrypts it using Ted's public key (she thinks it is Bob's).
- When Ted receives Alice's message, he can decrypt it.
- Ted can then send Bob a modified or entirely different message, encrypting it was Bob's public key.
- Bob decrypts the message, thinking it came from Alice.
- This kind of problem motivates the need for a public key infrastructure (PKI)



Public Key Infrastructure

- A trusted third party certifies valid keys
- Alice would receive Bob's public key through a trusted third party, a certification authority (CA).
- The CA would say, in effect: "Alice, trust us, Bob is a dependable fellow and this is Bob's public key."

- This does not solve the matter of trust (the security problem)
- How can Alice be sure that she can trust the so-called trusted authority?
- One of the largest CAs is Verisign
- Verisign performs background checks on applicants before issuing them a public key for a fee

- Verisign's track record is not perfect.
- Several people registered with Verisign under the name "Bill Gates".
- In March 2001 Microsoft announced that two false keys with MS's name on them had been issued by a CA.

Diffie-Hellman – Sharing Secret Key

- Developed in 1976 by Whitfield Diffie and Martin Hellman of Stanford University
- In 1997 it was revealed that British cryptographers had developed a similar idea in the 1960s and early 1970s

Key protocol: two parties compute a message key for symmetric encryption without that secret being shared explicitly:

1. Each party independently generates a private key
2. Each computes a public key as a function of private key
3. They exchange public keys
4. Each computes a message key (secret key) which is derived from their private key and the other one's public key
5. Both arrive at the same message key

Diffie-Hellman (cont.)

- The public keys must be computed using a one-way function (a hashing function) that makes it impossible to get back the private keys from the publicly exchanged keys
- If an attacker has access to one party's public key and the other party's private key, the attacker could compute the message key
- The mathematics is such that the publicly exchanged keys cannot reveal either party's private key

Diffie-Hellman Key Generation

Basic Idea:

1. Alice and Bob agree on an integer g
2. - Alice secretly chooses integer x, computes $X = g^x$ and sends it to Bob.
   - Bob secretly chooses integer y, computes $Y = g^y$ and sends it to Alice.
3. - Alice computes $Y^x=(g^y)^x=g^{xy}$.
   - Bob computes $X^y=(g^x)^y=g^{xy}$.
4. Alice and Bob both use g^{xy} to create a secret key

Diffie-Hellman Key Generation

Wait!! It's not secure. If Eve overhears what g, X, and Y are she can compute:

$$x = \log_g X \text{ and } y = \log_g Y$$

And use this information to calculate g^{xy}

To make this secure Alice and Bob pick a large prime number P and reduce everything mod P (take the remainder after division by P)

Diffie-Hellman Key Generation

New and Improved Idea:

1. Alice and Bob agree on an integer g and prime P.
2. - Alice secretly chooses integer x, computes $X = g^x \bmod P$ and sends it to Bob.
   - Bob secretly chooses integer y, computes $Y = g^y \bmod P$ and sends it to Alice.
3. - Alice computes
     $$Y^x \bmod P = (g^y)^x \bmod P = g^{xy} \bmod P$$
   - Bob computes
     $$X^y \bmod P = (g^x)^y \bmod P = g^{xy} \bmod P$$
4. Alice and Bob use $g^{xy} \bmod P$ to create a secret key

Diffie-Hellman Key Generation

By adding the prime P into the equation we now need to make sure that g is a "generator" of P. This means that for every integer x in $\{1,2,3,\ldots,P\text{-}1\}$ there exists an integer d such that:

$$x = g^d \bmod P.$$

d is called the "discrete log" of g^d mod P

Diffie-Hellman Key Generation

Why Does This Work?

1. Because the positive integers less than *P* form a multiplicative, cyclic group with generator *g*
2. Hard to compute the discrete log of generative group element

Given these two things:

1. This algorithm works
2. It is hard for attacker to calculate g^{xy} mod *P*

What does this all mean for Diffie-Hellman Key Generation?

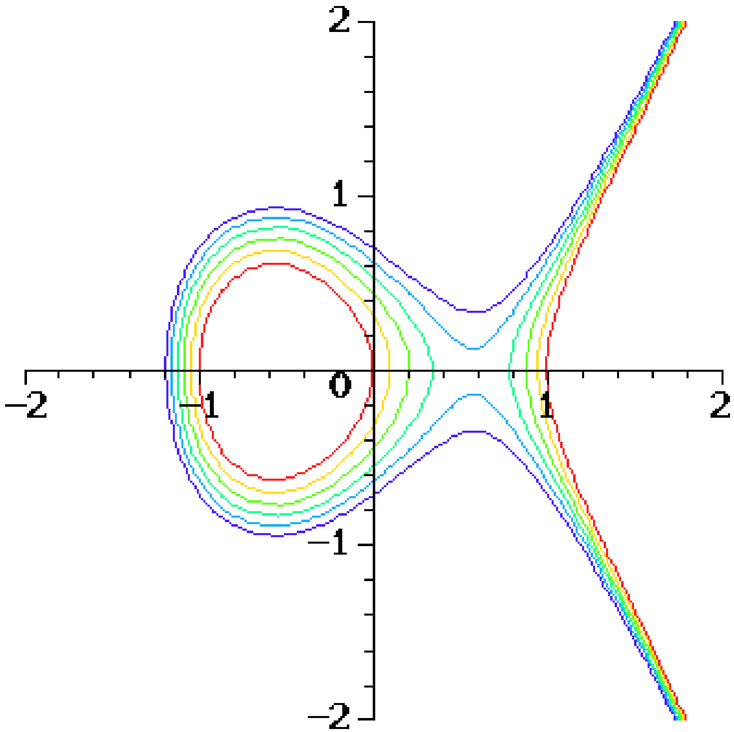
- **Diffie-Hellman will work as a key exchange algorithm in any cyclic group where computing discrete logarithms is hard**

Elliptic Curve Cryptography

- Elliptic Curves are a way of modifying existing crypto systems like DH to make them “stronger”
- “Stronger” means the expected time of an attack is longer with equal key sizes
- This allows us to use smaller key sizes and therefore speed up the whole process
- This makes ECC very useful for small devices like phones or other embedded systems

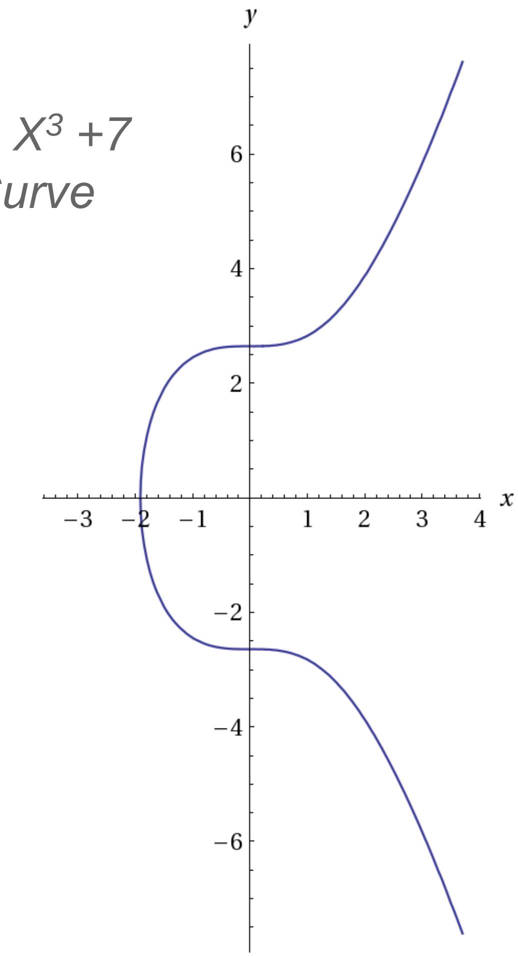
Elliptic Curves

- An Elliptic Curve is such an alternate cyclic group. The group consists of all points of the form: $y^2 = x^3 + ax + b$. Where x, y, a, and b are all elements of a field F.



secp256k1 : $Y^2 = X^3 + 7$

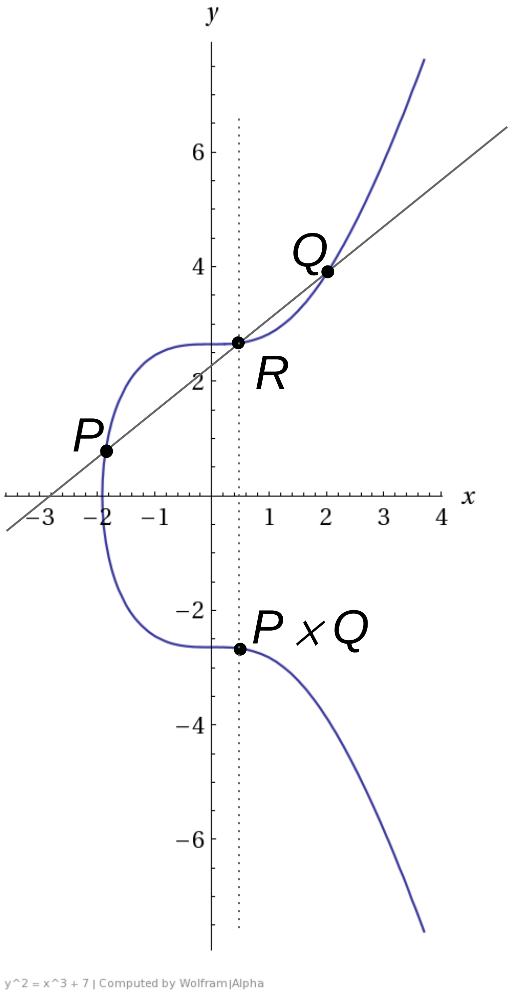
Bitcoin's Elliptic Curve



y^2 = x^3 + 7 | Computed by Wolfram|Alpha

Elliptic Curves - Group Law

- One can define a group law on an elliptic curve using the chord-tangent process
- Given two elliptic curve points, P and Q, we define P × Q as the following:
- We find the line intersecting P and Q, which must intersect with one final point, R. If we then reflect R across the x-axis, we obtain another point which we define as P × Q.



Elliptic Curves - Group Law

More formally,

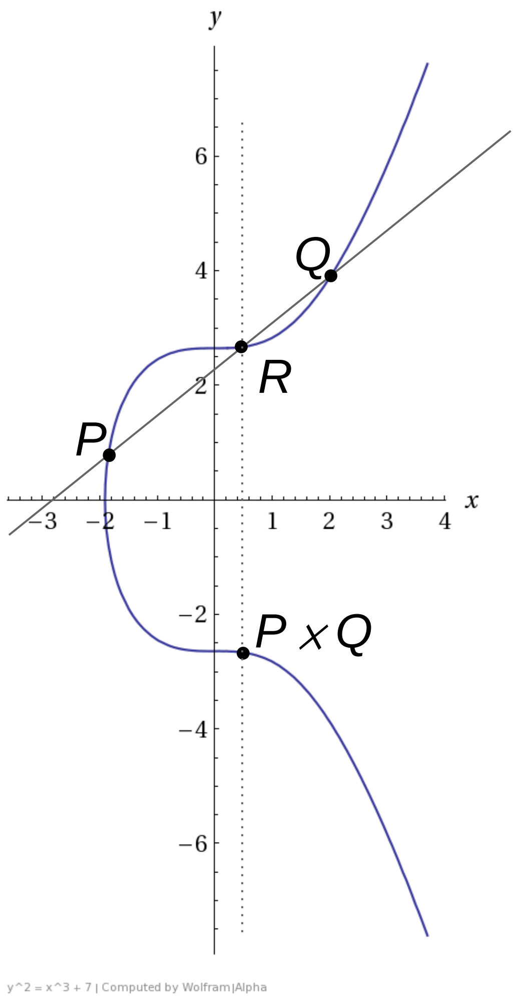
Let $P = (x_1, y_1),\ Q = (x_2, y_2),\ P \times Q = (x_3, y_3)$

$s = \frac{y_2 - y_1}{x_2 - x_1}$

$x_3 = s^2 - x_1 - x_2$

$y_3 = s(x_1 - x_3) - y_1$

Over certain curves and finite fields, this forms a cyclic (or nearly cyclic) finite abelian group.



Elliptic Curves - Group Law

If P = Q, we find the tangent at P, extend it to the point, R, then reflect across the x-axis to P^2.

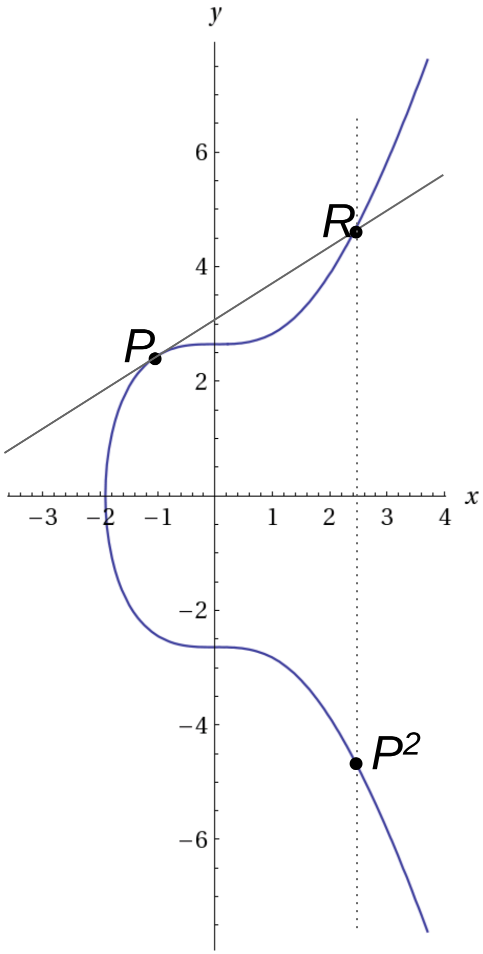
$$P = (x, y)$$

$$s = \frac{3x^2 - a}{2y}$$

$$x' = s^2 - 2x$$

$$y' = s(x - x') - y$$

$$P^2 = (x', y')$$



y^2 = x^3 + 7 | Computed by Wolfram|Alpha

Elliptic Curve Discrete Logarithm Problem (ECDLP)

For some positive integer, m, we can define:

$$P^m = P \times P \times \cdots \times P = \prod_{i=1}^{m} P$$

It is believed that finding m given P and P^m

$$m = \log_P P^m$$

is computationally difficult over certain finite fields and certain curves. As such the ECDLP forms the basis of elliptic curve cryptography.

ECDLP

- Compared with discrete logarithms over vanilla finite fields, the elliptic curve discrete logarithm problem has no known, sub-exponential algorithms (assuming the curve is not supersingular or otherwise anomalous)

- The fastest, practical algorithm for elliptic curve discrete logarithms is parallel Pollard's Rho, which runs in $O(N^{1/2})$

- To achieve security equivalent to a 128-bit block cipher, we need to choose a curve of group order $\approx 2^{256}$

- Compare this with RSA or other factoring-based algorithms, which requires ≈ 2048 bit keys for equivalent security

Elliptic Curve Cryptography

Newer and more Improved Idea:

1. Alice and Bob agree on an Elliptic Curve E (specified by the field F and parameters a, b) and a base point g on E.
2. (a) Alice secretly chooses integer x, computes $X = xg$ and sends it to Bob.
   (b) Bob secretly chooses integer y, computes $Y = yg$ and sends it to Alice.
3. (a) Alice computes: $xY = x(yg) = xyg.$
   (b) Bob computes: $yX = y(xg) = yxg = xyg.$
4. Alice and Bob both share the point xyg which they can use to create a secret key.

Cryptography Hashing

Hashing Function

We define a cryptographic hash function: $H : \{0,1\}^* \longmapsto \{0,1\}^k$

Maps arbitrarily-sized bit string to some fixed-size bit string.

It is deterministic; same input always yields same output.

“The workhorses of modern cryptography” - Bruce Schneier

Example: SHA256 maps to a 256-bit string

```
> echo “Hello, world!” | sha256sum
```

```
0xd9014c4624844aa5bac314773d6b689ad467fa4e1d1a50a1b8a99d5a95f72ff5
```

Properties of Cryptographic Hash Function

Notation: is hidden, is public

Preimage Resistance:

Let $x = \{0,1\}^*$ = message

$y = H(x)$

$x = H^{-1}(y)$ → computationally difficult to find preimage (original value) of a hash output

Second Preimage Resistance:

Given message x

Find some x' s.t. $H(x') = H(x)$ is computationally difficult.

Collision Resistance:

Finding x_1, x_2 such that $H(x_1) = H(x_2)$ computationally difficult.

Upper bound to find a collision is $O(N^{1/2})$ (Birthday Attack)

Cryptographic Hash Functions Are Useful

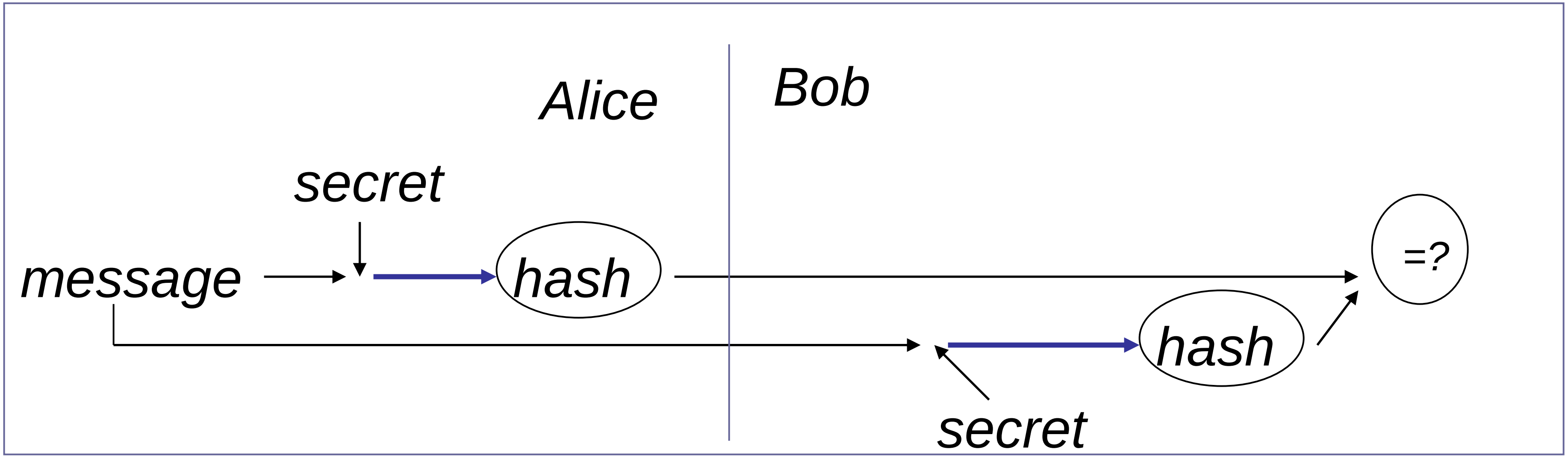
- Fundamental operation in many cryptographic protocols
- Hash-based Message Authentication Codes (HMACs)
- Password Verification
- Commitment Schemes
- Pseudo-Random Number Generators (PRNGs)

In Bitcoin and BlockChain:

- ❑ **Merkle Trees**
- ❑ **Proof-of-Work (Bitcoin, Litecoin, Dogecoin, etc…)**
- ❑ **Transactions, Blocks, Addresses all referenced by hash value**

Hash Functions: Security Uses

- **Password hashing**
  - System store a hash of the password (not the password itself)
  - When a password is supplied, it computes the password's hash and compares it with the stored value.
- **Message integrity**
  - Using cryptographic hash functions to generate a MAC (Message Authentication Code)



Hash Functions: Security Uses

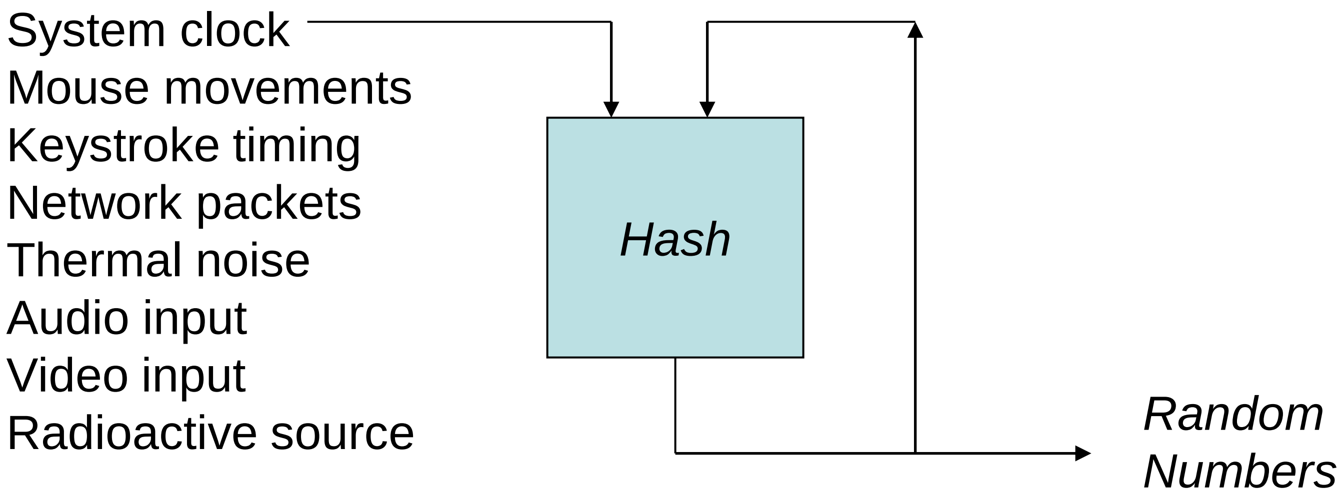
- **Message fingerprint**
  - Save the message digest of the data on a tamper-proof backing store
  - Periodically re-compute the digest of the data to ensure it is not changed.

- **Download security**
  - Using a hash function to ensure a download program is not modified

- **Improving signature efficiency**
  - Compute a message digest (using a hash function) and sign that

Random Number Generation

- Random = not guessable by an attacker.
- Requires a hardware source of entropy.



Simple Hash Commitment Scheme

Why are these hash properties useful?

Consider a simple example: Alice and Bob bet $100 on a coin flip

1) Alice calls the outcome of the coin flip
2) Bob flips the coin
3) Alice wins the $100 if her guess was correct

What if Alice and Bob are separated and don't trust one another?

- Alice wants to give Bob a *commitment* to her guess, without revealing her guess before Bob flips the coin, otherwise Bob can cheat!

Simple Hash Commitment Scheme

Design the "protocol" to bind Alice's guess with a commitment:

1) Alice chooses a large random number, R
2) Alice guesses the outcome of the coin flip, B
3) Alice generates a *commitment* to the coin flip, $C = H(B \,||\, R)$
4) Alice sends this commitment to Bob
5) Bob flips the coin and sends the value to Alice
6) Alice sends Bob the random number and her guess: (R', B')
7) Bob then checks that $C' = H(R' \,||\, B') = C = H(B \,||\, R)$, to ensure Alice did not change her guess mid commitment
8) Both can now agree on who won the $100

Simple Hash Commitment Scheme - Cheating

Could Bob cheat Alice?

When Bob receives $C = H(B \mid\mid R)$, if he can compute $H^{-1}(C) = B \mid\mid R$, Bob can recover Alice's guess and send her the opposite outcome!

If H is preimage resistant, this is not possible

Could Alice cheat Bob?

Alice sends Bob her commitment $C = H(B \mid\mid R)$, but reveals the opposite guess, $(!B, R')$. Alice wins if she can pick R' s.t. $C' = H(!B \mid\mid R') = C$.

This fails if hash function, H, is second preimage resistant

Cryptography in Bitcoin and Blockchain

ECDSA signatures are used in Bitcoin to show proof of ownership of the outputs of a transaction!

Digital Signature Schemes Security Definitions

Similar to a handwritten signature.

Other people can verify that a message with your signature was, in fact, written by you.

Likewise, it should be difficult to forge a signature without you.

The (message, sig) recipients desire the following properties:

- Message integrity - the message hasn't been modified between sending and receiving.
- Message origin - the message was indeed sent by the original sender.
- Non-repudiation - the original sender cannot backtrack and claim they did not send the message.

Digital Signature Schemes

Digital signature scheme consists of two algorithms:

A signing algorithm, Sign, which uses a secret key, sk.

s = Sign(m, sk), s is the signature for message m.

A verification algorithm, Verify, which uses a public key, pk.

valid = Verify(Sign(m, sk), pk),

invalid = Verify(s', pk) for all s' != s.

ECDSA : Elliptic Curve Digital Signature Algorithm

ECDSA is defined by:

E: an elliptic curve.

g: a generator point of the elliptic curve with large prime order, p.

p: a large, prime integer where $g^p = O$.

H: a cryptographic hash function.

ECDSA - Setup

The signer creates:

The secret key, sk, chosen randomly from [0, …, p-1].

The public key, $pk = g^{sk}$, which should be distributed publicly.

ECDSA - Sign

Sign(m, sk):

$h = H(m)$ Hash the message

$z = h[0 : \log_2 p]$ Take the $\log_2 p$ left-most bits of h

k = randomly chosen from $[1, \ldots, p-1]$ k is kept secret

$r = \text{x-coord}(g^k) \pmod{p}$ $\text{x-coord}(P = (x, y)) = x$

$s = (z + sk \cdot r) \cdot k^{-1} \pmod{p}$

return (r, s)

The signature for our message

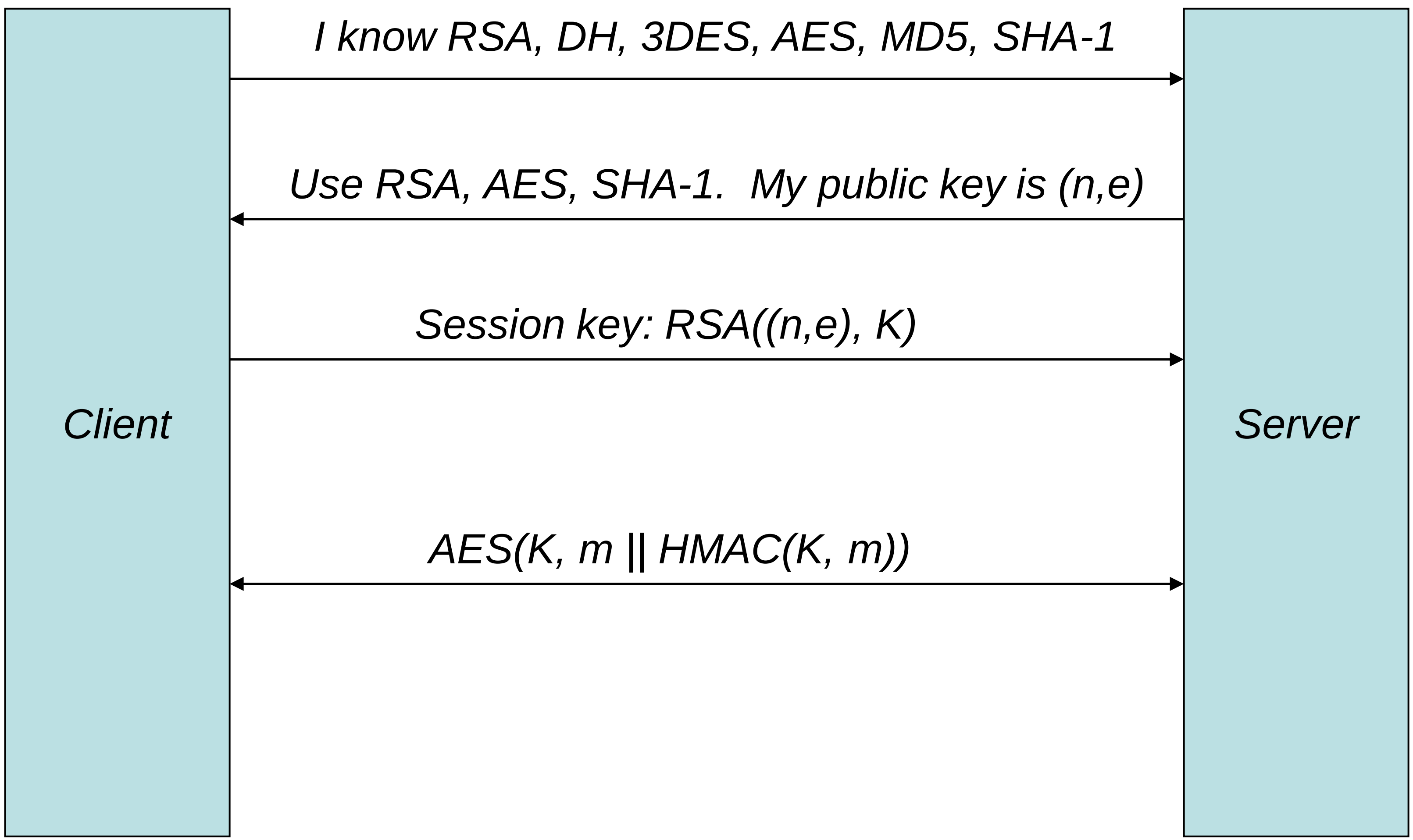
Verify This!

Secure Communication

Secure Communication - Secure Sockets Layer (SSL)

- **https protocol (secure channel)**
- **Version 3.0 developed by Netscape in 1996**
- **Also known as TLS 1.0 (Transport Layer Security)**
- **Supports many algorithms**
  - Public Key: RSA, DH, DSA
  - Symmetric Key: RC2, RC4, IDEA, DES, 3DES, AES
  - Hashes: MD5, SHA
- **Public keys are signed by CA (Certificate Authority) using X.509 certificates.**

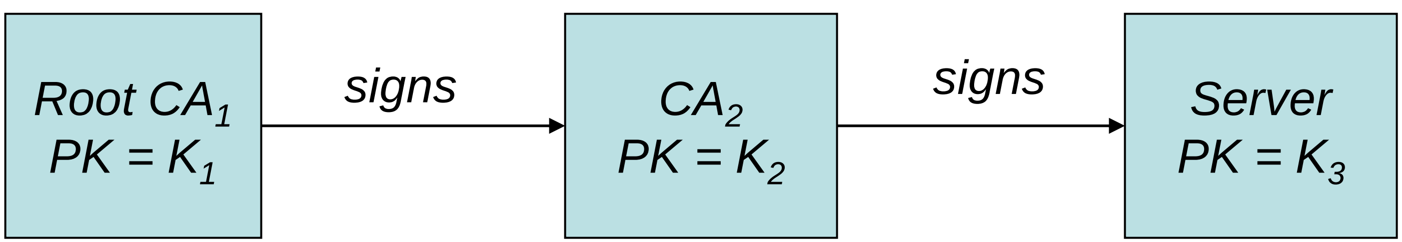
SSL Example



Here Message Auth Code:
HMAC(K, m) = SHA-1(K xor 0x5c5c…|| SHA-1(K xor 0x3c3c… || m))

X.509 Certificates

- Goal: prevent man in the middle attacks.
- Binds public keys to servers (or clients).
- Signed by a “trusted” certificate authority (CA).
- Chains to a root CA.



Key Distribution

Key Distribution

- Symmetric schemes require both parties to share a common secret key
- How to securely distribute this key
- Security system often fails due to a break in the key distribution scheme
- There are various key distribution alternatives:
  1. A can select key and physically deliver to B
  2. 3rd party can select & deliver key to A & B
  3. if A & B have communicated previously, they can use previous key to encrypt a new key
  4. if A & B have secure communications with a 3rd party C, C can relay key between A & B

Trusted Intermediaries

Symmetric key problem:

- How do two entities establish shared secret key over network?

Solution:

- Trusted Key Distribution Center (KDC) acting as intermediary between entities

Public key problem:

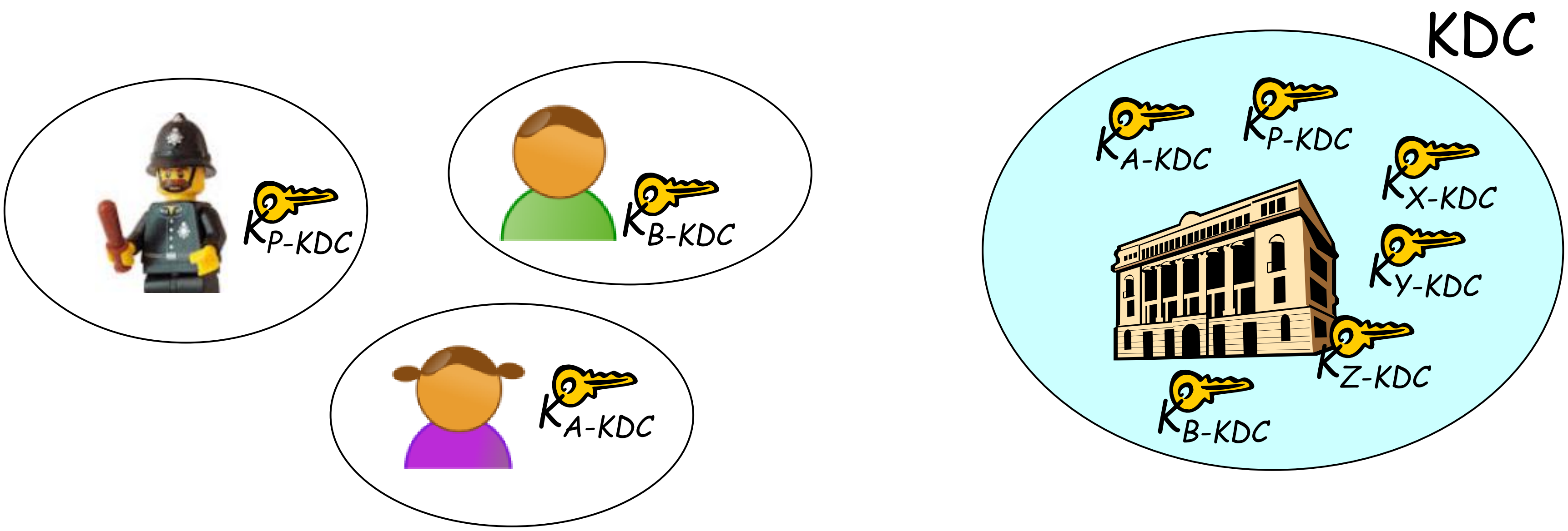
- When Alice obtains Bob's public key (from web site, e-mail, disk), how does she know it is Bob's public key, not Ted's?

Solution:

- Trusted Certification Authority (CA)

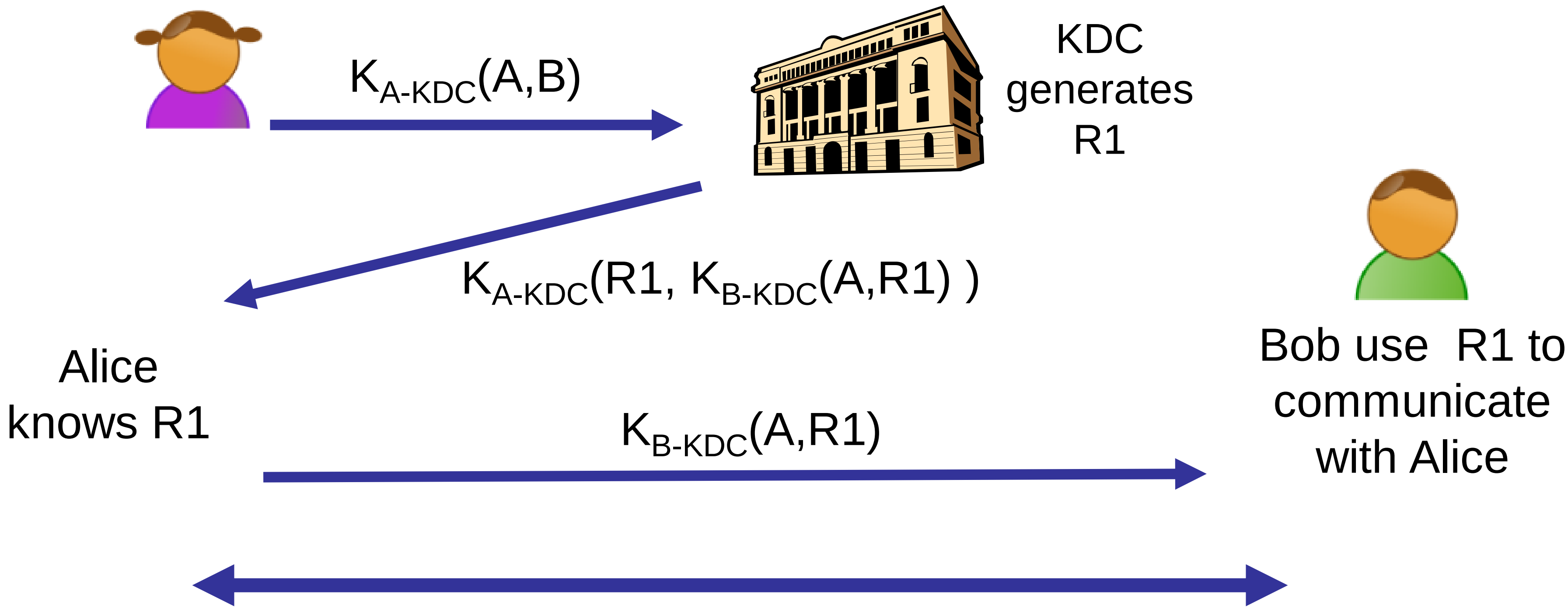
Key Distribution Center (KDC)

- Alice, Bob need shared symmetric key
- **KDC:** server shares different secret key with *each* registered user
- Alice, Bob know own symmetric keys, $K_{A\text{-}KDC}$ $K_{B\text{-}KDC}$, for communicating with KDC



Key Distribution Center (KDC)

How does KDC help Bob, Alice to determine shared symmetric secret key to communicate with each other?



Alice and Bob communicate: using R1 as session key for shared symmetric encryption

Key Management (Public Key)

- **Public-key encryption helps address key distribution problems**
- **have two aspects of this:**
  - Distribution of public keys
  - Use of public-key encryption to distribute secret keys

Distribution of Public Keys

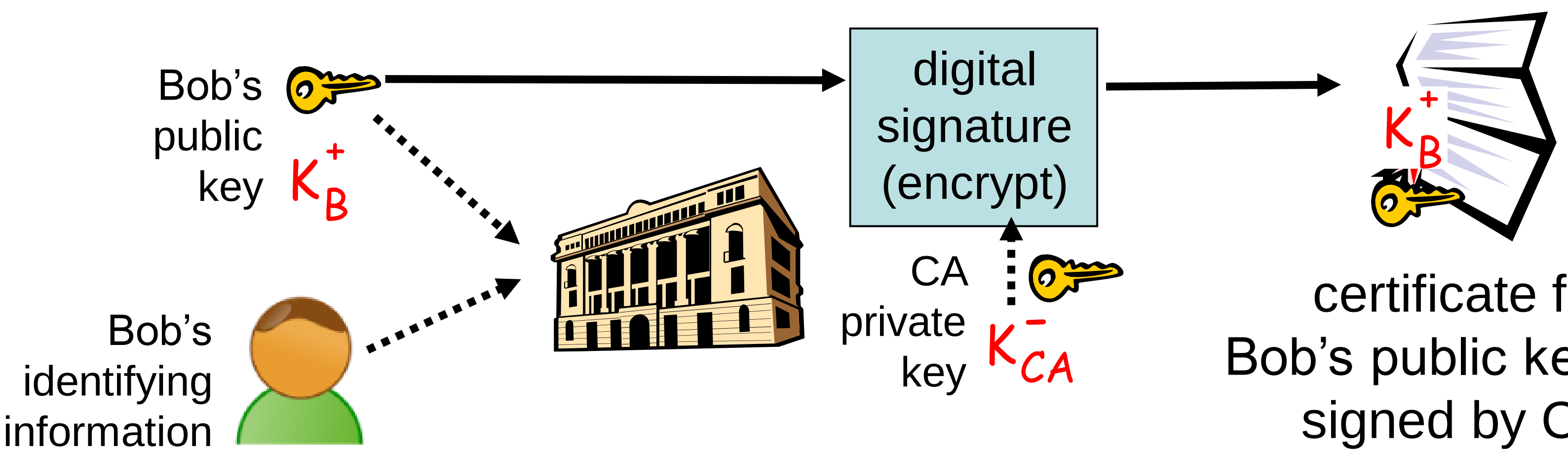
- can be considered as using one of:
  - Public announcement
  - Publicly available directory
  - Public-key authority
  - Public-key certificates

Public Announcement

- **Users distribute public keys to recipients or broadcast to community at large**
  - e.g. append PGP keys to email messages or post to news groups or email list
- **Major weakness is forgery**
  - Anyone can create a key claiming to be someone else and broadcast it
  - Until forgery is discovered can masquerade as claimed user

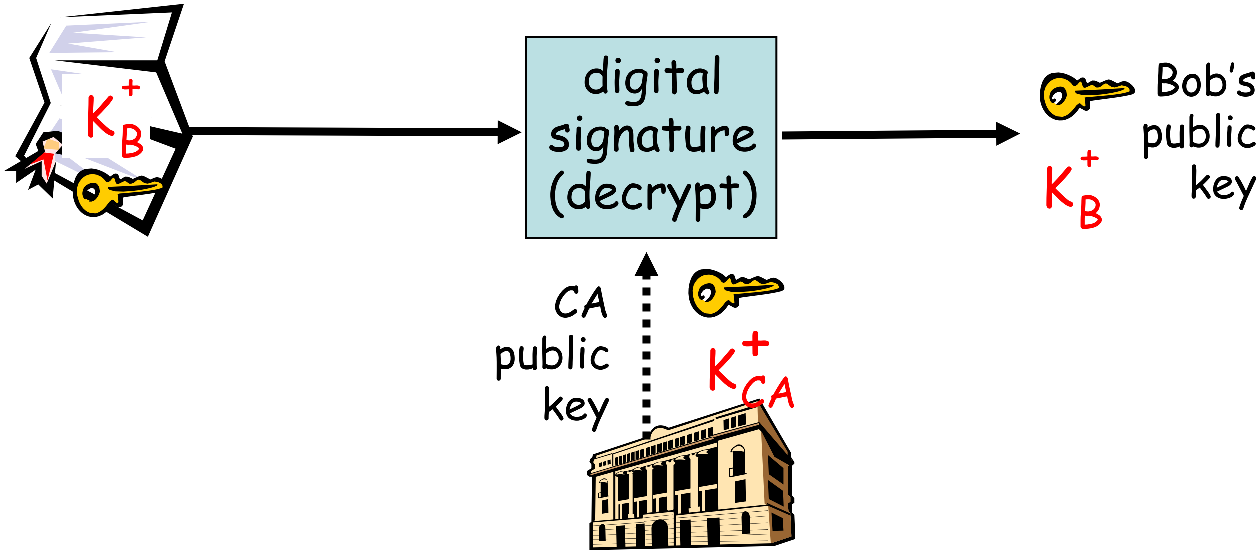
Certification Authorities

- **Certification authority (CA): binds public key to particular entity E**
- **E (person, router) registers its public key with CA.**
  - E provides "proof of identity" to CA
  - CA creates certificate binding E to its public key
  - Certificate containing E's public key digitally signed by CA: CA says "this is E's public key"



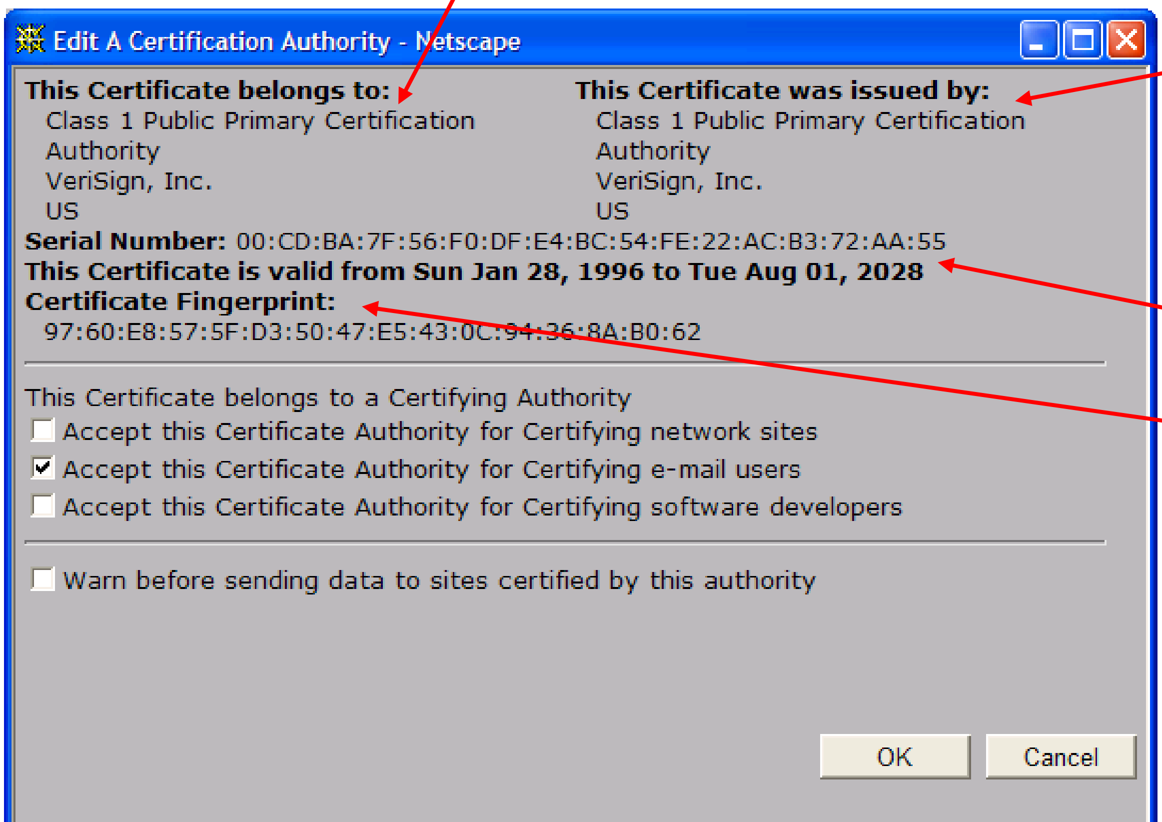
Certification Authorities

- When Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key



A certificate contains:

- Serial number (unique to issuer)
- info about certificate owner, including algorithm and key value itself (not shown)



- info about certificate issuer
- valid dates
- digital signature by issuer

Decentralized Identifiers (DIDs) v1.0

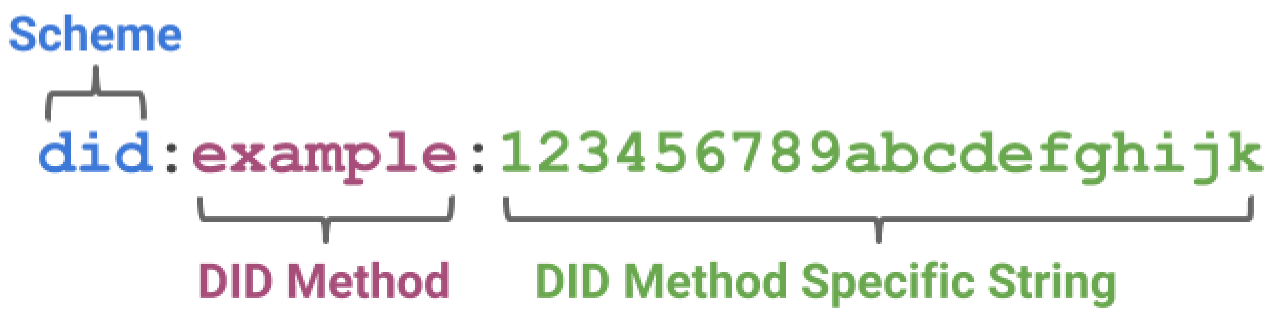
- A Decentralized Identifier (DID) is an identifier that is globally unique, resolveable with high availability, and cryptographically verifiable.
- DIDs are typically associated with cryptographic material, such as public keys, and service endpoints, for establishing secure communication channels.
- DIDs are useful for any application that benefits from self-administered, cryptographically verifiable identifiers such as personal identifiers, organizational identifiers, and identifiers for IoT scenarios.
- This *decentralized public key infrastructure (DPKI)* could have as much impact on global cybersecurity and cyberprivacy as the development of the SSL/TLS protocol for encrypted Web traffic (now the largest PKI in the world).

https://www.w3.org/TR/did-core/ W3C Working Draft 25 February 2021

Compare with Other Globally Unique Identifiers

- The need for globally unique identifiers that do not require a centralized registration authority is not new. **UUID**s (Universally Unique Identifiers, or GUIDs, Globally Unique Identifiers) developed 1980s as IETF RFC 4122.
- The need for persistent identifiers (assigned once and never change) is not new. Standardized as **URN**s (Uniform Resource Names) by IETF RFC 2141 and RFC 8141.
- UUIDs are not globally resolvable and URNs – if resolvable – require a centralized registration authority. Neither UUIDs or URNs inherently address a third characteristic – the ability to **cryptographically verify ownership of the identifier**.
- For **self-sovereign identity**, which can be defined as a lifetime portable digital identity that does not depend on any centralized authority, DID fulfills all four requirements: <u>persistence, global resolvability, cryptographic verifiability, and decentralization</u>

Format of a DID



- DID infrastructure is a global key-value database in which the database is all DID-compatible blockchains, distributed ledgers, or decentralized networks.
- In this virtual database, the key is a DID, and the value is a DID document.
- The purpose of the DID document is to describe the public keys, authentication protocols, and service endpoints necessary to bootstrap cryptographically-verifiable interactions with the identified entity.
- DIDs and DID documents can be adapted to any blockchains capable of resolving a unique key into a unique value - public, private, permissionless, or permissioned.

Decentralized DNS (DDNS)

	DDNS	Traditional DNS
Technology architecture	P2P distributed technology	Centralized tree technology
Registration fee	Low cost	High cost
Domain name operation	Blockchain wallet	Centralized system
Preventing attacks	51% attack required	Only needed to attack from 1%
Domain name interception	Difficult to intercept	Easy to intercept
Domain name blocking	The domain name can be replaced by anti-blocking	The domain name can be blocked by the blacklist
Domain name management	No one has the right to freeze the domain name	Authorities can stop serving the domain name
Privacy protection	The nickname of a domain name is hard to guess	Domain name has record violation privacy

Assignment 1 - Due in 3 Weeks (Mar 19, 2021)

1. Write a program or go to the reference web site to hash the phrase: “Hello, world!*” with a number of appended to generate 4 leading “0”s

2. Show the correctness of the verification model of ECDSA signature scheme.

3. Experience Blockchain: (1) Install a Crypto Wallet on your PC and/or mobile phone; (2) Make some transactions (e.g. purchase something or exchange with a friend), and show the transaction record as proof; (3) List three areas of improvement for the wallet software that you use.

Assignment 1 – Problem 1: Hash Function

Reference website:
https://www.xorbin.com/tools/sha256-hash-calculator

Please hash the phrase: “Hello, world!” with a number appended. For example, you can input the following phrases to see the hash result:

Hello, world!0
Hello, world!1
…..
Hello, world!978
…..
Can you
- generate 1 leading “0” …
- generate 2 leading “0” …
- generate 3 leading “0”s …
- generate 4 leading “0”s …

Thank you