Cryptographic tools for decentralized systems

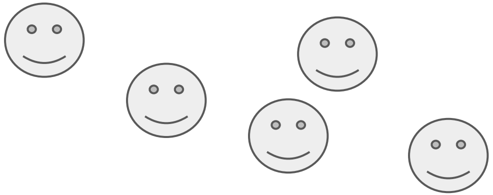
CS-438: Decentralized Systems Engineering

(slide credits: **Haoqian Zhang**, Kirill Nikitin, Dusan Kostic, Cristina Basescu, Bryan Ford)

Alice sends a message to Bob...

Secrecy? Integrity? Authenticity?

Share a secret with multiple parties but trust no individual (Byzantine attacker model)



Threshold secret sharing

Introduction

- What is cryptography?
  - A toolbox for many security mechanisms
  - Information security and communication security
  - Secure data at rest and data in motion
- Cryptography is not:
  - The solution to all security problems
  - Reliable unless implemented properly
  - Reliable unless used properly
  - Something you should try to invent yourself

Credit: https://crypto.stanford.edu/cs155/lectures/07-crypto.pdf

Outline

- **Shared-algorithm cryptography**
- Symmetric-key cryptography
- Public-key cryptography
- Cryptographic hash functions
- Key infrastructure
- Threshold secret sharing

Naive Approach

- Two parties agree on an encryption algorithm (e.g., rot13) and keep it secret
  - 50BC - Caesar's Cipher - substitution
- Use it to encrypt messages to each other
- 1883. Kerckhoffs' Principle – A cryptosystem should be secure even if everything about the system, except the key, is public knowledge

One-Time Pad

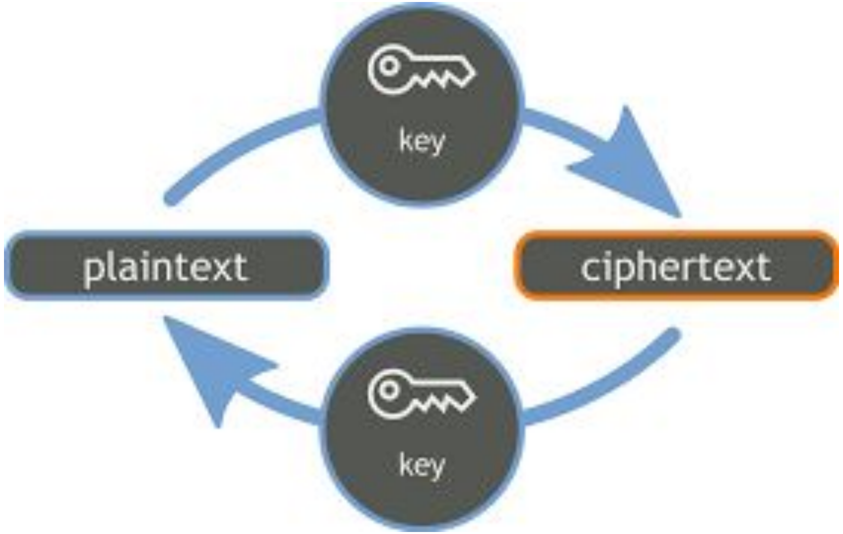
- First described by Miller in 1882, then reinvented by Vernam in 1917
- One-time pad (OTP)
  - c = Encryption(m) = m XOR key
  - m = Decryption(c) = c XOR key
  - Key is a random string at least as long as the plaintext
- Provides “perfect” secrecy in principle
- Practical disadvantages:
  - Keys must not be used more than once and must be truly random and uniform
  - Key length depends on the message length

Outline

- Shared-algorithm cryptography
- **Symmetric-key cryptography**
- Public-key cryptography
- Cryptographic hash functions
- Key infrastructure
- Threshold secret sharing

Definition

- Encryption of plaintext and decryption of ciphertext are done using a well-known algorithm and the same key, hence *symmetric* crypto

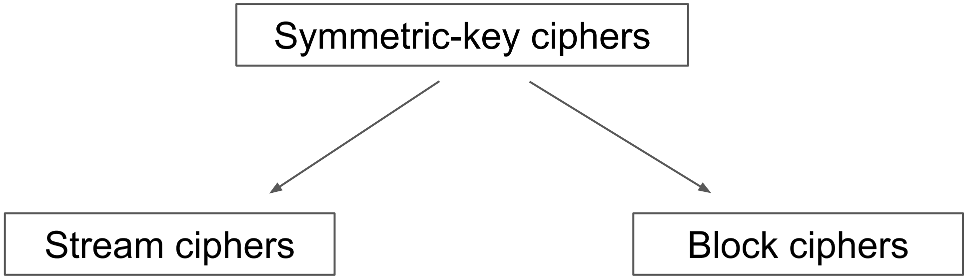


Characteristics

✅ An algorithm is normally public, so anyone can analyze it and try to find flaws

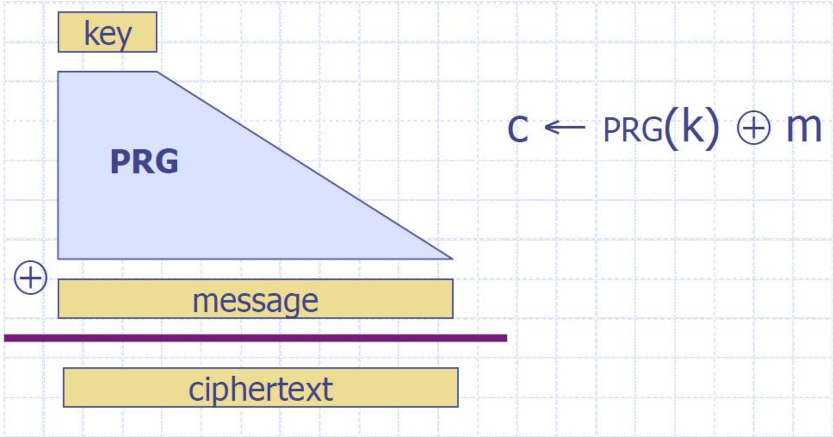
❓ Key size: as computers get faster, key sizes have to increase

- DES (1976) used 56-bit keys - brute force search now feasible



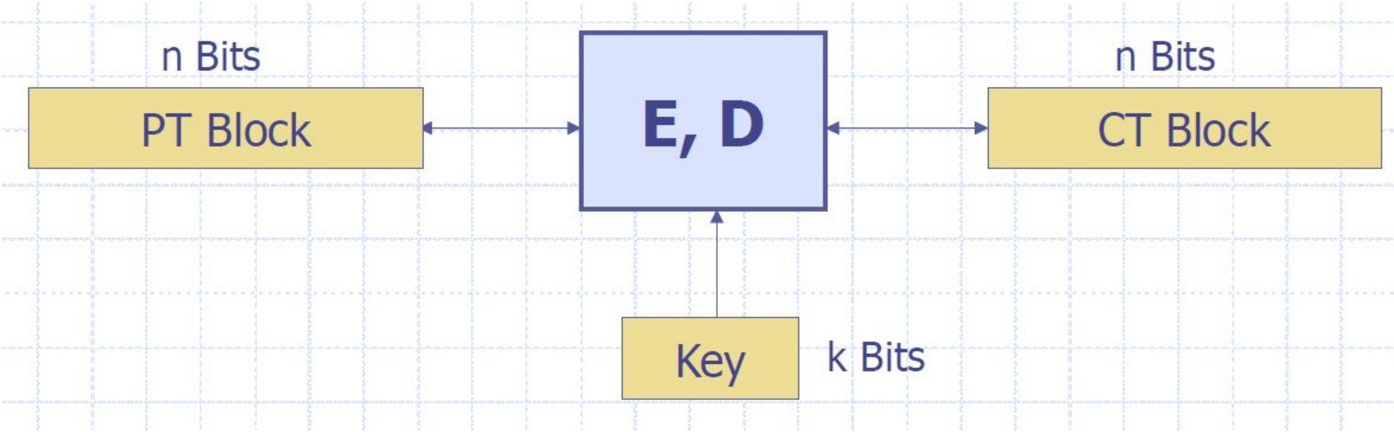
Stream Ciphers

- Making OTP practical (and less secure)
- Require Pseudo Random Generators
- Examples: RC4 (was used in HTTPS and WEP), CSS (DVD encryption), E0 (Bluetooth), A5/1,2 (GSM encryption), Salsa20/ChaCha, …



Block Ciphers

- Encrypt blocks of data of fixed size
- Modes of operation handle variable length data

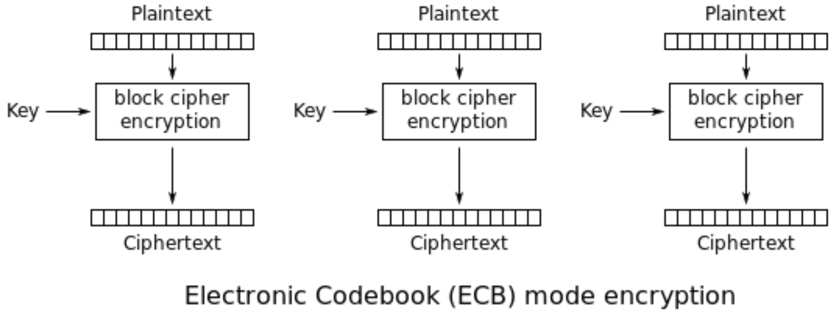


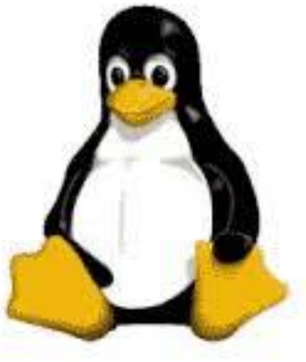
Examples of Block Ciphers

- Data Encryption Standard (DES):
  - Block size 64 bits
  - Key size 56 bits
  - Deprecated
- Advanced Encryption Standard (AES):
  - Block size 128 bits
  - Key size 128/192/256 bits
  - Hardware support in Intel and AMD processors

Modes of Operation - Electronic Code Book

- Electronic Code Book (ECB)
- Example of a *bad* mode of operation (insecure, obsolete): same plaintext blocks encrypt to same ciphertext blocks





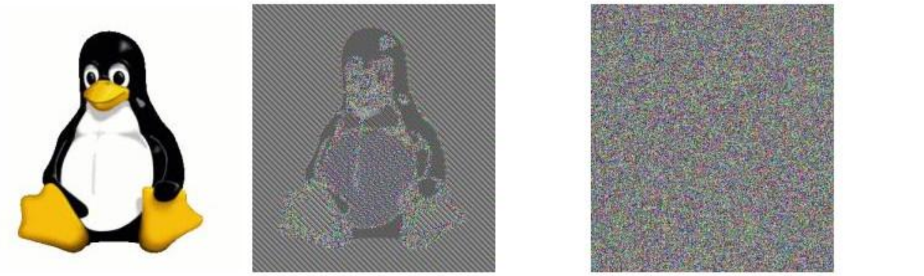
Original

ECB-encrypted image

en.wikipedia.org

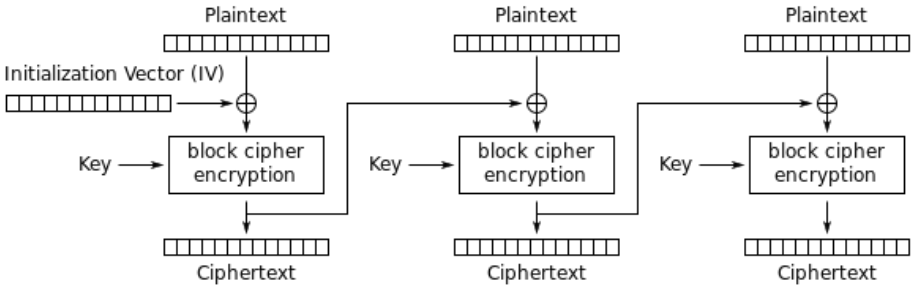
Modes of Operation - Cipher Block Chaining

- Cipher Block Chaining (CBC)
- A secure mode of operation (when used correctly)



Original — Encrypted using ECB mode — Modes other than ECB result in pseudo-randomness

en.wikipedia.org

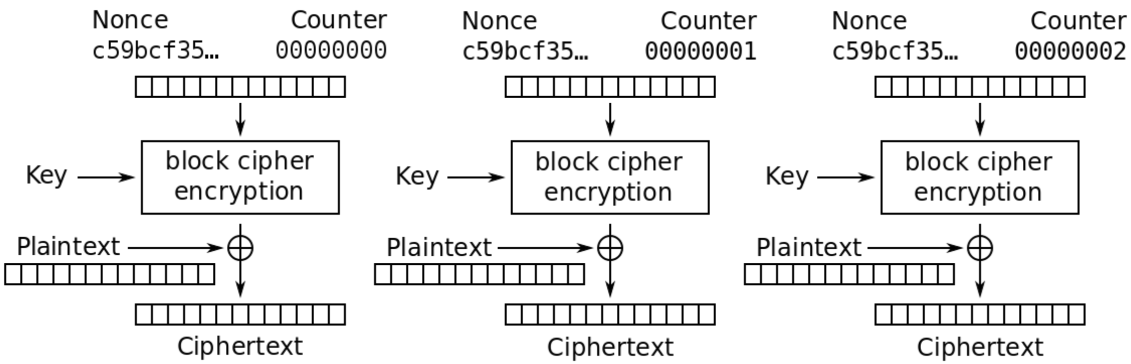


Cipher Block Chaining (CBC) mode encryption

Modes of Operation - Counter Mode (CTR)

- Somewhat turns a block cipher into a stream cipher
- Generates the next keystream block by encrypting values of a "counter"

➔ (+) Parallelizable, software and hardware efficient, random access to blocks, simplicity, message of arbitrary bit length



Counter (CTR) mode encryption

Figure credit: [Wikipedia]

Outline

- Shared-algorithm cryptography
- Symmetric-key cryptography
- **Public-key cryptography**
- Cryptographic hash functions
- Key infrastructure
- Threshold secret sharing

Public-key Cryptography

- Solves the problem of having to agree on a pre-shared key
- (Public, private) key pair instead
- Public & private key mathematically related
  - uses large-number arithmetic
  - relies on computational assumptions believed to be difficult
- "Owner" of identity holds private key secret, distributes public key to communication partners

First Approach of Asymmetric Crypto

- 1975. Diffie and Hellman in “New directions in cryptography” describe the idea of asymmetric (public key) cryptography:

We stand today on the brink of a revolution in cryptography. The development of cheap digital hardware has freed it from the design limitations of mechanical computing and brought the cost of high grade cryptographic devices down to where they can be used in such commercial applications as remote cash dispensers and computer terminals.

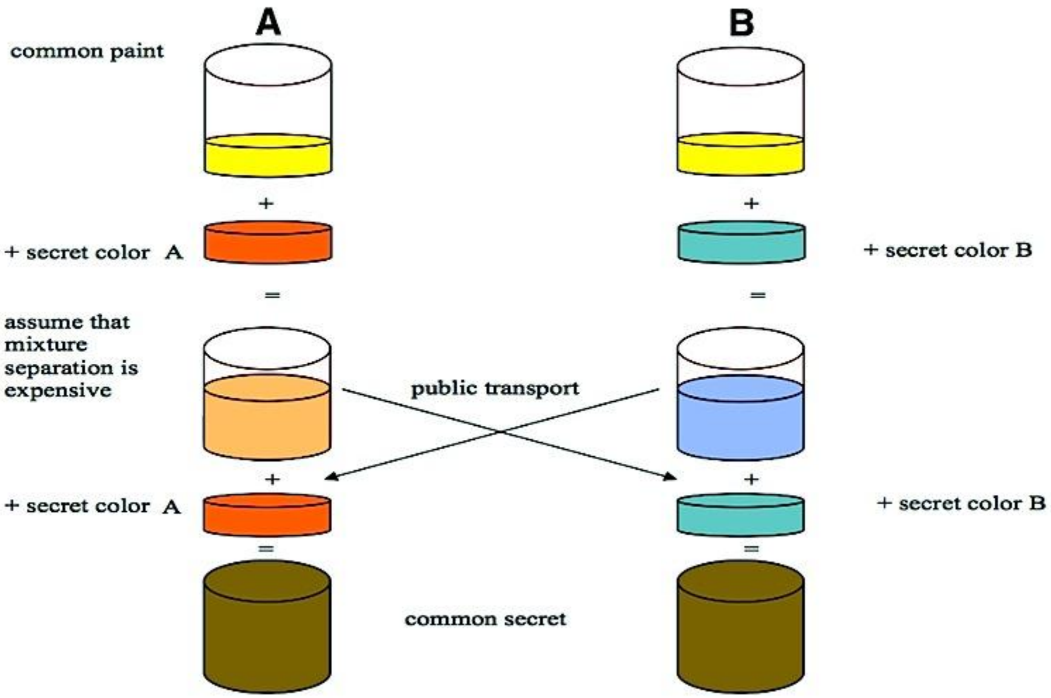
*In turn, such applications create a need for new types of cryptographic systems which **minimize the necessity of secure key distribution channels and supply the equivalent of a written signature.** At the same time, theoretical developments in information theory and computer science show promise of providing provably secure cryptosystems, changing this ancient art into a science.*

Primitives

- Public and private key
  - Two keys (numbers), *public* is distributed widely, *private* is kept secret
  - Easy: f(private) -> public
  - Hard: f(public) -> private
- Encryption and Decryption
  - Encrypt with the public key, decrypt with the private key
  - Hard to decrypt without the private key
- Digital signatures
  - Sign with the private key, verify the signature using the public key
  - Hard to create a signature if only public key is known
- Interactive key exchange
  - Create a shared *secret* over an insecure communication channel

Interactive Key Exchange

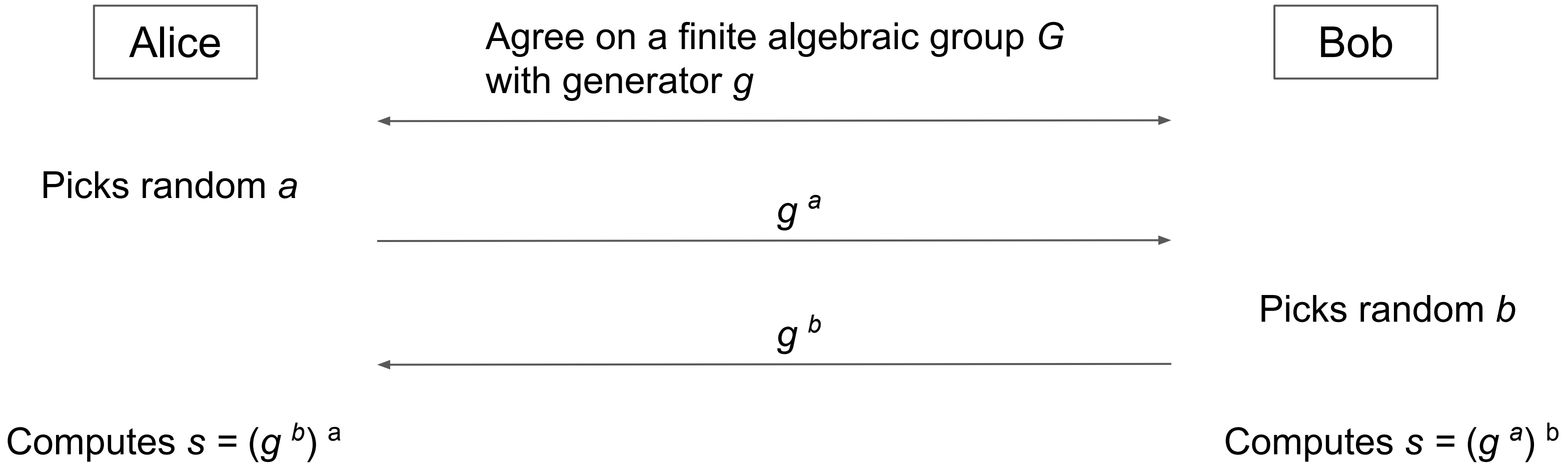
- Diffie-Hellman Key Exchange



Credit: A.J. Han Vinck, Introduction to public key cryptography

EPFL

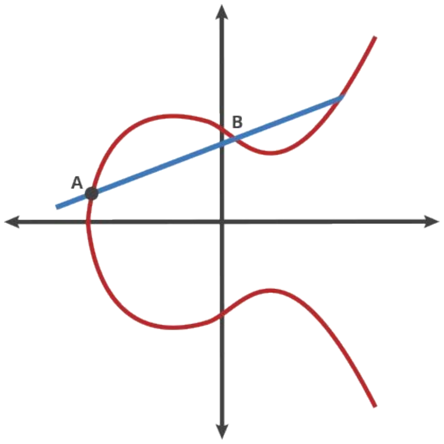
Diffie-Hellman Key Exchange



- Security of DH relies on hardness of the Discrete Logarithm Problem (DLP)
- The DLP is hard not in all groups – must choose appropriately
- Application example: the Handshake protocol in TLS

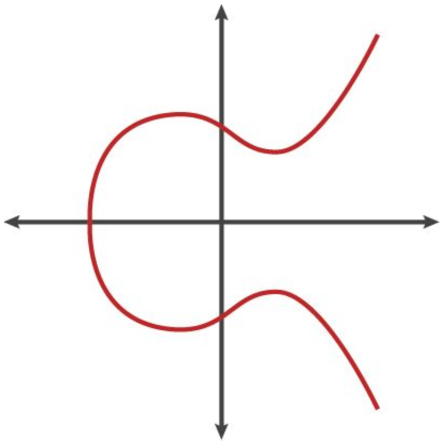
Elliptic Curve Cryptography

- Elliptic curve cryptography (ECC) is based on the algebraic structure of elliptic curves over finite fields.
  - General elliptic curve: $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$
  - Set of points satisfying an equation with degree two in one of the variables and three in the other
- Hardness (Trapdoor)
  - Determining n from Q = nP given known Q and P if n is large is hard
  - Computing Q is easy



Elliptic Curve Cryptography

- Elliptic curve cryptography (ECC) is based on the algebraic structure of elliptic curves over finite fields.
    a. General elliptic curve: $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$
    b. Montgomery curve: $y^2 = x^3 + ax^2 + x$
- Smaller keys for equivalent security than traditional crypto (e.g., 256-bit for ECC comparable to 2048-bit RSA)
    a. Faster operations
    b. Smaller public keys -> smaller certificates and less data
- Popular secure curves with known generation parameters:
    a. Curve25519
    b. Curve448



Elliptic Curve Diffie-Hellman (ECDH)

- Diffie-Hellman key exchange on an elliptic curve
- Steps
  - Alice and Bob agree on a curve with a base point G that generates a subgroup of order n
  - Alice picks a random d_A and sends $H_A = d_A G$ to Bob
  - Bob picks a random d_B and sends $H_B = d_B G$ to Alice
  - They compute $S = d_A H_B = d_A(d_B G) = d_B(d_A G) = d_B H_A$

Elliptic Curve Digital Signature Algorithm (ECDSA)

- A variant of the Digital Signature Algorithm (DSA) using elliptic curve crypto
- For 80-bit security (2^{80} operations), ECDSA public key – 160 bits, DSA – 1024 bits; whereas signature size is the same
- Used in Bitcoin to authenticate transactions and every Bitcoin address is a cryptographic hash of an ECDSA public key
- iMessages and iCloud keychain syncing use ECDSA

Elliptic Curve Digital Signature Algorithm

Steps

- Compute hash of message m and truncate it to z to be the same bit length as order n.
- Select a **cryptographically secure random** integer k from $[1,n-1]$.
- Calculate the curve point $P = kG$.
- Calculate the number $r = x_P \bmod n$ (where x_P is the x coordinate of P); $r \mathrel{!}= 0$.
- Calculate $s = k^{-1}(z + rd_A) \bmod n$.

The pair **(r,s)** is the signature. To verify

- $u_1 = s^{-1}z \bmod n$, $u_2 = s^{-1}r \bmod n$, $P = u_1G + u_2H_A$
- The signature is valid only if $r = x_P \bmod n$

Elliptic Curve Digital Signature Algorithm

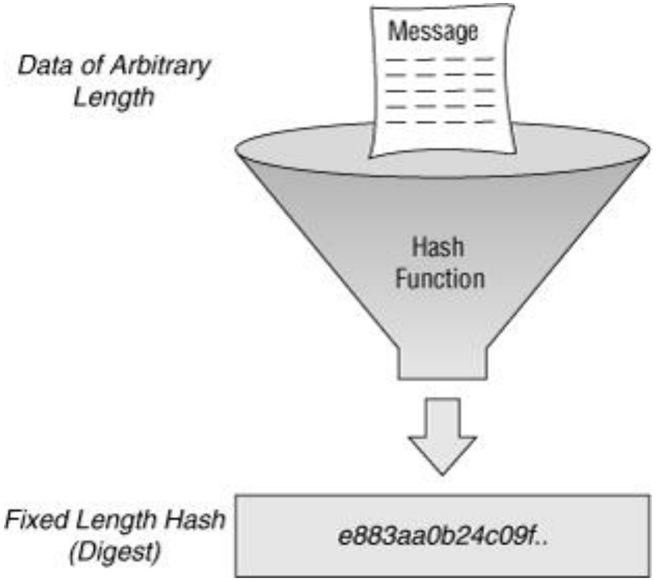
- Requires random or unpredictable data as input to generate *k*, different *k* for different signatures
- 2010: Recovery of the ECDSA private key used by Sony to sign software for the PlayStation 3 due to static *k*
- 2013: Loss of funds in Android Bitcoin Wallet due to *k* by a faulty random number generator

Outline

- Shared-algorithm cryptography
- Symmetric-key cryptography
- Public-key cryptography
- **Cryptographic hash functions**
- Key infrastructure
- Threshold secret sharing

Cryptographic hash functions

- Map bit-strings of any length to a fixed-length output in a deterministic way
- Desirable properties of hash functions (informal definitions):
    - **One-way:** given y it is infeasible to find any x such that y = h(x)
    - **Collision-resistance:** infeasible to find x and x' such that h(x)=h(x')
    - **Pseudo-randomness:** indistinguishable from a random oracle



[credit: Cisco]

Examples of Usage

- Password storage
- Files/Messages integrity verification
- Key derivation
- Proof-of-work
- Blockchains
- ...

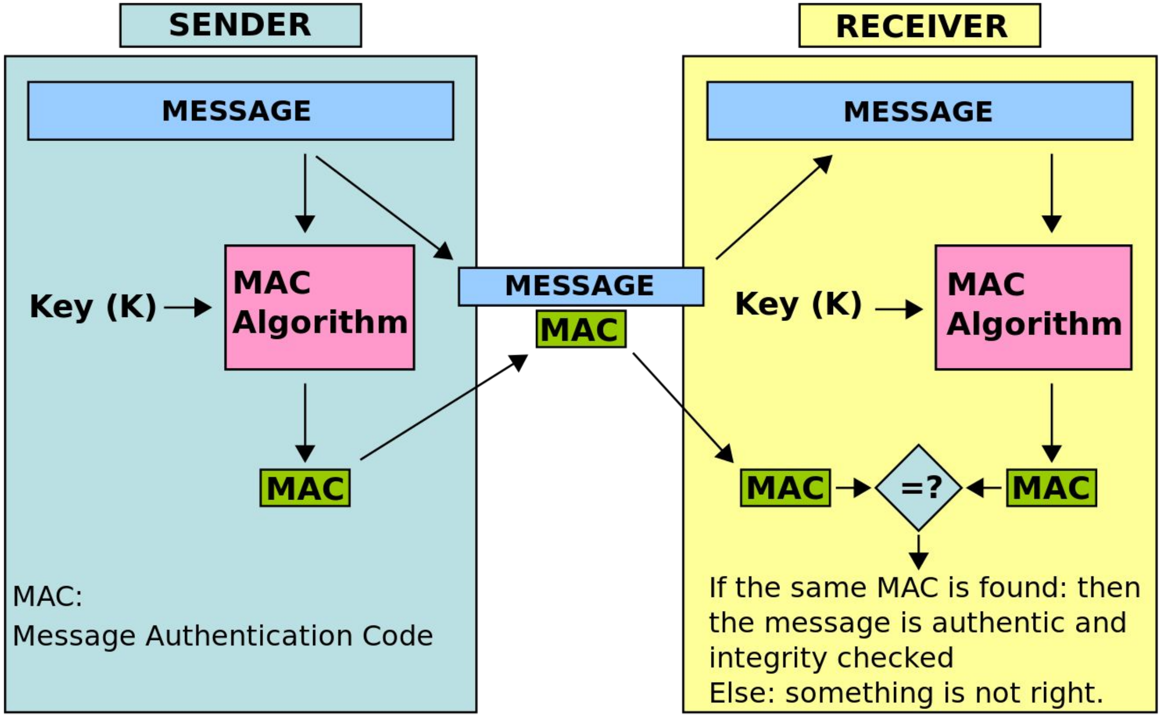
Usage for Integrity

- Create a small (constant-size) digest of an arbitrarily large message
- Knowing the digest, one can verify that the message matches (recall that cryptographic hash functions are collision-resistant so one is unable to find another message that hashes to the same digest)

Keyed Hash Functions for Authentication

- Usually to prove message authenticity between two parties who share a key
- Takes an input message and a key, yields a message digest that depends on both
- Hard to derive message or key from resulting hash
- Hard to find any relationship between hashes with different keys
- Naive implementation: just use unkeyed hash on key + message, for subtle cryptographic reasons, not the best design

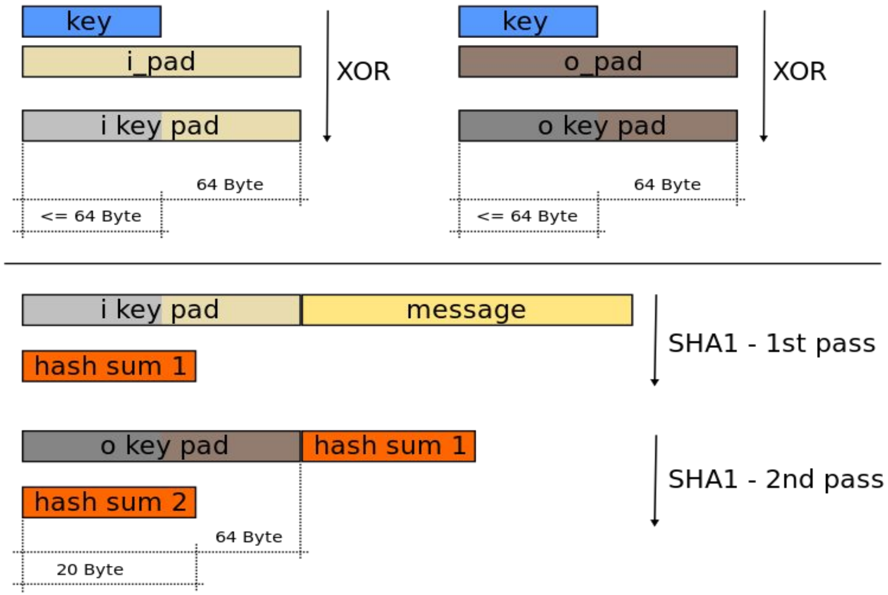
Message Authentication Code(MAC)



Credit: Wikipedia

Hash Message Authentication Code (HMAC)

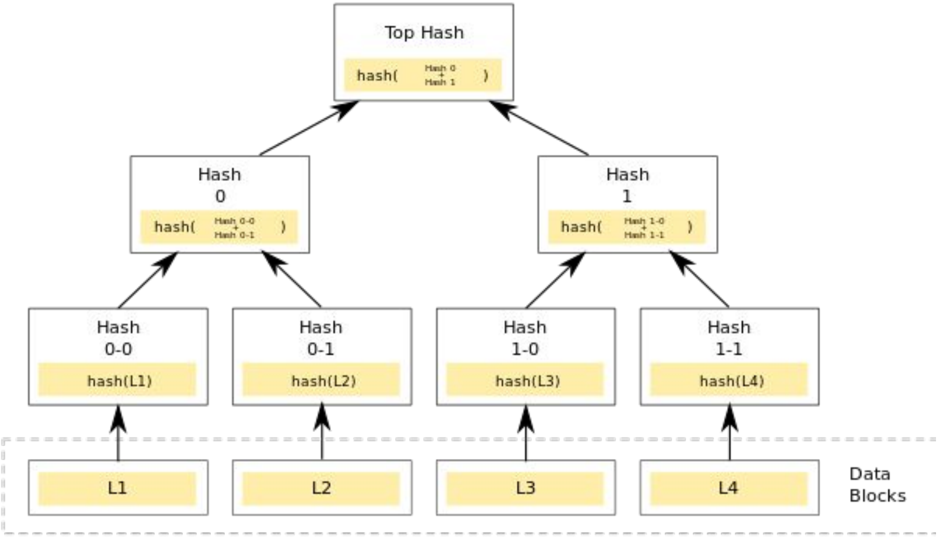
- Examples: HMAC_SHA256, HMAC_SHA1, …
- Used in TLS, IPsec, …



Credit: Wikipedia

Merkle Tree

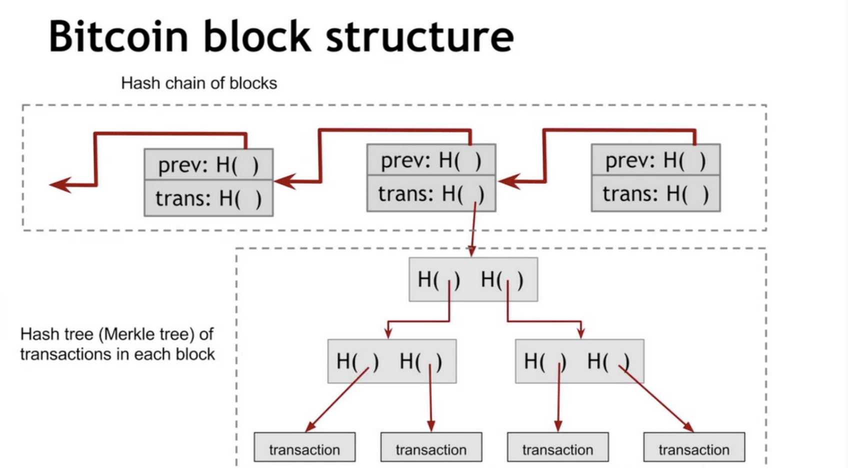
- Hash-based data structure for efficient summarizing and verifying the integrity of large sets of data
- Useful for ensuring integrity of stored data and data in transmission
- Every leaf node – data block, non-leaf node – the crypto hash of its children



Credit: Wikipedia

Merkle Tree

- Inclusion of data (leaf node) is verifiable in time proportional to the logarithm of the number of tree leaf nodes
- Possible to verify inclusion of a block *without* knowing the other data blocks
- Used in IPFS, BitTorrent protocol, Git, Apache Cassandra, Bitcoin, …



Source: btc-investor.net

Outline

- Shared-algorithm cryptography
- Symmetric-key cryptography
- Public-key cryptography
- Cryptographic hash functions
- **Key infrastructure**
- Threshold secret sharing

Key Distribution

- For both symmetric and asymmetric crypto we have to distribute keys
- Symmetric cryptosystems require the exchange of secret keys
  - Need for a secret/confidential channel
- Asymmetric cryptosystems require the exchange of public keys
  - Need for a trusted/integrity protected channel
- Authorities trusted to provide secret / trustworthy keys:
  - Key Distribution Centers (KDC)
  - Certification Authorities (CA)

Using Hierarchy of Trust

- One KDC/CA is not enough to serve all users
- KDCs/CAs are organized into hierarchies or peer networks

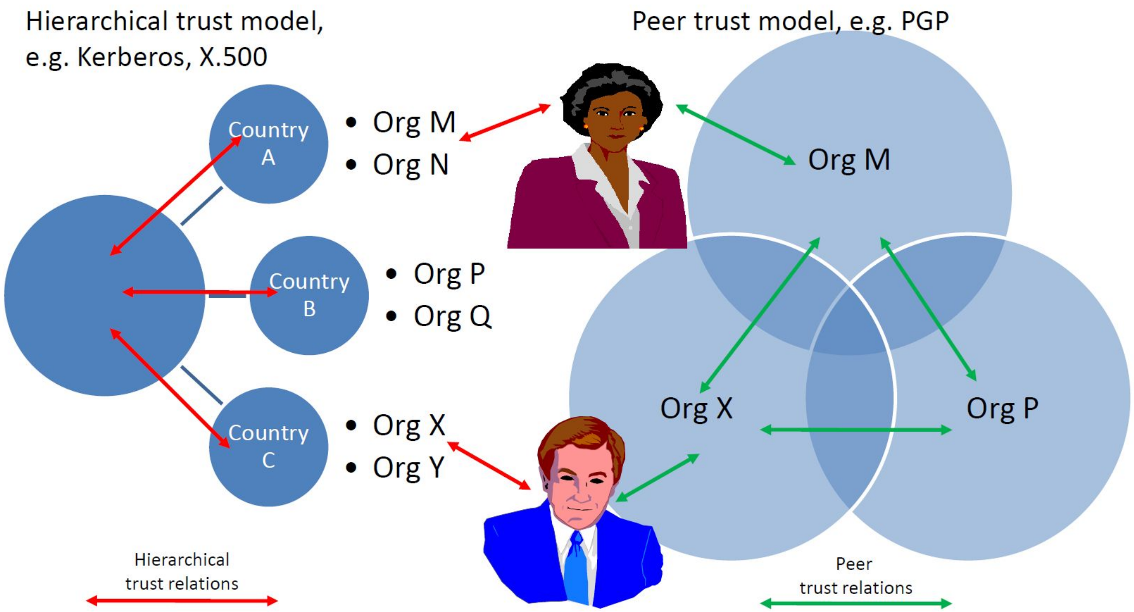


Figure Credit:
P. Janson "IT Security Engineering" course

Public Key Infrastructure (PKI)

PKI binds *public keys* to their *owners*

- Certificate Authority (CA): stores, issues and signs the digital certificates
- Root certificate public keys are embedded in web browser
- Few Root CAs sign "delegation" certificates declaring that other CAs are also trusted to sign server certs
- Subsidiary "issuing" CA signs a certificate for, e.g., Google servers
- When your browser connects to Google server via SSL, the server sends its server-side certificate and the "chain" of signatures down from the root CA

Attacks

- Huge problem when CA gets compromised (Comodo, DigiNotar)


theguardian

UK world sport football opinion culture business lifestyle fashion environment tech travel ≡ browse all sections

home › tech

Hacking

DigiNotar SSL certificate hack amounts to cyberwar, says expert

Dutch government revokes certificates used for all its secure online transactions, while CIA, Google, Microsoft and others affected by hack called 'worse than Stuxnet'

This article is 5 years old

0 10

Charles Arthur and agencies

@charlesarthur

Monday 5 September 2011 18.14 BST

Aanpak DigiNotar-problematiek

Voor publieksvragen over DigiNotar bel 0800 1351

The Dutch government has revoked all trust in digital certificates issued by DigiNotar

Most popular in US

Minnesota Vikings' new glass-plated stadium becomes 'death trap' for birds

Uber CEO Travis Kalanick caught on video arguing with driver about fares

Increased risk of 11 types of cancer linked to being overweight, researchers warn

Detained by US immigration: 'In that

Methods Used for PKI

- Certificate Authorities (CAs):
- Web of Trust (WOT):
  - PGP, GnuPG
  - Self-signed certificates

Outline

- Shared-algorithm cryptography
- Symmetric-key cryptography
- Public-key cryptography
- Cryptographic hash functions
- Key infrastructure
- **Threshold secret sharing**

But Who Holds the Keys?

Any encrypted data is secured with a *private key*

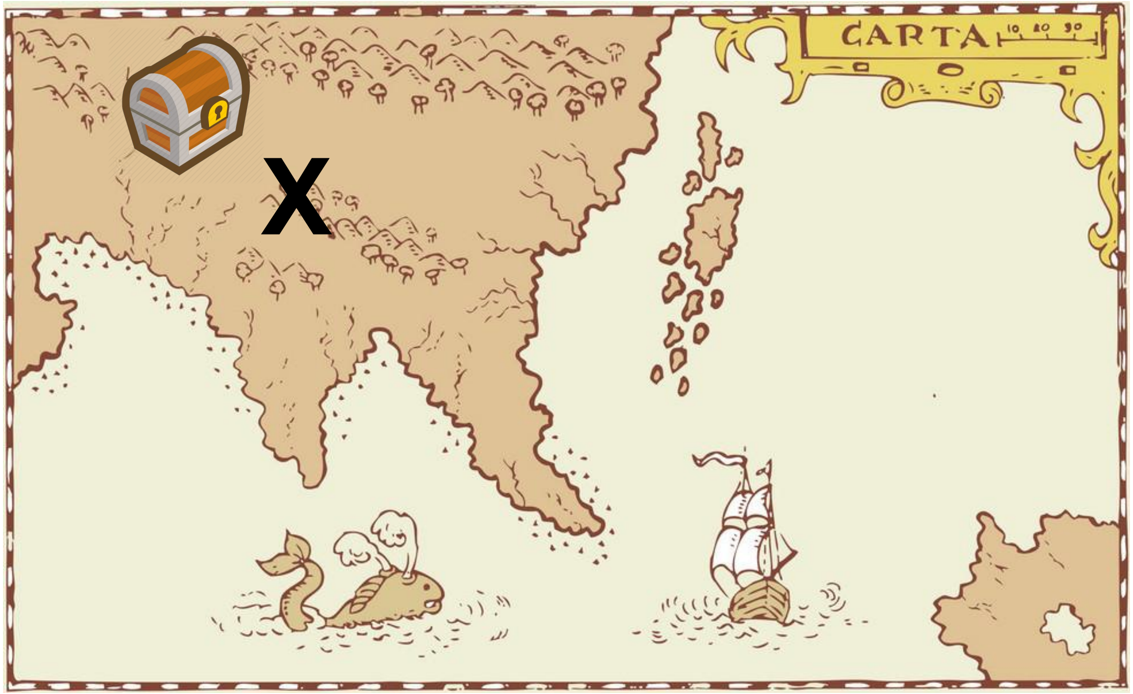
- A private key is *just information* (a number)!
- If the *key* leaks, anyone can decrypt the data
  - Regardless of where it’s stored: cloud, blockchain…

Privacy & Accountability with secret-sharing

- Essential idea: after encrypting data,
  “deal” the secret key to a *threshold* ***t*** of ***n*** parties

Secret Sharing: Illustration

Suppose you're a pirate & bury your treasure…

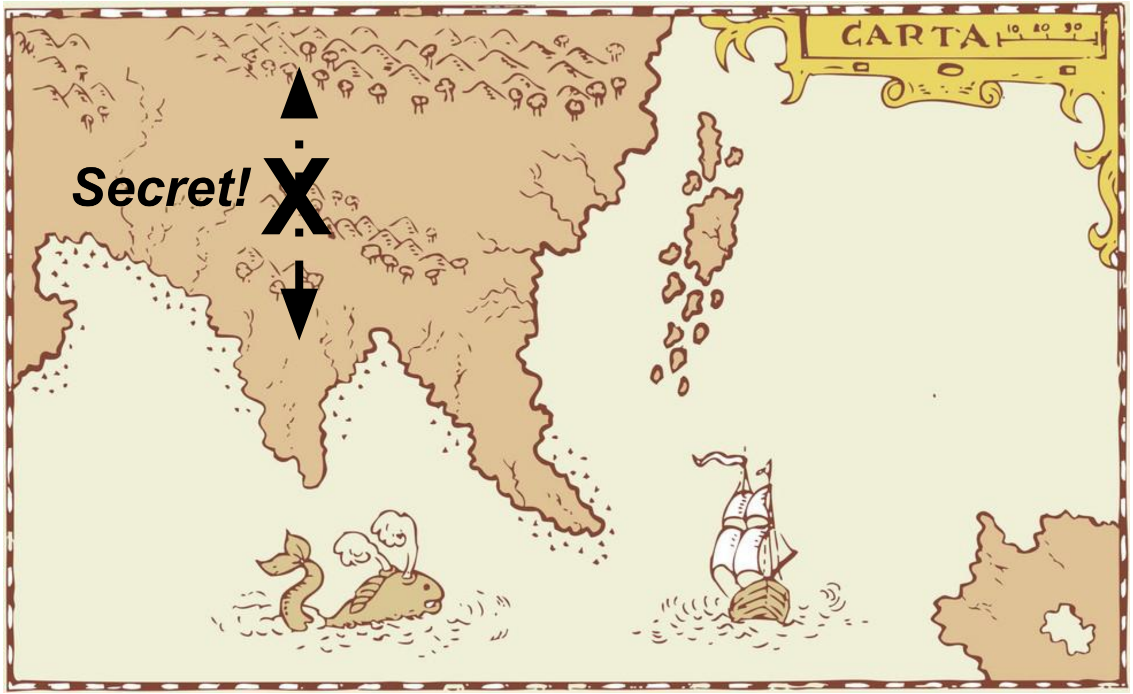


Keeping the Location Secret

You have 3 henchmen who you want to send back for it later, but you don’t trust *any one* completely

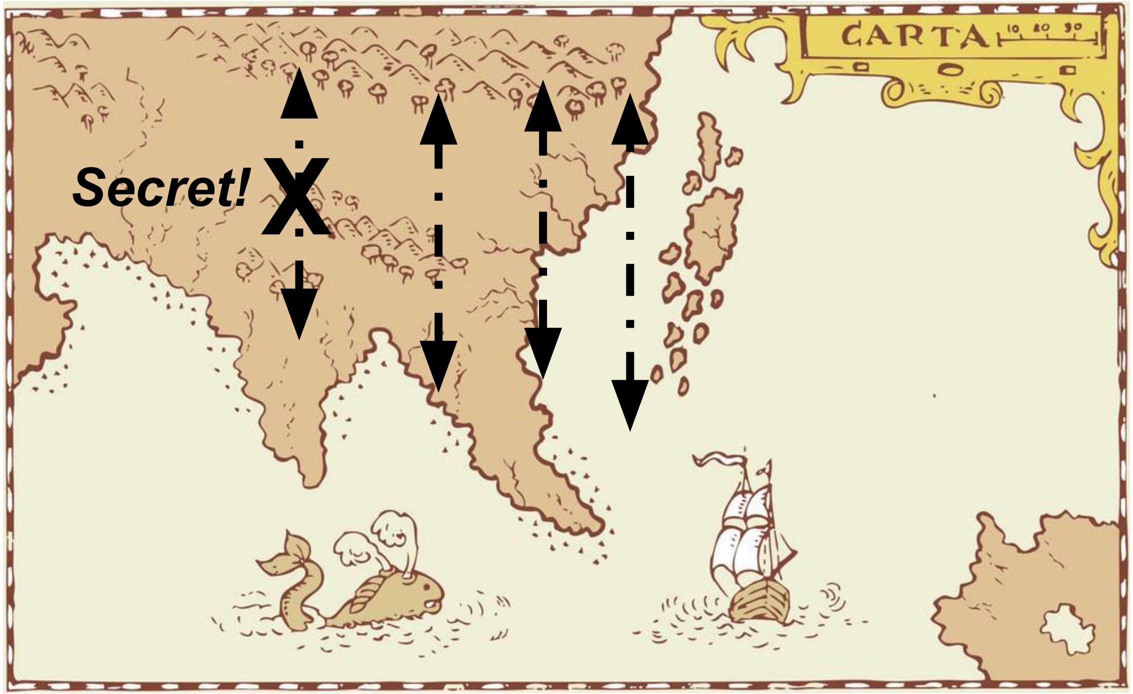
Secret Sharing: Illustration

You mark the spot between two reference points



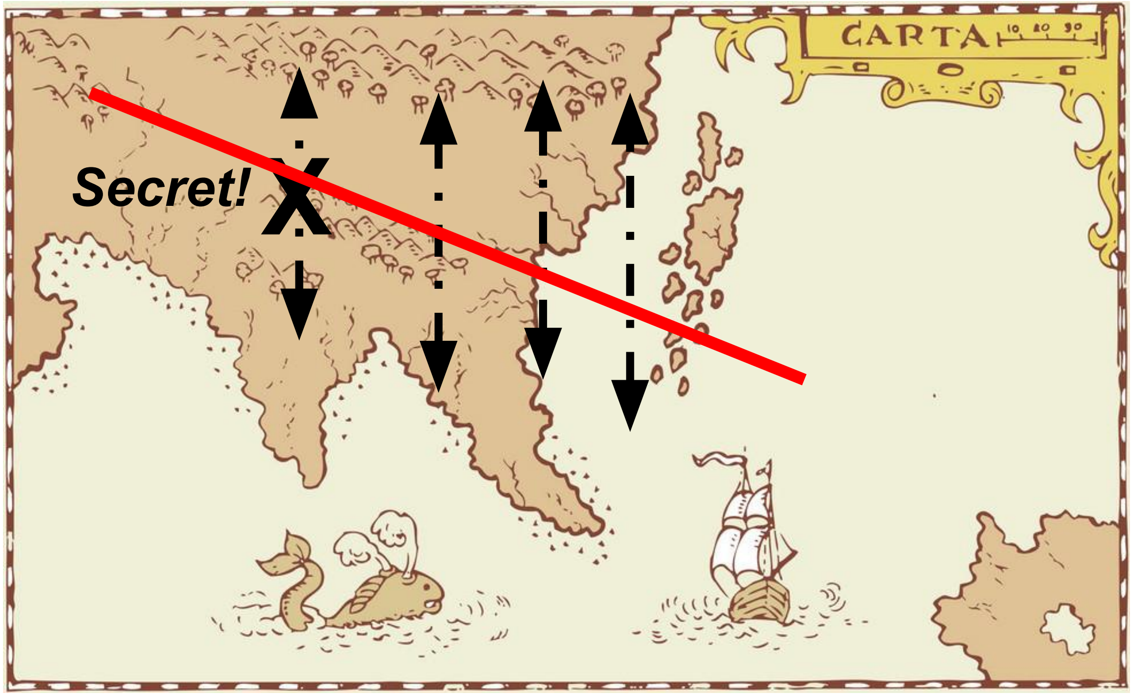
Secret Sharing: Illustration

Then draw three parallel reference lines…



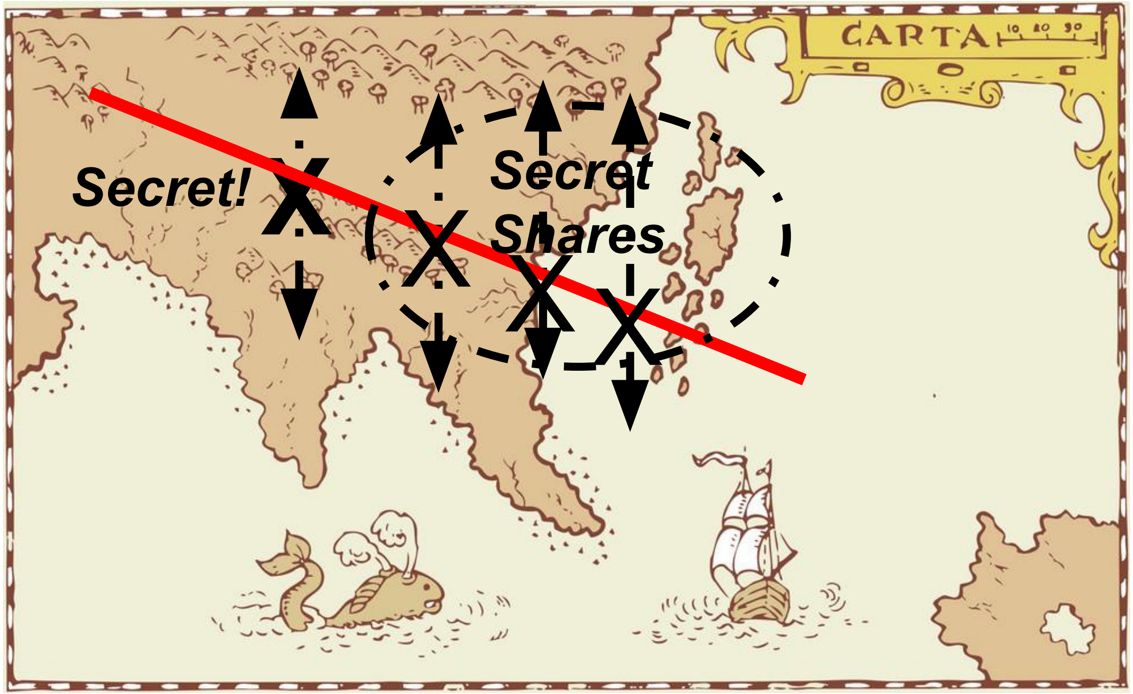
Secret Sharing: Illustration

…and another line intersecting all four…



Secret Sharing: Illustration

The intersection points are the *secret shares*...



Secret Sharing: Illustration

You give *one* of these shares to *each* henchman

Threshold Secret Sharing

Now suppose your henchmen come back later to recover the treasure…

- Any **one** henchman won't know how to find it
- Any **two** henchmen will be able to!

You get both **threshold privacy** of the secret…

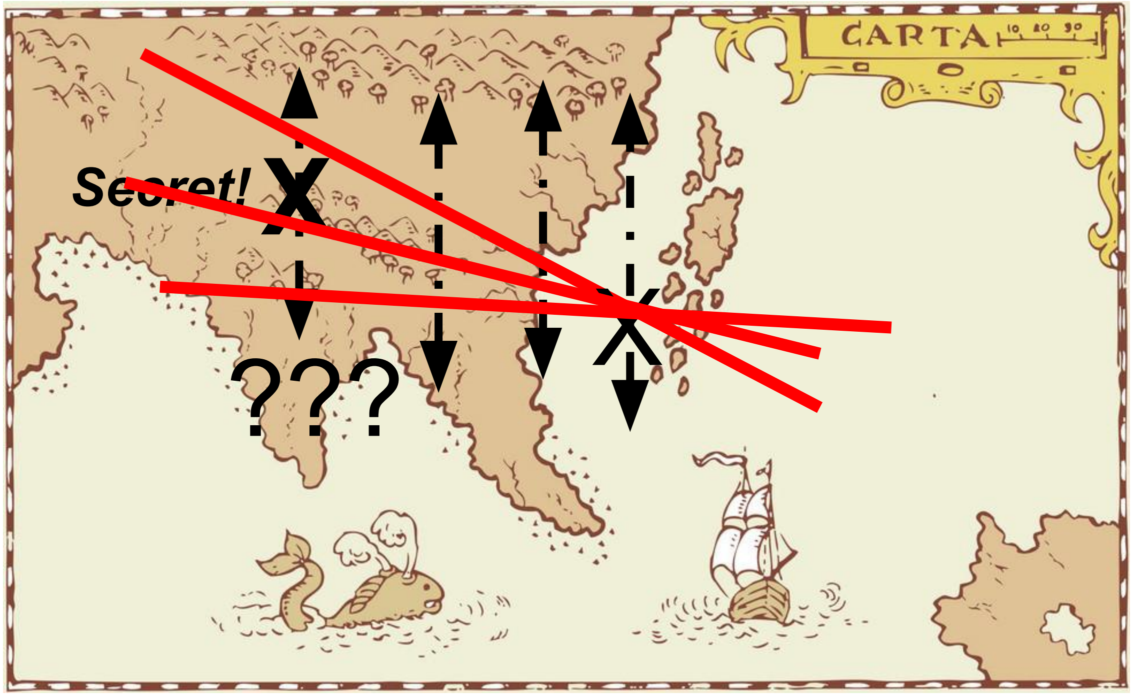
- No single compromised party can recover it

You also get **threshold availability** of the secret

- Can still recover if one henchman goes missing

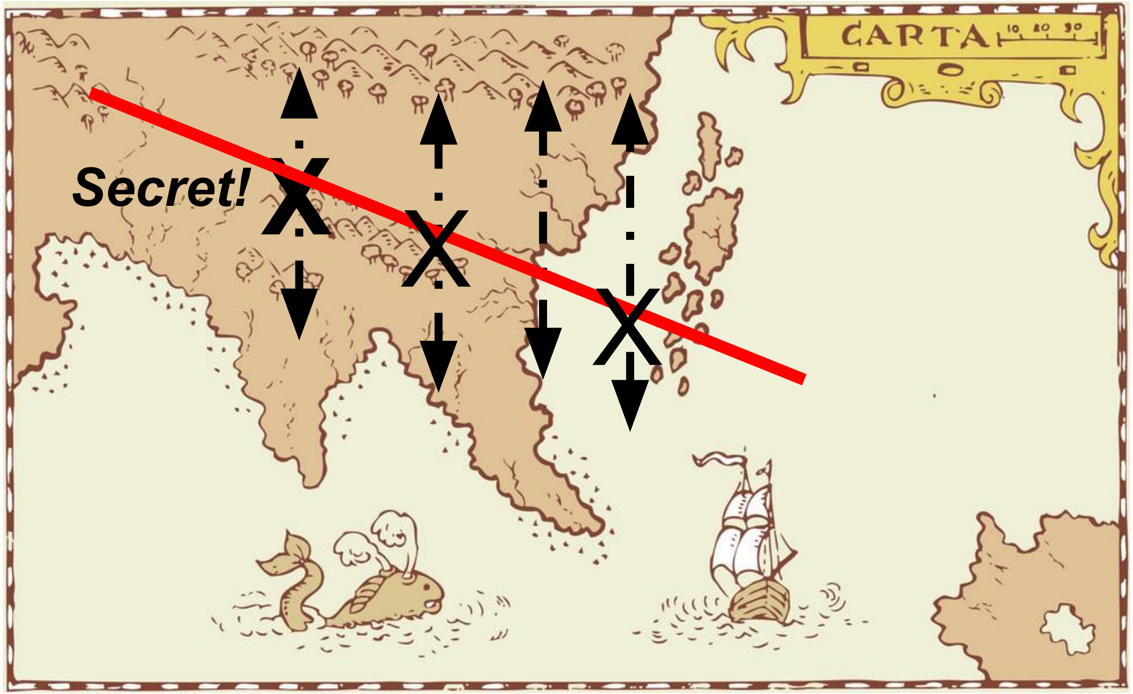
Secret Sharing: Illustration

One henchman alone can't recover secret



Secret Sharing: Illustration

…but *any two* working together can!



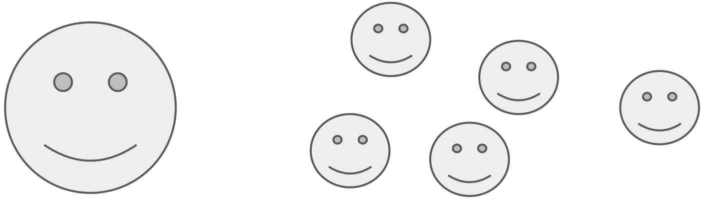
Threshold Cryptosystem

- **Shamir's Secret Sharing (SSS)**
- **Verifiable Secret Sharing (VSS)**
- **Publicly Verifiable Secret Sharing (PVSS)**
- **Distributed Key Generation (DKG)**

Alice sends a message to Bob...

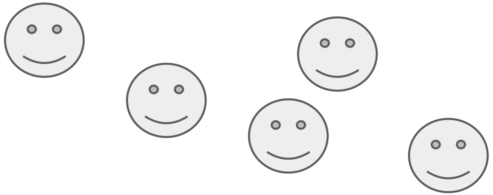
Secrecy? Integrity? Authenticity?

Alice wants to prove that an item belongs to a set, without revealing the set



Accumulators

Share a secret with multiple parties but trust no individual (Byzantine attacker model)



Threshold secret sharing

Interesting Facts

- The DH key exchange protocol was published in 1976
- The RSA scheme was published in 1977

Rewind...

- 1970. James Ellis at UK GCHQ realises that the idea of public key crypto is possible, in a secret report “The possibility of Non-secret Encryption”
- 1973. Clifford Cocks at UK GCHQ discovers a workable mathematical formula for non-secret encryption in a secret report “A note on Non-Secret Encryption”. His formula was a special case of the RSA algorithm
- 1974. Malcolm Williamson at UK GCHQ describes a key exchange method in a secret report “Non-Secret Encryption Using a Finite Field”. His method was similar to the one discovered by Diffie and Hellman

End-to-end encryption

- Only the communicating users can read the messages
- In principle, prevents potential eavesdroppers – including telecom and Internet providers – from being able to decrypt the conversation
- Examples: TLS, Signal Protocol, ...

Man-in-the-Middle

- Open source: mitmproxy.org
- Commercial: Blue Coat SSL Inspector (formerly Netronome)
- Deployment models:
  - "Intended" commercial deployment: at a company's firewall: use custom root CA, but tweak internal web browser to trust
  - Iran's approach in 2011: just use untrusted root cert, assume many/most users will click OK to browser warning
  - More slick (NSA?) approach: subpoena/steal/acquire any root or subordinate CA's private key
- CAs are a major concern and a potential point of failure to MiTM attacks:
  - Google Certificate Transparency project aims at unveiling problems promptly

accumulators

Merkle tree

can classify it as a "static" accumulator

You can add many elements, but only once. Can prove inclusion, but given the root, you can't add more elements

Slide credits: Tadge Dryja

accumulators

accumulator terms

"Dynamic": There's a Remove() function in addition to Add(), which does what you'd think

"Universal": There's a Prove() and Verify() for elements not in the set.

Slide credits: Tadge Dryja

RSA accumulator

RSA-based accumulators...

Wait, RSA? Tough to cover in a few minutes, but a quick refresher!

The original digital signature algorithm. Also does encryption. Powerful, but a bit of a minefield.

Implement with caution!

Slide credits: Tadge Dryja

RSA

make 2 prime numbers, p, q. n = pq.

phi = (p-1)(q-1)

e rnd between 1 and phi s.t.

gcd(e,phi) = 1

Compute d s.t. d * e = 1 mod phi

pubkey: (n,e)

private key: (n,d)

Slide adapted from Tadge Dryja

RSA

encrypt: $c \equiv m^e \mod n$

decrypt: $m \equiv c^d \mod n$

sign: $s \equiv m^d \mod n$

verify: $m \equiv s^e \mod n$

cool!

Slide credits: Tadge Dryja

RSA accumulating

for the accumulator n = pq, but there is no d and no e.

Start with v = 3 or some other starter prime.

Every element x in the set must be prime, so need to hash onto primes

Slide credits: Tadge Dryja

RSA accumulating

Add(x, v): $v' \equiv v^x \mod n$

keep doing that for x_1, x_2, x_3 ...

Prove(x, v): an inclusion proof p is the accumulator v with every element *except* x added

Verify(x, p, v): $p^x \overset{?}{\equiv} v \mod n$

Slide credits: Tadge Dryja

RSA accumulator properties

constant size: v, p, x -- everything's the same length as n, regardless of number of elements

Can prove many inclusions at once, again same size

Slide credits: Tadge Dryja

RSA accumulator issues

p, q are trusted setup. Anyone who knows p, q can create false proofs

while proofs are aggregatable, proof updates are not

Slide credits: Tadge Dryja

RSA proof updates

many proofs $p_1=v^{X-x1}$, $p_2=v^{X-x2}$, $p_3=v^{X-x3}$...

add single element x8

Must compute p_1^{x8}, p_2^{x8}, p_3^{x8}

adding multiple elements x8, x9

must compute $(p_1^{x8})^{x9}$, $(p_2^{x8})^{x9}$, $(p_3^{x8})^{x9}$

Slide credits: Tadge Dryja

What do we want to accumulate?

How about accumulating some bitcoins?

If proof updates are few / infrequent, then we're OK.
But if we're looking at the UTXO set, proof updates happen every 10 minutes.

Slide credits: Tadge Dryja

What do we want to accumulate?

If we wanted to prove every bitcoin:

60M utxos
~6K updates every 600 sec (10/sec)

For individual proofs, 60M * 10 =
600M exponentiations / sec

@1ms per op, need 600K cpu cores!

Slide credits: Tadge Dryja

scalability

Do we need to keep proofs for every possible transaction?

Maybe not; if wallets keep track of their own UTXOs and proofs, it's much more reasonable

Slide credits: Tadge Dryja

scalability

Lightly used wallet: 10 utxos

6K updates per block * 10 txos =
60K exponentiations per block

@1ms each, that's 1 minute of CPU
time per block
Doable, but still lots of work

Slide credits: Tadge Dryja