

Introduction to Cryptography

Alessandro Barenghi

Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
Politecnico di Milano

alessandro -dot- barenghi - at - polimi -dot- it

What is cryptography (alt. cryptology)?

Definition

- The study of techniques to allow secure communication and data storage in presence of attackers

Features provided

- **Confidentiality:** data can be accessed only by chosen entities
- **Integrity/freshness:** detect/prevent tampering or replays
- **Authenticity:** data and their origin are certified
- **Non-repudiation:** data creator cannot repudiate created data
- **Advanced features:** proofs of knowledge/computation

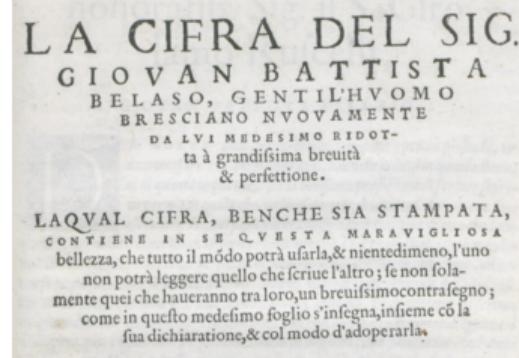
As old as written communication

- Born for commercial (recipe for lacquer on clay tablets) or military (Spartans) uses
- Designed by humans, for human computers
- Algorithms computed by hand, with pen and paper



Original approach

- A battle of wits between
 - cryptographers: ideate a secret method to obfuscate a text
 - cryptanalysts: figure out the method, break the “cipher”
- Bellaso (1553) [1] separates the encryption method from the key



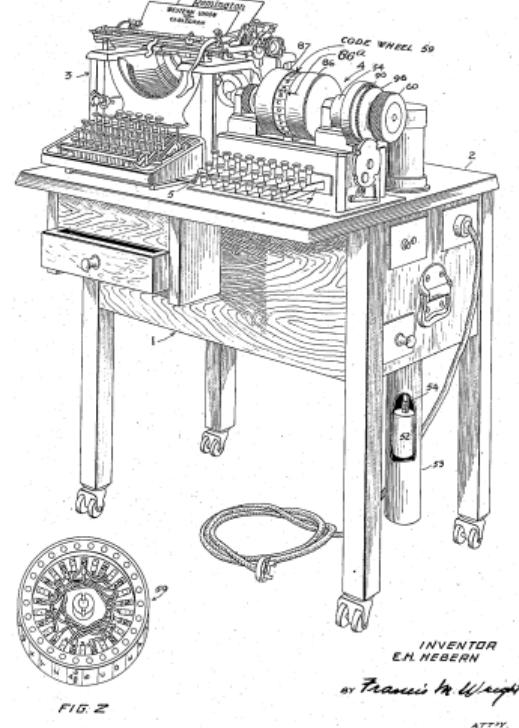
1883 - Kerchoff's six principles for a good cipher (apparatus)

- ① It must be practically, if not mathematically, unbreakable
- ② It should be possible to make it public, even to the enemy
- ③ The key must be communicable without written notes and changeable whenever the correspondants want
- ④ It must be applicable to telegraphic communication
- ⑤ It must be portable, and should be operable by a single person
- ⑥ Finally, given the operating environment, it should be easy to use, it shouldn't impose excessive mental load, nor require a large set of rules to be known

Cryptographic modern history

The advent of the machines

- Mechanical computation changes cryptography
 - First rotor machine in 1917 by Ed Hebern
 - Design “popularized” in WWII by German Enigma
- Cryptanalyst at Bletchley park (Turing among them) credited for a decisive effort in winning the war by Eisenhower



An end to the battle of wits

- Shannon (1949) [4] - Proves that a mathematically unbreakable cipher exists
- Nash (1955) [2] - Argues that computationally secure ciphers are ok
 - Considers a cipher with a finite, λ bit long, key
 - Conjecture: if "*parts of the key interact complexly [...] in the determination of their effects on the ciphertext*", the attacker effort to break the cipher would be $\mathcal{O}(2^\lambda)$
 - The owner of the key takes $\mathcal{O}(\lambda^2)$ to compute the cipher
 - The computational gap is unsurmountable for large λ

Outline of the topics

In this course

- Definitions of ciphers as components with functionalities
- How to obtain confidentiality, integrity, data/origin authentication
- An overview of protocols (combinations of ciphers)
- Goal: be able to **use** cryptographic components properly

In Cryptography and Architectures for Computer Security

- Design and cryptanalysis techniques for ciphers
- Cryptographic protocols and their inner workings
- Efficient, side channel attack resistant implementations
- Goal: be able to **engineer** cryptographic components

Before we start...

```
int getRandomNumber()
{
    return 4; // chosen by fair dice roll.
               // guaranteed to be random.
}
```

<https://xkcd.com/221/>

A word on randomness

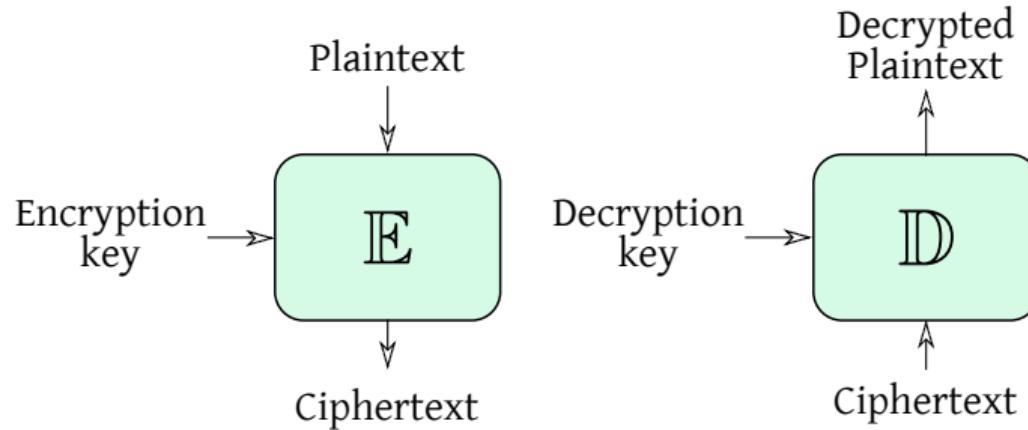
- Randomness (in this course) characterizes a generative process
- Stating: “00101 is a random string” actually makes little sense

Data

- Plaintext space **P**: set of possible messages $\text{ptx} \in \mathbf{P}$
 - Old times: words in some human-readable alphabet, modern times $\{0, 1\}^l$
- Ciphertext space **C**: set of possible ciphertext $\text{ctx} \in \mathbf{C}$
 - Usually $\{0, 1\}^{l'}$, not necessarily $l = l'$ (ciphertexts may be larger)
- Key space **K**: set of possible keys
 - $\{0, 1\}^\lambda$, keys with special formats are derived from bitstrings

Functions

- Encryption function $\mathbb{E} : \mathbf{P} \times \mathbf{K} \rightarrow \mathbf{C}$
- Decryption function $\mathbb{D} : \mathbf{C} \times \mathbf{K} \rightarrow \mathbf{P}$
 - Correctness: for all $\text{ptx} \in \mathbf{P}$, we need $k, k' \in \mathbf{K}$ s.t. $\mathbb{D}(\mathbb{E}(\text{ptx}, k), k') = \text{ptx}$



Goal

- Prevent anyone not authorized from being able to understand data

Possible attacker models

- The attacker simply eavesdrops (ciphertext only attack)
- The attacker knows a set of possible plaintexts
 - Limit case: the attacker chooses the set of plaintexts
- The attacker may tamper with the data and observe the reactions of a decryption-capable entity
 - Limit case: the attacker sees the actual decrypted value

Perfectly secure cipher

Definition

- In a *perfect cipher*, for all $\text{ptx} \in \mathbf{P}$ and $\text{ctx} \in \mathbf{C}$,
 $\Pr(\text{ptx sent} = \text{ptx}) = \Pr(\text{ptx sent} = \text{ptx} \mid \text{ctx sent} = \text{ctx})$
- In other words: seeing a ciphertext $c \in \mathbf{C}$ gives us *no information* on what the plaintext corresponding to c could be

Question

- The definition is not constructive! Does a perfect cipher exist?
 - If yes, what does it look like?

Perfectly secure cipher

Theorem (Shannon 1949)

Any symmetric cipher $\langle \mathbf{P}, \mathbf{K}, \mathbf{C}, \mathbb{E}, \mathbb{D} \rangle$ with $|\mathbf{P}| = |\mathbf{K}| = |\mathbf{C}|$ is perfectly secure if and only if

- every key is used with probability $\frac{1}{|\mathbf{K}|}$
- a unique key maps a given plaintext into a given ciphertext:
 $\forall (\text{ptx}, \text{ctx}) \in \mathbf{P} \times \mathbf{C}, \exists! k \in \mathbf{K} \text{ s.t. } \mathbb{E}(\text{ptx}, k) = \text{ctx}$

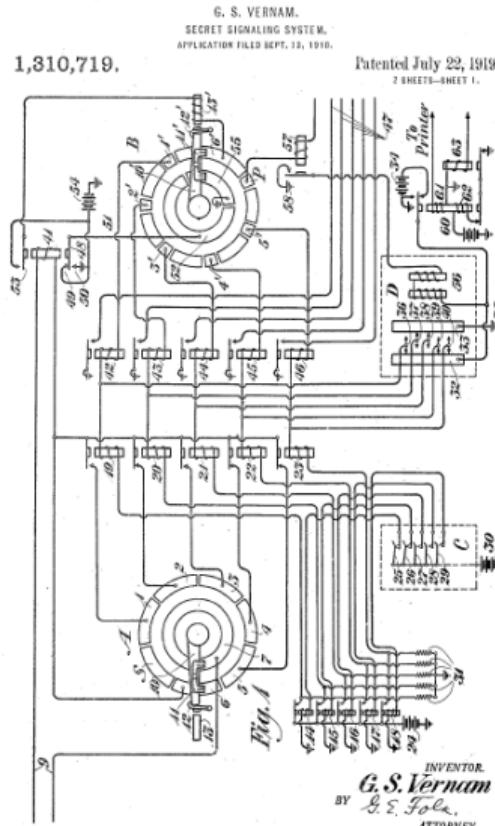
A simple working example

- Assume $\mathbf{P}, \mathbf{K}, \mathbf{C}$ to be set of binary strings. The encryption function draws a *uniformly random, fresh* key k out of \mathbf{K} each time it is called and computes $\text{ctx} = \text{ptx} \oplus k$

Anticausal implementations

A concrete apparatus

- Gilbert Vernam actually patented a telegraphic machine implementing $ptx \oplus k$ on Baudot code in 1919
- Joseph Mauborgne suggested the use of a random tape containing k
- Using Vernam's encrypting machine with Mauborgne's suggestion implements a perfect cipher



Key storage/management

- storing key material and changing keys is a nightmare
- perfect cipher broken in practice due to key theft/reuse
- generating random keys was also an issue (and caused breaks)



Photo courtesy of [Cryptomuseum.com](http://www.cryptomuseum.com)

Computationally secure cryptography

A more practical assumption

- Build a cipher so that a successful attack is also able to solve a hard computational problem efficiently
 - Solve a generic nonlinear Boolean simultaneous equation set
 - Factor large integers / find discrete logarithms
 - Decode a random code/find shortest lattice vector

Can we avoid assumptions?

- Open question: prove an exponential lower bound for the time taken to solve a hard problem, which has efficiently verifiable solution
 - it would shift from a computational security **assumption** to a **theorem**
 - ... and prove $P \neq NP$ as a **corollary**

Outline of the method

- ① Define the ideal attacker behaviour
- ② Assume a given computational problem is hard
- ③ Prove that any non ideal attacker solves the hard problem

How to represent attacker and properties?

- Attacker represented as a program able to call given libraries
- Libraries implement the cipher at hand
- Define the security property as answering to a given question
- The attacker wins the game if it breaks the security property more often than what is possible through a random guess

Motivation and assumption

- We want to use a finite-length key and a Vernam cipher
 - We somehow need to “expand” the key
- We assume that the attacker can only perform $\text{poly}(\lambda)$ computations

Definition

A CSPRNG is a deterministic function $\text{PRNG}: \{0, 1\}^\lambda \rightarrow \{0, 1\}^{\lambda+l}$ whose output cannot be distinguished from an uniform random sampling of $\{0, 1\}^{\lambda+l}$ in $\mathcal{O}(\text{poly}(\lambda))$. l is the CSPRNG **stretch**.

Existence

- In practice, we have only candidate CSPRNGs
 - We have no **proof** that a function PRNG exists
 - Proving that a CSPRNG exists implies directly $P \neq NP$

Practical constructions

- Building a CSPRNG “from scratch” is possible, but it is not the way they are commonly built (not efficient)
- Practically built with another building block: PseudoRandom Permutations (PRPs)
 - defined starting from PseudoRandom Functions (PRFs)

Randomly drawing a function

- Consider the set $\mathbf{F} = \{f : \{0, 1\}^{in} \rightarrow \{0, 1\}^{out}; in, out \in \mathbb{N}\}$
- A uniformly randomly sampled $f \xleftarrow{\$} \mathbf{F}$ can be encoded by a 2^{in} entries table, each entry out bit wide. $|\mathbf{F}| = (2^{out})^{2^{in}}$

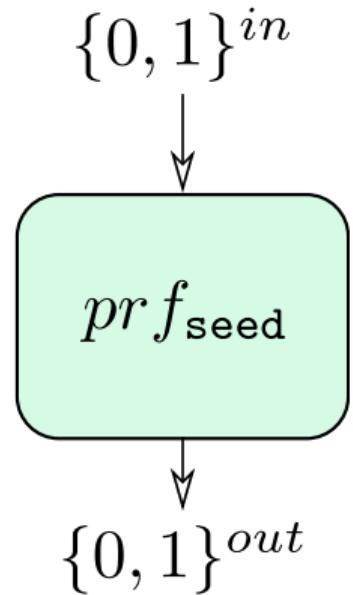
Toy example $in = 2, out = 1$

- $\mathbf{F} = \{f : \{0, 1\}^2 \rightarrow \{0, 1\}^1\}$ is the set of the 16 Boolean functions w/ two inputs
- Each one is represented by a 4-entry truth table
- Intuitively, the functions are 16 as there are $2^4 = 16 = (2^1)^{2^2}$ tables

Pseudorandom Functions (PRFs)

Definition

- A function $\text{prf}_{\text{seed}} : \{0, 1\}^{\text{in}} \rightarrow \{0, 1\}^{\text{out}}$ taking an input and a λ bits seed.
- The entire prf_{seed} is described by the value of the seed
- It cannot be told apart from a random $f \in \{f : \{0, 1\}^{\text{in}} \rightarrow \{0, 1\}^{\text{out}}\}$ in $\text{poly}(\lambda)$
- That is, if they give you $a \in \{f : \{0, 1\}^{\text{in}} \rightarrow \{0, 1\}^{\text{out}}\}$, you can't tell which one of the following is true
 - $a = \text{prf}_{\text{seed}}(\cdot)$ with seed $\xleftarrow{\$} \{0, 1\}^\lambda$
 - $b \xleftarrow{\$} \mathbf{F}$, where $\mathbf{F} = \{f : \{0, 1\}^{\text{in}} \rightarrow \{0, 1\}^{\text{out}}\}$



Pseudorandom Permutations (PRPs)

Pseudorandom Permutation definition

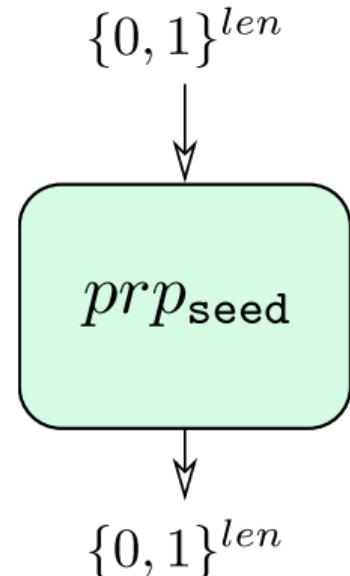
- A **bijective** PRF: $prf_{seed} : \{0, 1\}^{len} \rightarrow \{0, 1\}^{len}$

Wrapping your mind around it

- It is uniquely identified by the value of the seed
- It is not possible to tell apart in $\text{poly}(\lambda)$ from a RF
- It's a permutation of all the possible $\{0, 1\}^{len}$ strings

Operatively speaking

- acts on a **block** of bits outputs another one of the same size
- the output “looks unrelated” to the input
- its action is fully identified by the seed
 - Useful to think of the seed as a key



The issue

- No formally proven PRP exists, yet
 - again, its existence would imply $P \neq NP$

Typical construction

- ① Compute a small bijective Boolean function f of input and key
- ② Compute f again between the previous output and the key
- ③ Repeat 2 until you're satisfied

In the real world...

Practical solution: public scrutiny

- Modern PRPs are the outcome of public contests
- Cryptanalytic techniques provide ways ($=\text{poly}(\lambda)$ tests) to detect biases in their outputs: good designs are immune

PRPs a.k.a. Block ciphers

- Concrete PRPs go by the historical name of **block ciphers**
- Considered broken if, with less than 2^λ operations, they can be told apart from a PRP, e.g., via:
 - Deriving the input corresponding to an output without the key
 - Deriving the key identifying the PRP, or reducing the amount of plausible ones
 - Identifying non-uniformities in their outputs
- The key length λ is chosen to be large enough so that computing 2^λ guesses is not **practically feasible**

Quantifying computational unfeasibility

Boolean operations vs. energy to bring from 20 °C → 100 °C

- 2^{65} op.s ≈ an Olympic swimming pool
- 2^{80} op.s ≈ the annual rainfall on the Netherlands
- 2^{114} op.s ≈ all water on Earth

Practically acceptable unfeasibility

- Legacy level security: at least 2^{80} Boolean operations
- 5 to 10 years security: at least 2^{128} Boolean operations
- Long term security: at least 2^{256} Boolean operations

Advanced Encryption Standard (AES)

- 128 bit block, three key lengths: 128, 192 and 256 bits
- Selected after a 3 years public contest in 2000-10-2 by NIST out of 15 candidates, re-standardized by ISO/IEC
- ARMv8 and AMD64 include dedicated instructions accelerating its computation (hitting 3+ GB/s)

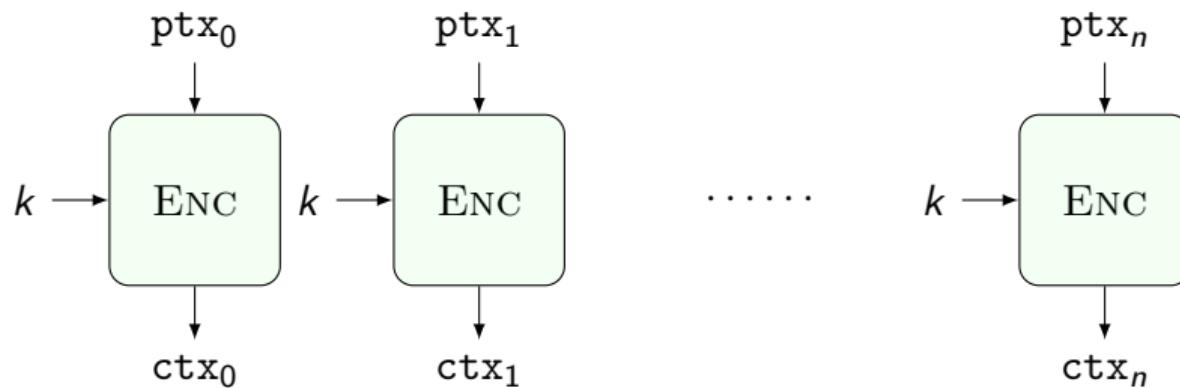
Data Encryption Algorithm (DEA, a.k.a. DES)

- Legacy standard by NIST (1977), the key is too short (56b)
- Patch via triple encryption, $\lambda = 112$ equivalent security
- Still found in some legacy systems, officially deprecated

An Electronic CodeBook (ECB)

A first attempt to encryption with PRPs

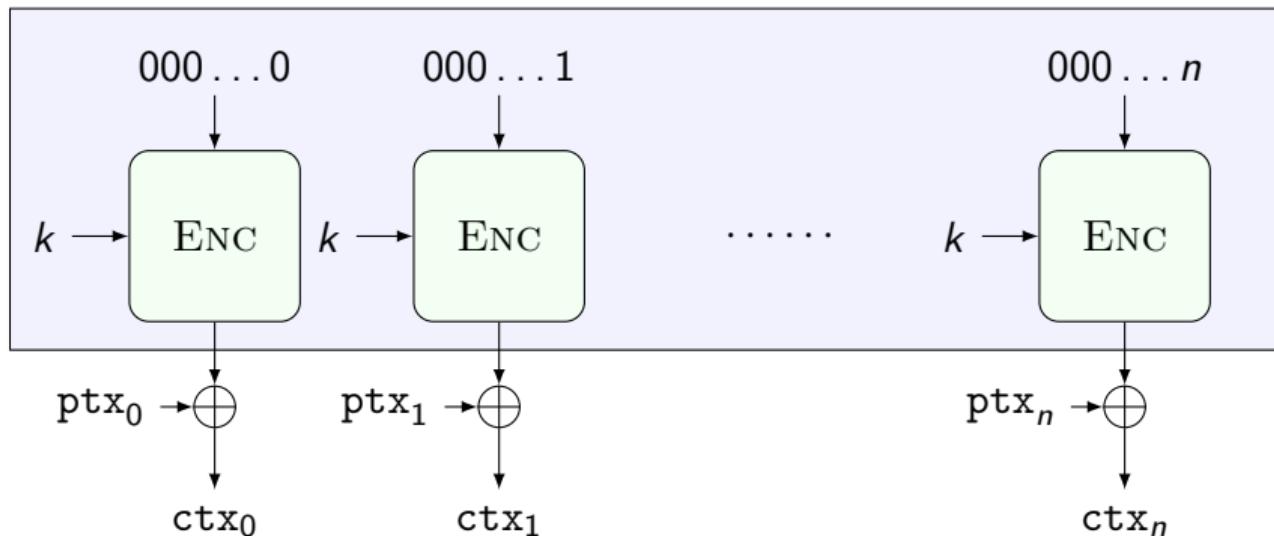
- It's ok to encrypt a plaintext \leq block size with a block cipher
- An extension to multiple blocks could be split-and-encrypt
 - Is it good (equivalent to Vernam fed with a CSPRNG)?



Counter (CTR) mode

Getting it right

- The boxed construction is provably a PRNG if Enc is a PRP
- There is nothing special in the starting point of the counters



Confidentiality achieved ... for CoA

- Up to now, the attacker knew only ciphertext material

Confidentiality against Chosen Plaintext Attacks (CPAs)

- Our attacker knows a set of plaintexts which can be encrypted
- He wants to understand **which** one is being encrypted
- Ideal attacker: cannot tell which plaintext was encrypted out of two *he chose* (having the same length)
- Feels strange, but it happens with:
 - management data packets in network protocols (e.g., ICMP)
 - telling apart encrypted commands to a remote host

No deterministic encryption

- The CTR mode of operation is insecure against CPA
 - The encryption is deterministic: same ptex → same ctx

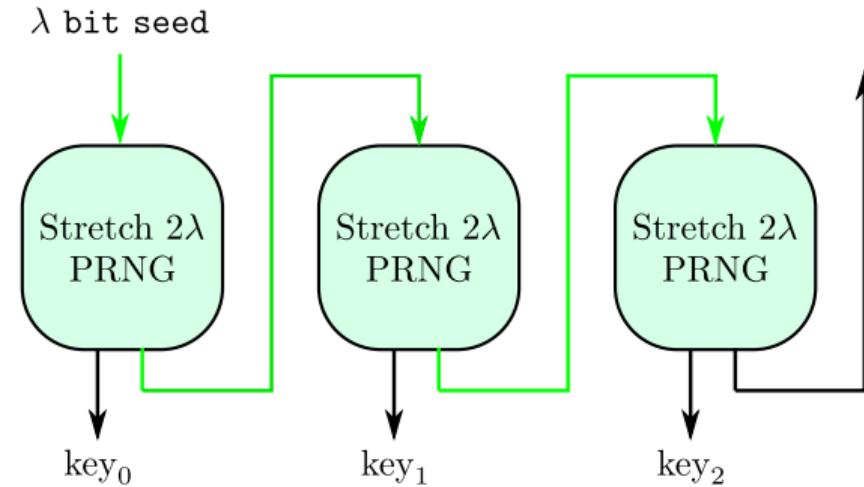
Decryptable nondeterministic encryption

- ① Rekeying: change the key for each block with a ratchet
- ② Randomize the encryption: add (removable) randomness to the encryption (change mode of employing PRP)
- ③ Numbers used ONCE (NONCEs): in the CTR case, pick a NONCE as the counter starting point. NONCE is public

Symmetric ratcheting

Getting it right

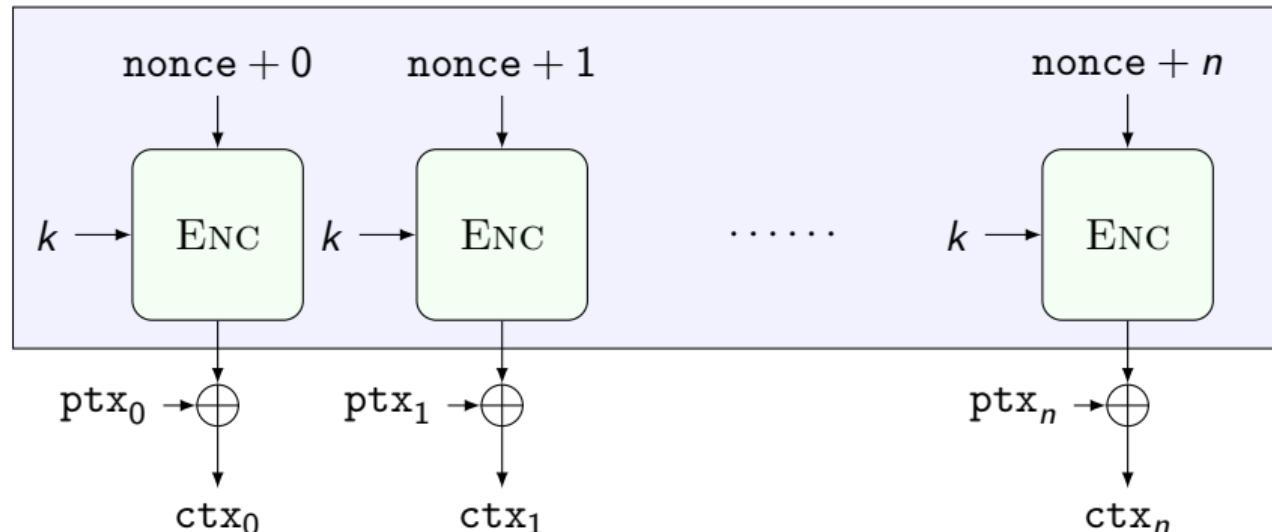
- The construction takes the name from the mechanical component: it is not possible to roll-back the procedure once you delete the value carried by green arrows



CPA-Secure Counter (CTR) mode

Getting it right

- Picking the counter start as a NONCE generates different bitstreams to be xor-ed with the ptx each time
- The same plaintext encrypted twice is turned into two different, random-looking, ciphertexts



Malleability

- Making changes to the ciphertext (not knowing the key) maps to predictable changes in the plaintext
 - Think about AES-CTR and AES-ECB
- Can be creatively abused to build decryption attacks
- Can be turned into a feature (homomorphic encryption)

How to avoid malleability

- Design an intrinsically non malleable scheme (non trivial)
- Add a mechanism ensuring data integrity (against attackers)

Providing data integrity

Confidentiality $\not\Rightarrow$ Integrity

- Up to now our encryption schemes provide confidentiality
- Changes in the ciphertext are undetected (at best)

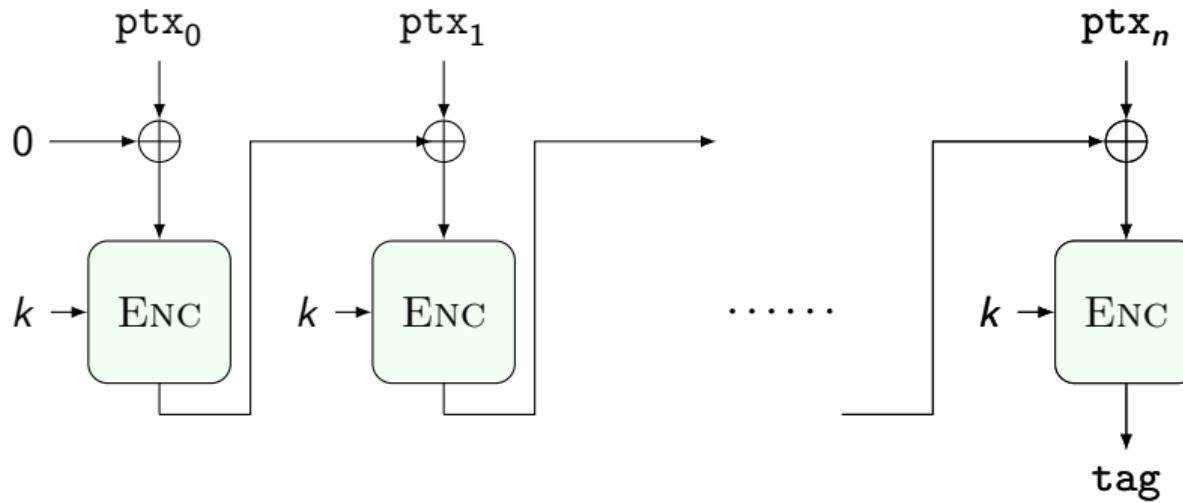
Message Authentication Codes (MAC)

- Add a small piece of information (tag) allowing us to test for the message **integrity** of the encrypted message itself
 - Adding it to the plaintext and then encrypting is not good
- Nomenclature misleads: MACs do not provide data authentication

Definition

- A MAC is constituted by a pair of functions:
 - COMPUTE_TAG(string,key): returns the tag for the input string
 - VERIFY_TAG(string,tag,key): returns true or false
- Ideal attacker model:
 - knows as many message-tag pairs as he wants
 - cannot forge a valid tag for a message for which he does not know it already
 - forgery also includes tag splicing from valid messages
- N.B. the tag creating entity and the verifying entity must both know the same secret key
 - The tag verifier is able to create a valid tag too
 - ... and there goes the non-repudiation property

How to build a MAC? the CBC-MAC



Building a MAC with a PRP (block cipher)

- The CBC-MAC is secure for prefix free messages (why?)
- Encrypting the tag once more fixes (provably) the issue

Browser cookies

- HTTP cookies are a “note to self” for the HTTP server^a
- The note should not be tampered between server reads
- Solution: server runs $\text{COMPUTE_TAG(cookie, } k)$ and stores both the (cookie,tag)

^aYou can find a two slides cookies refresher at slides 40-41 of
https://polimi365-my.sharepoint.com/:b/r/personal/10032133_polimi_it/Documents/FCI/2-Livello_Applicativo_v2020.pdf

Later in the course

- Mitigating SYN-based denial of service attacks (SYN Cookies)
- Time-based two-factor authentication mechanisms (TOTP/HOTP)

Testing integrity

- Testing the integrity of a file requires us to compare it bit by bit with an intact copy or read it entirely to compute a MAC
- It would be **fantastic** to test only short, fixed length strings independently from the file size, representing the file itself
 - Major roadblock: there is a lower bound to the number of bits to encode a given content without information loss
- Can we build something close to the ideal scenario?

A pseudo-unique labeling function

- A cryptographic hash is a function $H : \{0, 1\}^* \rightarrow \{0, 1\}^l$ for which the following problems are computationally hard
 - ① given $d = H(s)$ find s (1st preimage)
 - ② given $s, d = H(s)$ find $r \neq s$ with $H(r) = d$ (2nd preimage)
 - ③ find $r, s; r \neq s$, with $H(s) = H(r)$ (collision)
- Ideal behaviour of a concrete cryptographic hash:
 - ① finding 1st preimage takes $\mathcal{O}(2^d)$ hash computations guessing s
 - ② finding 2nd preimage takes $\mathcal{O}(2^d)$ hash comp.s guessing r
 - ③ finding a collision takes $\approx \mathcal{O}(2^{\frac{d}{2}})$ hash computations
- The output bitstring of a hash is known as a **digest**

Concrete hash functions

What to use

- SHA-2 was privately designed (NSA), $d \in \{256, 384, 512\}$
- SHA-3 followed a public design contest (similar to AES), selected among ≈ 60 candidates, $d \in \{256, 384, 512\}$
- Both currently unbroken and widely standardized (NIST, ISO)

What not to use

- SHA-1: $d = 160$, collision-broken [6] (obtainable in $\approx 2^{61}$ op.s)
- MD-5: horribly broken [7]. Collisions in 2^{11} , public tools online [5]
 - In particular, collisions with arbitrary input prefixes in $\approx 2^{40}$

Uses for hash functions

Pseudonymized match

- Store/compare hashes instead of values (e.g., Signal contact discovery)

MACs

- Building MACs: generate tag hashing together the message and a secret string, verify tag recomputing the same hash
- A field-proven way of combining message and secret is HMAC
 - Standardized (RFC 2104, NIST FIPS 198)
 - Uses a generic hash function as a plug-in, combination denoted as HMAC-hash_name
 - HMAC-SHA1 (!), HMAC-SHA2 and HMAC-SHA3 are ok

Forensic use

- Write down only the hash of the disk image you obtained in official documents

Game changing ideas

Features we would like to have

- Agreeing on a short secret over a public channel
- Confidentially sending a message over a public authenticated channel without sharing a secret with the recipient
- Actual data authentication

Solution: asymmetric cryptosystems

- Before 1976: rely on human carriers / physical signatures
- DH key agreement (1976) / Public key encryption (1977)
- Digital signatures (1977)

Computational hardness

- Up to now, enumeration of the secret parameter was the best possible attack
- This is ok for modern block ciphers → best attack: $\mathcal{O}(2^\lambda)$
- Asymmetric cryptosystems rely on hard problems for which bruteforcing the secret parameter is **not** the best attack
 - Factoring a λ bit number takes $\mathcal{O}\left(e^{k(\lambda)^{\frac{1}{3}}(\log(\lambda))^{\frac{2}{3}}}\right)$
- Comparing bit-sizes of the security parameters instead of actual complexities is **really** wrong
- Concrete bit-sizes for λ depending on the cipher: www.keylength.org

The Diffie-Hellman key agreement

Goal

- Make two parties share secret value w/ only public messages

Attacker model

- Can eavesdrop anything, but not tamper
- The Computational Diffie-Hellman assumption should hold

CDH Assumption

- Let $(\mathbf{G}, \cdot) \equiv \langle g \rangle$ be a finite cyclic group, and two numbers a, b sampled unif. from $\{0, \dots, |\mathbf{G}| - 1\}$ ($\lambda = \text{len}(a) \approx \log_2 |\mathbf{G}|$)
- given g^a, g^b finding g^{ab} costs more than $\text{poly}(\log |\mathbf{G}|)$
- Best current attack approach: find either b or a (discrete log problem)

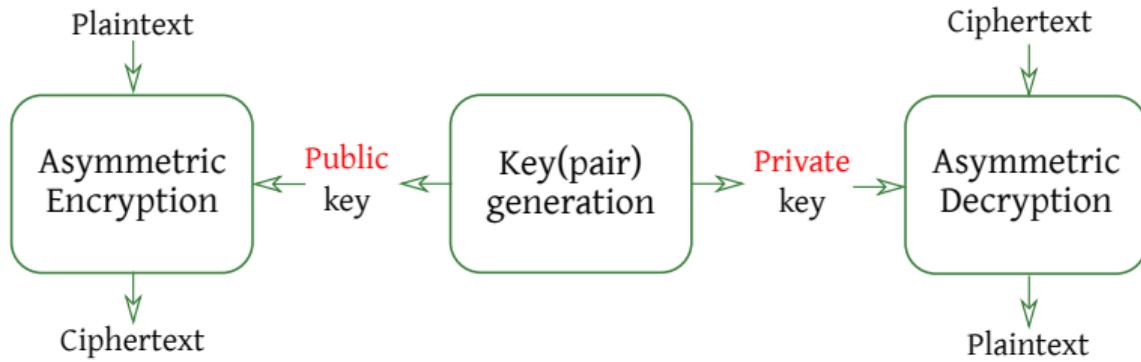
Key agreement between Alice and Bob

- Alice: picks $a \xleftarrow{\$} \{0, \dots, |\mathbf{G}| - 1\}$, sends g^a to Bob
- Bob: picks $b \xleftarrow{\$} \{0, \dots, |\mathbf{G}| - 1\}$, sends g^b to Alice
- Alice: gets g^b from Bob and computes $(g^b)^a$
- Bob: gets g^a from Alice and computes $(g^a)^b$
- (\mathbf{G}, \cdot) is commutative $\rightarrow (g^b)^a = (g^a)^b$, we're done!

Groups used in practice

- A subgroup (\mathbf{G}, \cdot) of (\mathbb{Z}_n^*, \cdot) (integers mod n), breaking CDH takes
$$\min \left(\mathcal{O} \left(e^{k(\log(n))^{\frac{1}{3}} (\log(\log(n)))^{\frac{2}{3}}} \right), \mathcal{O}(2^{\frac{\lambda}{2}}) \right)$$
- EC points w/ dedicated addition, breaking CDH takes $\mathcal{O}(2^{\frac{\lambda}{2}})$

Public Key Encryption



Components

- *Different keys* are employed in encryption/decryption
- It is computationally hard to:
 - Decrypt a ciphertext without the *private key*
 - Compute the *private key* given only the *public key*

Widespread Asymmetric encryption ciphers

Rivest, Shamir, Adleman (RSA), 1977

- 2048 to 4096 bit message-and key-sizes
- Patented after the invention, patent now expired
- No ciphertext expansion
- Incidentally, the encryption with a fixed key is a PRP

Elgamal encryption scheme, 1985

- Either kbit range keys, or 100's of bits keys, depending on the variant
- Not encumbered by patents
- The ciphertext is **twice** the size of the plaintext
- Widely used as an RSA alternative where patents were a concern

Assumption

- A public channel between Alice and Bob is available
- For the moment, the attacker model is “eavesdrop only”

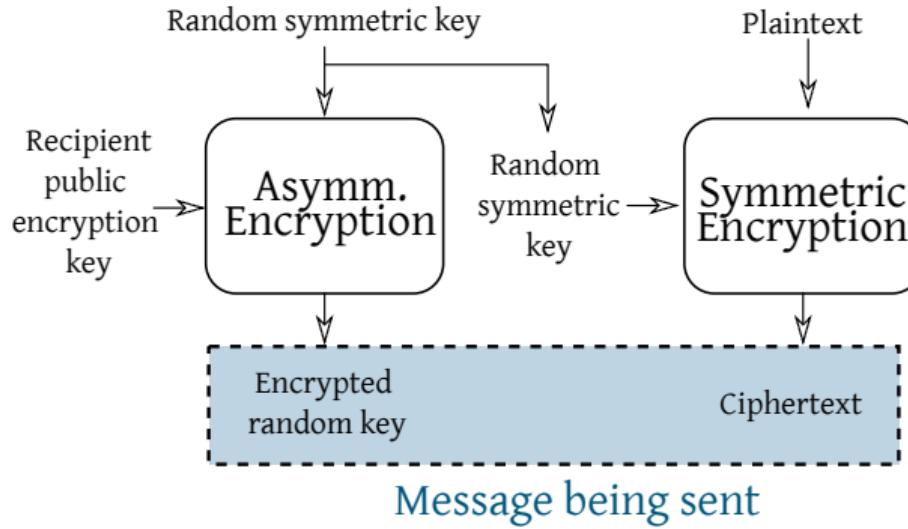
Sharing a secret without agreement

- Alice: generates a keypair (k_{pri}, k_{pub}) , sends to Bob
- Bob: gets $s \xleftarrow{\$} \{0, 1\}^\lambda$, encrypts it with k_{pub} , sends ctx to Alice
- Alice: decrypts ctx with k_{pri} , recovers s
- Note: Bob alone decides the value of the shared secret s
 - Repeat the procedure with swapped roles and combine the two secrets to achieve **similar** guarantees to a key agreement

Can't I skip a step?

- Employing an asymmetric cryptosystem Bob encrypts a text for Alice without the need of sharing a secret beforehand
- In principle, Bob and Alice could employ *only* an asymmetric cryptosystem to communicate
- In practice this approach would be extremely inefficient
 - Asymmetric cryptosystems are from $10\times$ to $1000\times$ slower than their symmetric counterparts

Best of both worlds



Hybrid encryption schemes

- Asymmetric schemes to provide key transport/agreement
- Symmetric schemes to encrypt the bulk of the data
- All modern secure transport protocols built around this idea

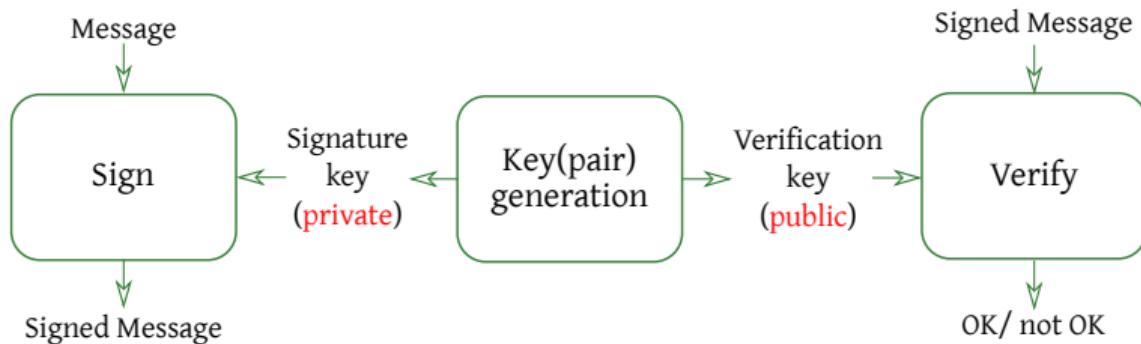
Motivations

- To build a secure hybrid encryption scheme we need to be sure that the public key the sender uses is the one of the recipient
- We'd like to be able to verify the authenticity of a piece of data without a pre-shared secret

Digital signatures

- Provide strong evidence that data is bound to a specific user
- No shared secret is needed to check (validate) the signature
- Proper signatures cannot be repudiated by the user
- They are **asymmetric** cryptographic algorithms
 - formally proven that you cannot get non repudiation otherwise

Digital signatures



Computationally hard problems

- Sign a message without the **signature** key
 - this includes splicing signatures from other messages
- Compute the **signature** key given only the **verification** key
- Derive the **signature** key from signed messages

Widespread Signature schemes

Rivest, Shamir, Adleman (RSA), 1977

- Unique case: the same hard-to-invert function to build an asymmetric encryption scheme and a signature (different message processing!)
- Signing definitely slower than verification ($\approx 300\times$)
- Standardized in NIST DSS (FIPS-184-4)

Digital Signature Standard (DSA)

- Derived from tweaking signature schemes by Schnorr and Elgamal
- Also standardized in NIST DSS (FIPS-184-4)
- Signature and verification take roughly the same time

Digital signature uses

Authenticating digital documents

- For performance reasons, sign the hash of the document instead of the document
 - Signature properties now guaranteed only if **both** signature and hash algorithms are not broken

Authenticating users

- Alternative to password-based login to a system
 - The server has the user's public verification key (e.g. deposited at account creation)
 - The server asks the client to sign a long randomly generated bitstring (challenge)
 - If the client returns a correctly signed challenge, it has proven its identity to the server

The importance of being static

- 2002-09-09: several pieces of software performing digital signatures with legal value in Italy allowed to sign MS Word documents *containing macros*
 - Macros allow to dynamically change the displayed text in a document according to, e.g., the current date
- Striking mismatch between what was thought to be signed (the visualized document) and the actual signed object (a program-document blend)
- Current standard for digital signatures on human-intended documents (PAdES,CAdES) target PDF and XML formats
 - n.b.: PDFs may embed Javascript, PDF/A do not

Cautionary note

- Both in asymmetric encryption and digital signatures, the public key must be bound to the correct user identity
- If public keys are not authentic:
 - A MITM attack is possible on asymmetric encryption
 - Anyone can produce a signature on behalf of anyone else
- The public key authenticity is guaranteed with... another signature
 - We need someone to sign the public-key/identity pair
 - We need a format to distribute signed pairs

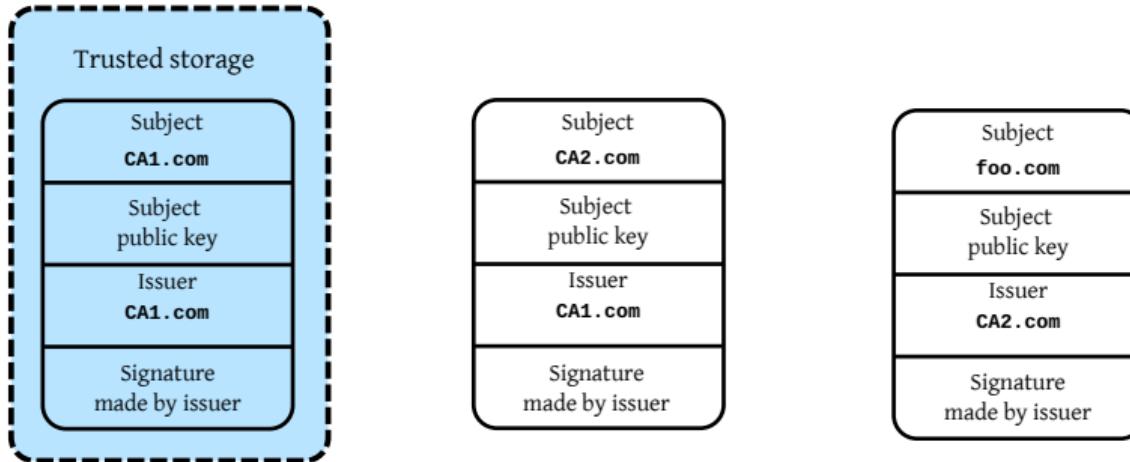
Digital certificates

- They bind a public key to a given identity, which is:
 - for humans: an ASCII string
 - for machines: either the CNAME or IP address
- They specify the intended use for the public key contained
 - Avoids ambiguities when a key format is ok for both an encryption and a signature algorithm
- They contain a time interval in which they are valid
- Most widely deployed format is described in ITU X.509

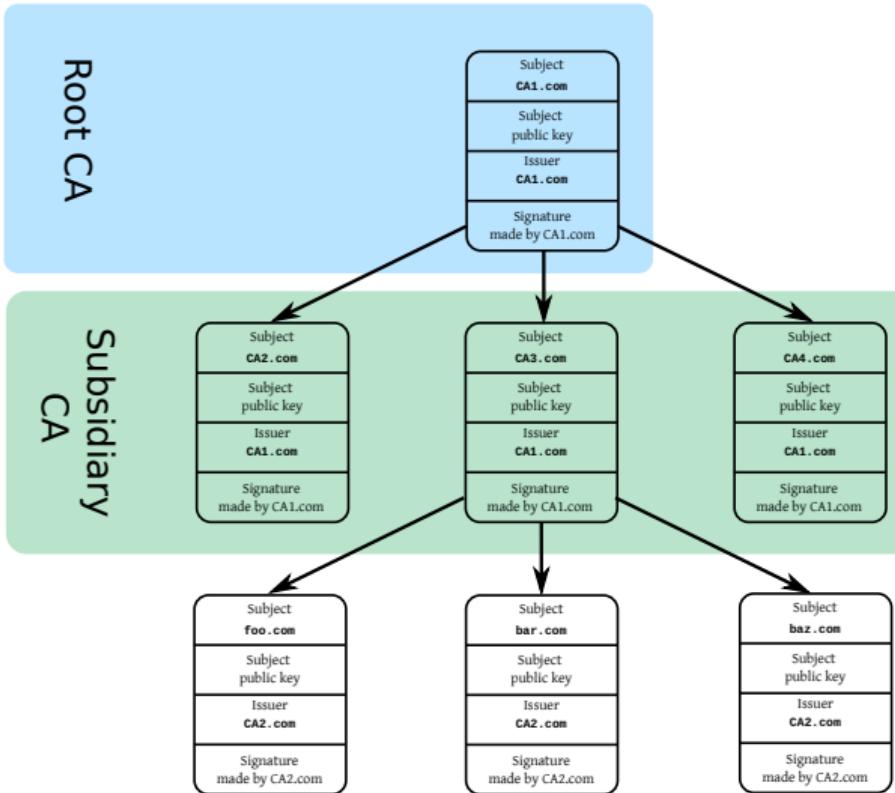
Certification authorities

Who signs the certificates

- The certificate signer is a trusted third party, the CA
- The CA public key is authenticated... with another certificate
- Up to a self-signed certificate which has to be trusted a priori



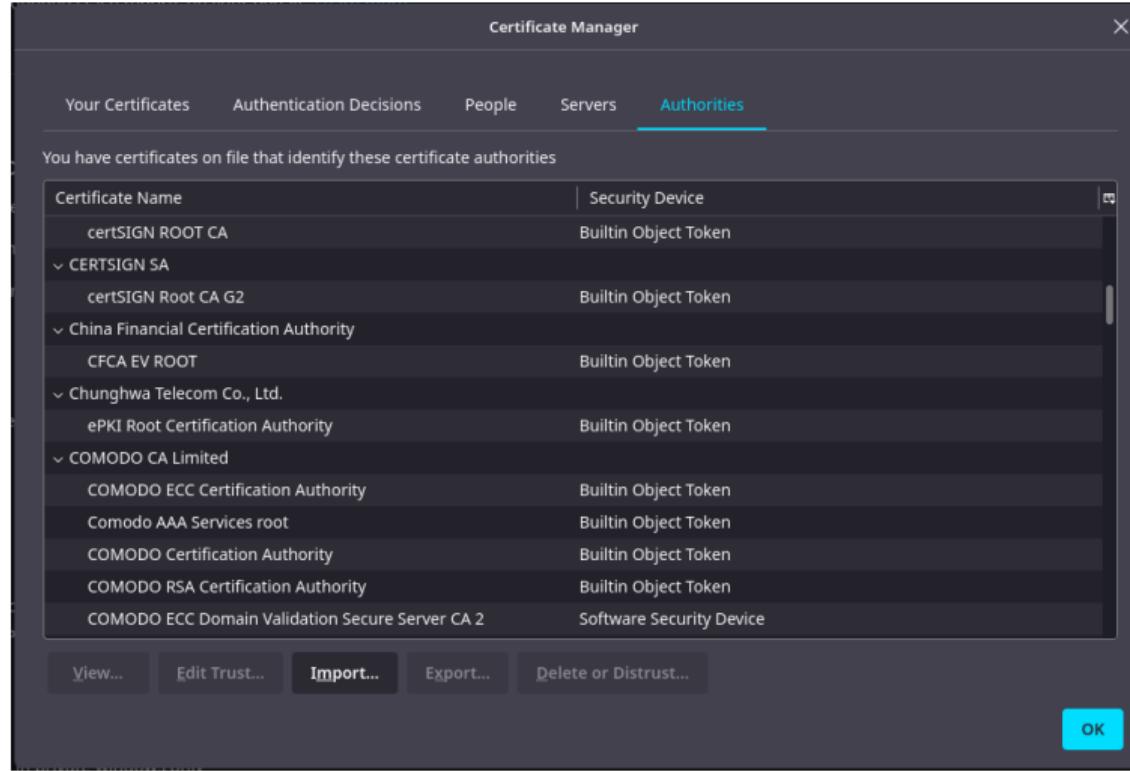
Certification authorities hierarchy



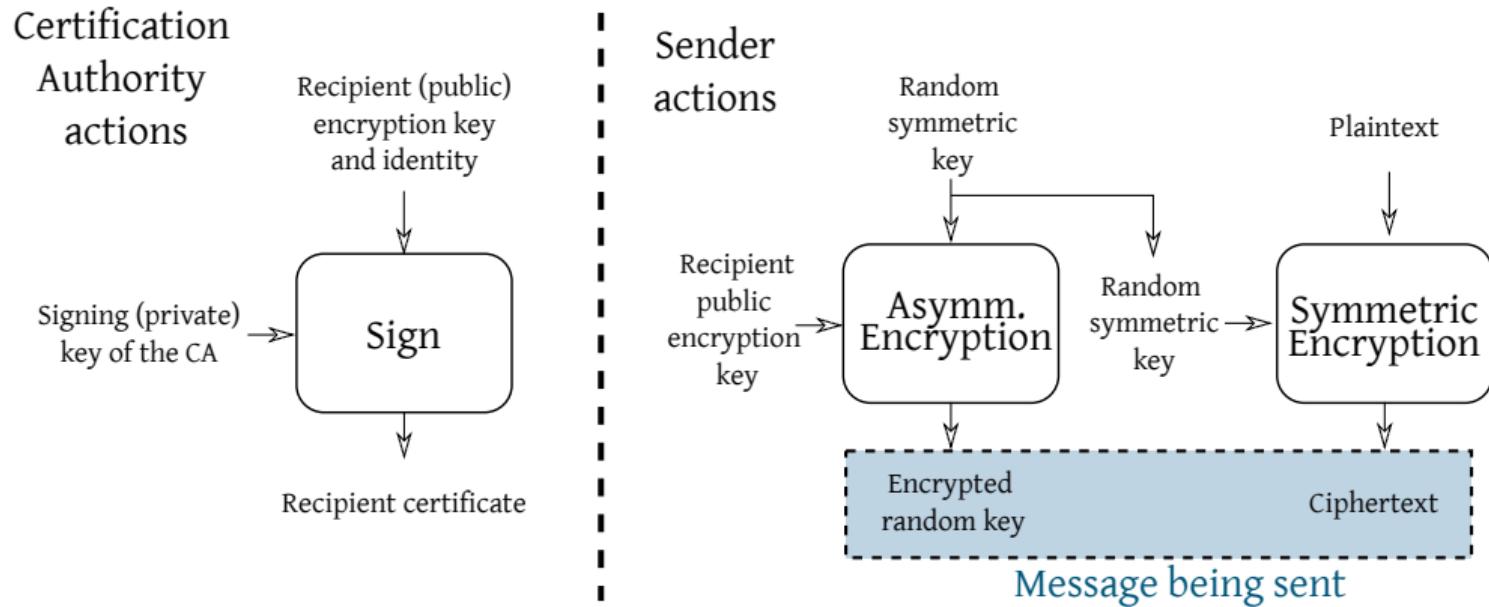
Trust with care

- 2003-02-20: Firma&Cifra was the digital signature application by PostECom
- When presented with any certificate bundled with a signed document, it considered the certificate authentic and added it to its trusted storage
- Signature forgery as easy as: 1) create your own CA certificate, 2) sign your target-user certificate, 3) sign the document
- Take away point: which certificates reside in your (applications') trusted storage determine **who you trust**

A recent browser certificate storage



Putting it all together



This way of communicating is the mainstay of modern secure comm. protocols
(TLS, OpenVPN, IPSec)

Issues to solve, features to realize

- What if we have a quantum computer?
 - Some computationally hard problems are no longer hard
 - Move away from cryptosystems based on factoring/dlog
 - Alternatives available and being standardized (2022-04)
- What if we want to compute on encrypted data?
 - Yes, but it's moderately-to-horribly inefficient
- What if the attacker has physical access to the device computing the cipher (or some way of remotely measure it)
 - Take into account **side channel** information in the attacker model

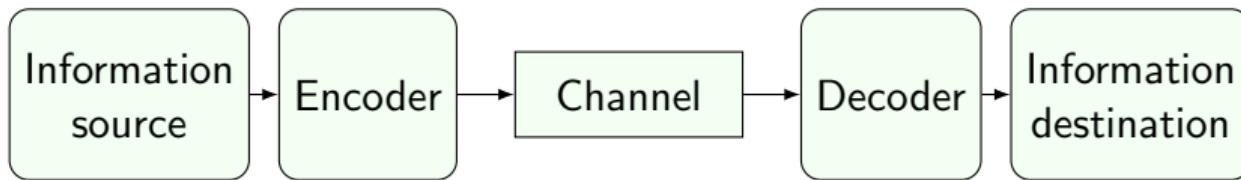
What is Shannon's information theory?

- Shannon's [3] way to mathematically frame communication
- A way to quantify information

What do we need this for (in this course)?

- Quantitatively frame “luck” and “guessing”

Basic Definitions



Basics

- A communication takes place between two **endpoints**
 - sender: made of an information source and an encoder
 - receiver: made of an information destination and a decoder
- Information is carried by a channel in the form of a sequence of **symbols** of a **finite alphabet**

Losing uncertainty = Acquiring information

- The receiver gets information only through the channel
 - it will be **uncertain** on what the next symbol is, until the symbol arrives
 - thus we model the sender as a **random variable**
- Acquiring information is modeled as getting to know an outcome of a random variable \mathcal{X}
 - the amount of information depends on the distribution of \mathcal{X}
 - intuitively: the closer is \mathcal{X} to a uniform distribution, the higher the amount of information I get from knowing an outcome
- Encoding maps each outcome as a finite sequence of symbols
 - More symbols should be needed when more information is sent

Measuring uncertainty: Entropy

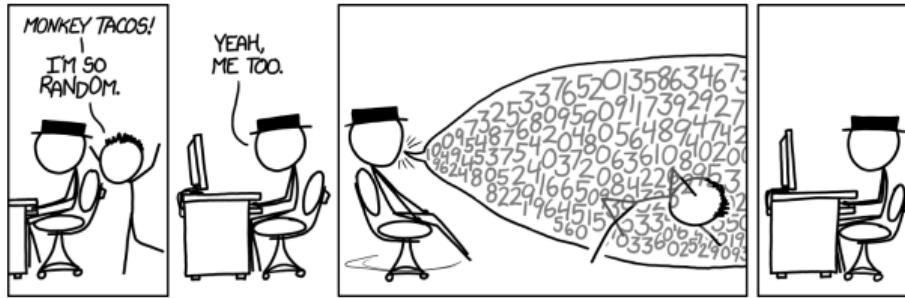
Desirable properties

- Non negative measure of uncertainty
- “combining uncertainties” should map to adding entropies

Definition

- Let \mathcal{X} be a discrete r.v. with n outcomes in $\{x_0, \dots, x_{n-1}\}$ with $\Pr(\mathcal{X} = x_i) = p_i$ for all $0 \leq i \leq n$
- The entropy of \mathcal{X} is $H(\mathcal{X}) = \sum_{i=0}^{n-1} -p_i \log_b(p_i)$
- The measurement unit of entropy depends on the base b of the logarithm: typical case for $b = 2$ is bits

Examples



<https://xkcd.com/1210/>

\mathcal{X} : Uniformly random 6 letters word

- \mathcal{X} is a sequence of 6 unif. random letters ($6^{26} \approx 3.1 \cdot 10^7$)
 - $H(\mathcal{X}) \approx \sum_{i=0}^{3.1 \cdot 10^7} -\frac{1}{3.1 \cdot 10^7} \log_b \left(\frac{1}{3.1 \cdot 10^7} \right) \approx 28.2b$
- \mathcal{X} is a uniform pick from 6-letters English words
 - $H(\mathcal{X}) \approx \sum_{i=0}^{6300} -\frac{1}{6300} \log_b \left(\frac{1}{6300} \right) \approx 12.6b$

Statement (informal)

It is possible to encode the **outcomes n of i.i.d. random variables**, each one with entropy $H(\mathcal{X})$, into **no less than $nH(\mathcal{X})$ bits** per outcome. If $< nH(\mathcal{X})$ bits are used, some information will be lost.

Consequences

- Arbitrarily compression of bitstrings is impossible without loss
 - Cryptographic hashes must discard some information
- Guessing a piece of information (= one outcome of \mathcal{X}) is at least as hard as guessing a $H(\mathcal{X})$ bit long bitstring
 - overlooking for a moment the effort of decoding the guess

A practical mismatch

- It is possible to have distributions with the same entropies

Plucking low-hanging fruits

- We define the **min-entropy** of \mathcal{X} as $H_{\infty}(\mathcal{X}) = -\log(\max_i p_i)$
- Intuitively: it's the entropy of a r.v. with uniform distribution, where the probability of each outcome is $(\max_i p_i)$
- Guessing the most common outcome of \mathcal{X} is at least as hard as guessing a $H_{\infty}(\mathcal{X})$ bit long bitstring

Example

A very biased r.v.

Consider \mathcal{X} :
$$\begin{cases} \mathcal{X} = 0^{128} & \text{with } \Pr \frac{1}{2} \\ \mathcal{X} = a, a \in 1\{0, 1\}^{127} & \text{with } \Pr \frac{1}{2^{128}} \end{cases}$$

Intuition and quantification

- Predicting an outcome shouldn't be too hard: just say 0^{128}
- $H(\mathcal{X}) = \frac{1}{2}(-\log_2(\frac{1}{2})) + 2^{127} \frac{1}{2^{128}} (-\log_2(\frac{1}{2^{128}})) = 64.5\text{b}$
- $H_\infty(\mathcal{X}) = -\log_2(\frac{1}{2}) = 1\text{b}$
- Min-entropy tells us that guessing the most common output is as hard as guessing a single bit string

Bibliography I

-  Giovan Battista Bellaso.
La cifra del sig. giovan battista bellaso, 1553.
-  John Nash.
Personal communication to the us national security agency, Feb. 1955.
-  Claude E. Shannon.
A mathematical theory of communication.
Bell Syst. Tech. J., 27(3 and 4):379–423 and 623–656, 1948.
-  Claude E. Shannon.
Communication theory of secrecy systems.
Bell Syst. Tech. J., 28(4):656–715, 1949.
-  Marc Stevens.
The hashclash project, 2009.
-  Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov.
The first collision for full SHA-1.
In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 570–596. Springer, 2017.

Bibliography II



Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger.

Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate.

In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2009.