Introduction

(historical background, security objectives, attacks and adversaries)

Security is meant to compensate the lack of trust

- Trust is a wonderful thing!

- We can safely buy food from a super-market

- Take pills as prescribed, without knowing what they do

- Enjoy a concert along hundreds/thousands of strangers

- Fly at a few thousand meters in the sky without knowing the pilot

Images: https://en.wikipedia.org

But … unfortunately, there are adversaries

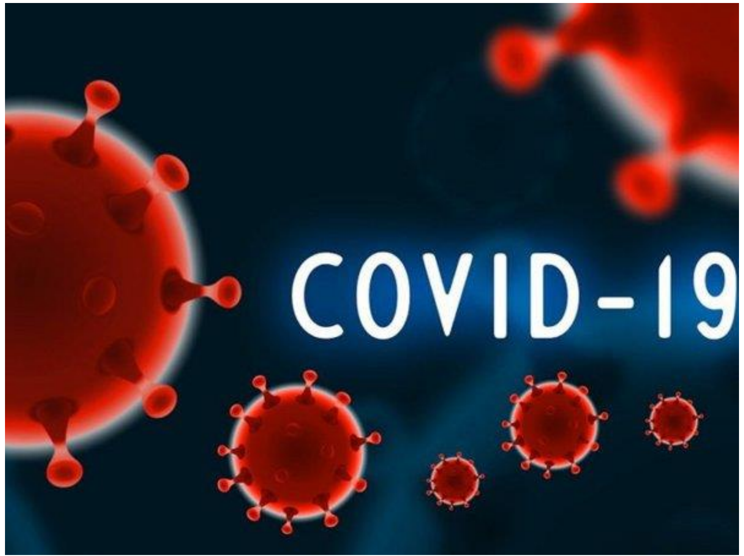
I) Burglars

II) Terrorists

III) And even **nature** itself (unwillingly) can become an adversary

2004 – Tsunami in the Indian Ocean

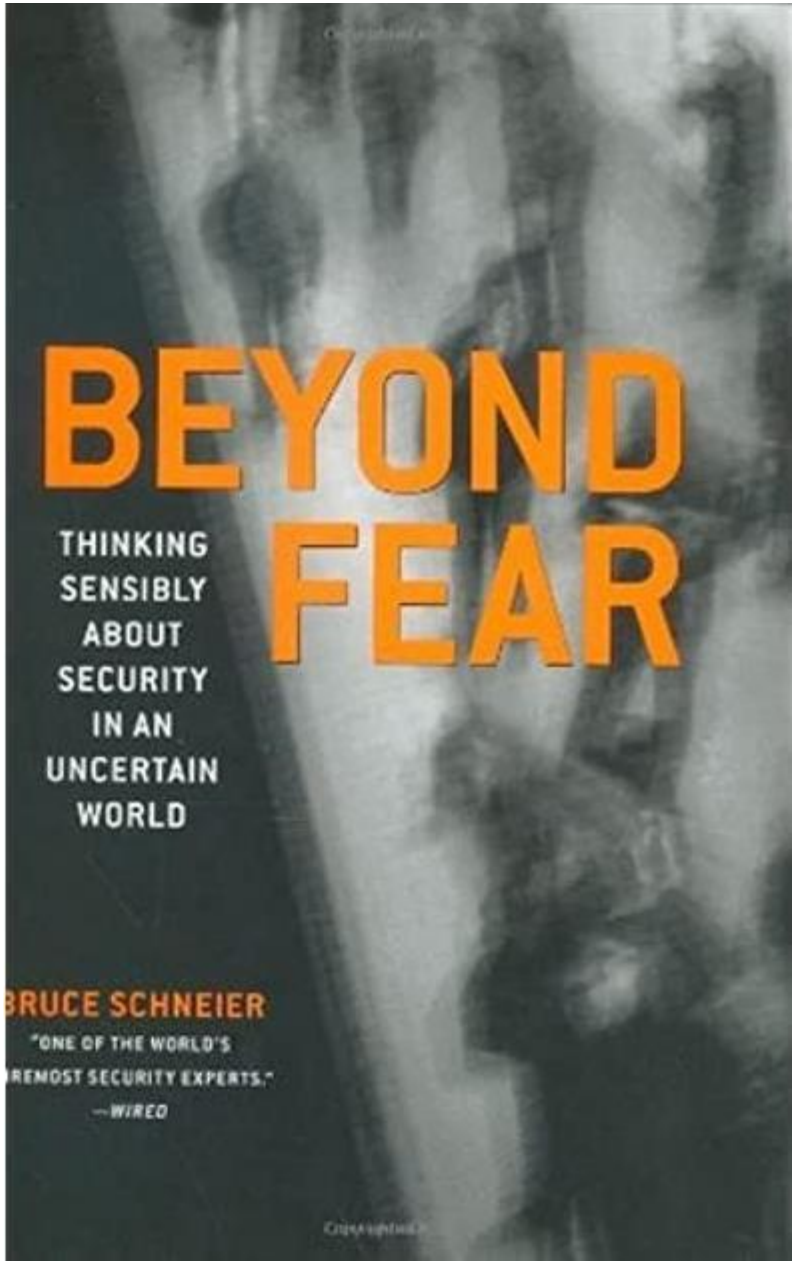
2019 – CORONAVIRUS pandemic



Solution?

- According to Bruce Schneier:

"We need to move beyond fear and start making sensible security trade-offs ... Once you move beyond fear and start thinking sensibly about trade-offs, you will be able to recognize bad or overpriced security when you see it. "



Information Security - InfoSec

- **Definition according to U.S. Code, Chapter 35**
  - means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide
    - (A) integrity, which means guarding against improper information modification or destruction, and includes ensuring information nonrepudiation and authenticity;
    - (B) confidentiality, which means preserving authorized restrictions on access and disclosure, including means for protecting personal privacy and proprietary information; and
    - (C) availability, which means ensuring timely and reliable access to and use of information.

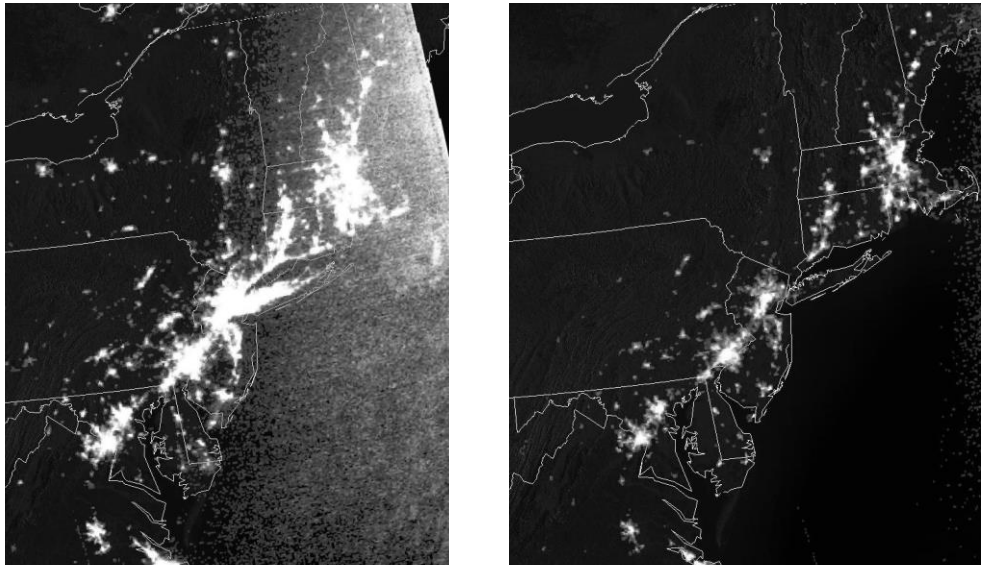
- **It may be instructive to reflect upon the following statements**
  - Security is a process, not a product
  - A system is only as secure as its weakest component (the weakest link)
  - A defender must cover all points of attack, the adversary need to find only a suitable one
  - Finally, security is merely a trade-off (at least between usability, costs and security level)

- **You may also want to have in mind the following:** Security Vulnerabilities + Adversaries => Security Risks

Motivation

- The InfoSec field is generally "**incident driven**" (if nothing happens, nobody cares)
- **Reactive** thinking, however, comes at times with staggering costs
- **Pro-active** thinking may be beneficial in minimizing costs

East Coast Blackout of 2003



What you should avoid

- **Deprecated principles: security through obscurity & isolated environments**
- Note that:
  - Moving to open standards reduces costs (you don't have to pay your own experts for designing security, but rather use what already exists)
  - There are no more isolated systems, e.g., the cloud is ubiquitous and pervasive

Adversaries (some examples)

- Depending on the target/context, there are many:
  - **Hackers**: usually with low financial resources, trying to impress or having fun
  - **Clients**: usually with low computational resources, interested in economic advantages
  - **Companies**: usually with average computation resources, interested in economic advantages
  - **Organized crime**: usually with low computational resources, interested in economic advantages
  - **Terrorists**: may have significant financial resources, driven by political reasons
  - **Governments**: high computational and financial resources, strategic interests
  - **Insiders**: not much resource, but they have the know-how, motivated by financial interests

Remember the most dangerous adversary

- Combos of the previous, e.g., hackers with insiders, etc.

Security objectives

- In the past: the **CIA** triad - **Confidentiality, Integrity, Availlability**

- Followed by **PAIN** - **Privacy, Availlability-Authentication, Integrity, Non-repudiation**

- Today, 4 objectives acknowledged by most books in cryptography:

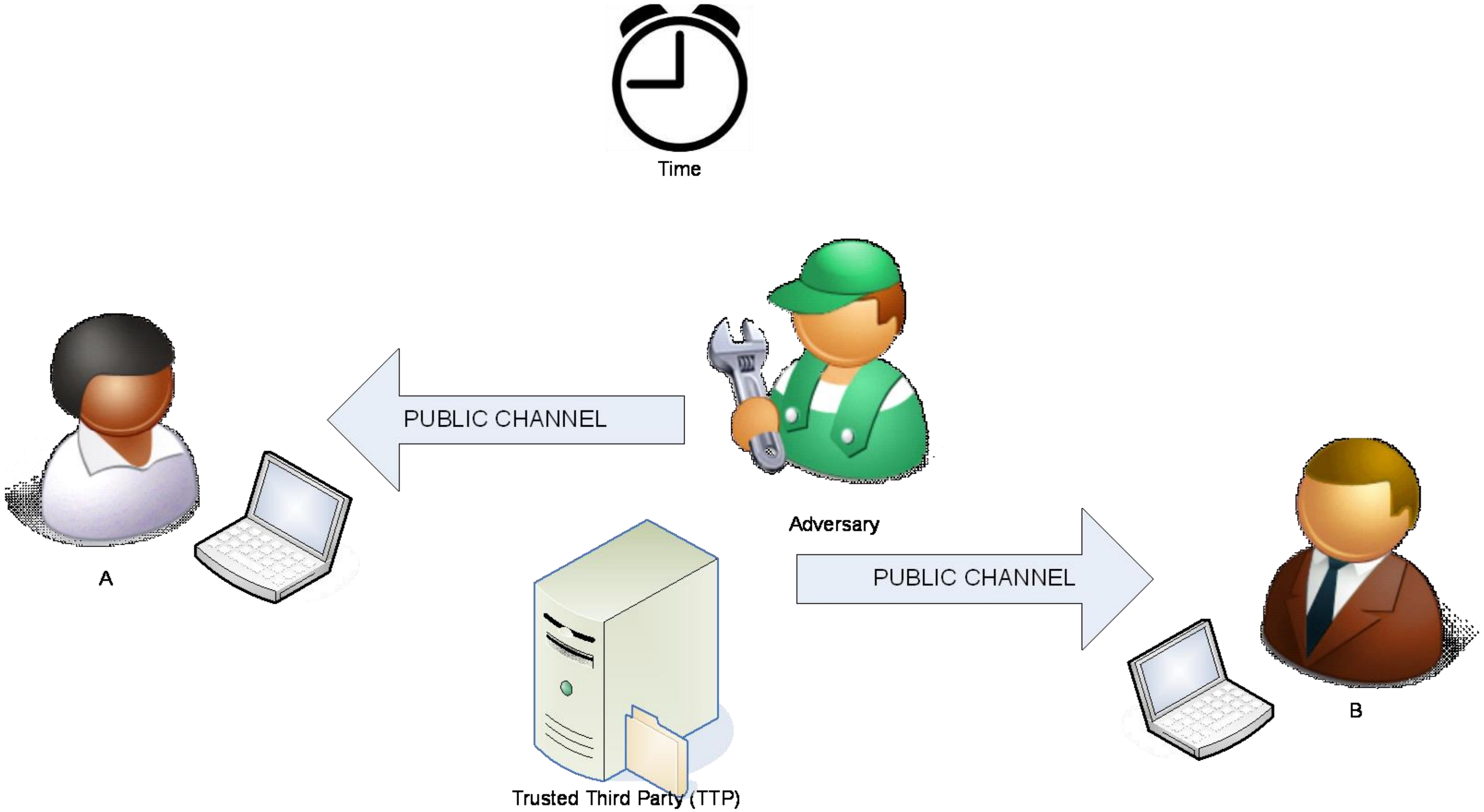
  - **Confidentiality** – information can be accessed only by authorized parties
  - **Integrity –** information was not altered
  - **Authentication** – entity authentication (identification, prove the identity of a principal) and message authentication (bind a message with an entity)
  - **Non-repudiation** – prevents an entity from denying an action

- But many other objectives exist as well:

  - **Freshness, Anonymity, Authorization, Availability, Third-party protection, Revocation, Traceability, etc.**

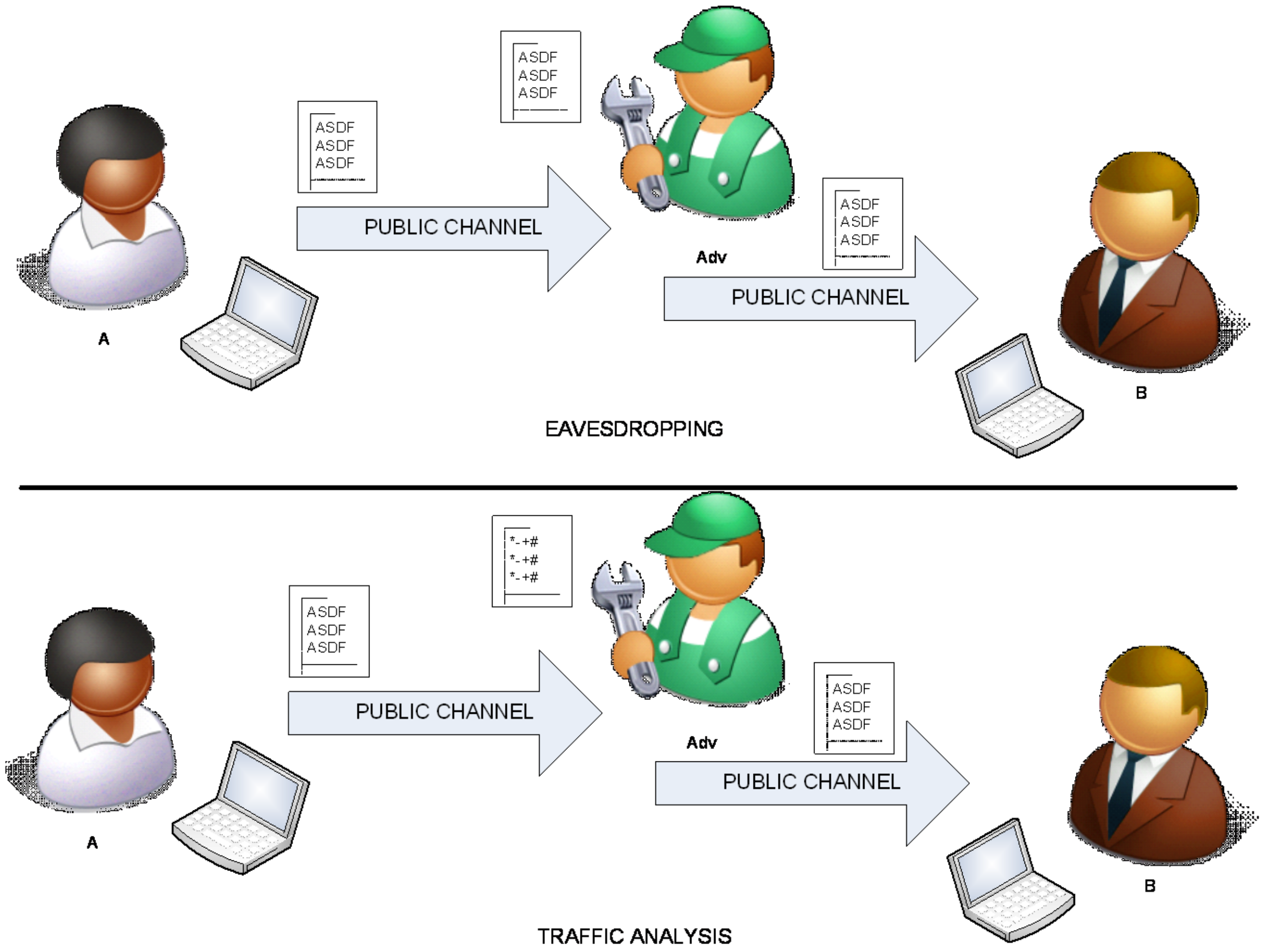
Generic setting (the communication channel)

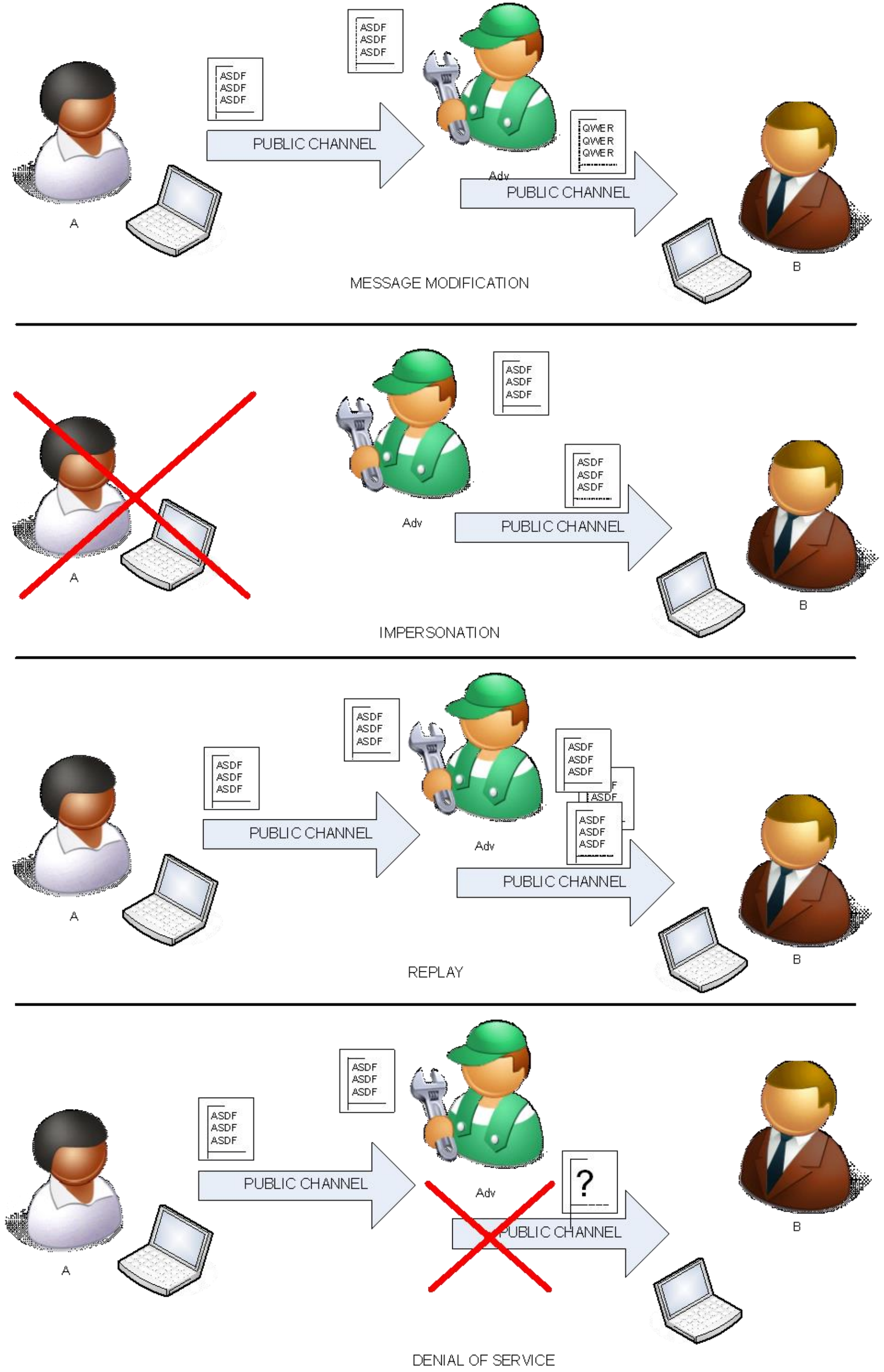
- Usual components:
  - Honest principals A and B
  - An adversary
  - A trusted third party TTP
  - Time



Generic attacks

- **Passive attacks**: eavesdropping and traffic analysis
- **Active attacks**: modification, impersonation, replay, denial-of-service





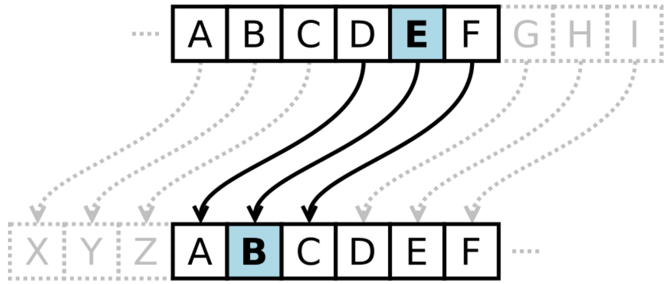
Cryptography and cryptanalysis

- **Cryptography** – the science of designing codes and protocols that block adversaries
- **Cryptanalysis** – the science of breaking codes and protocols
- **Cryptology** – the field comprising cryptography and cryptanalysis

Stage I – Classic cryptography

- **Antiquity**, the Caesar cipher (a mono-alphabetic substitution, i.e., a letter is changed with exactly another one), scythale, etc.
- **Renaissance**, Vigenere cipher (originally invented by Bellaso), or the era of poly-alphabetic substitutions, i.e., a letter will encrypt to more than a single letter

"Skytale". Licensed under CC BY-SA 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Skytale.png#mediaviewer/File:Skytale.png



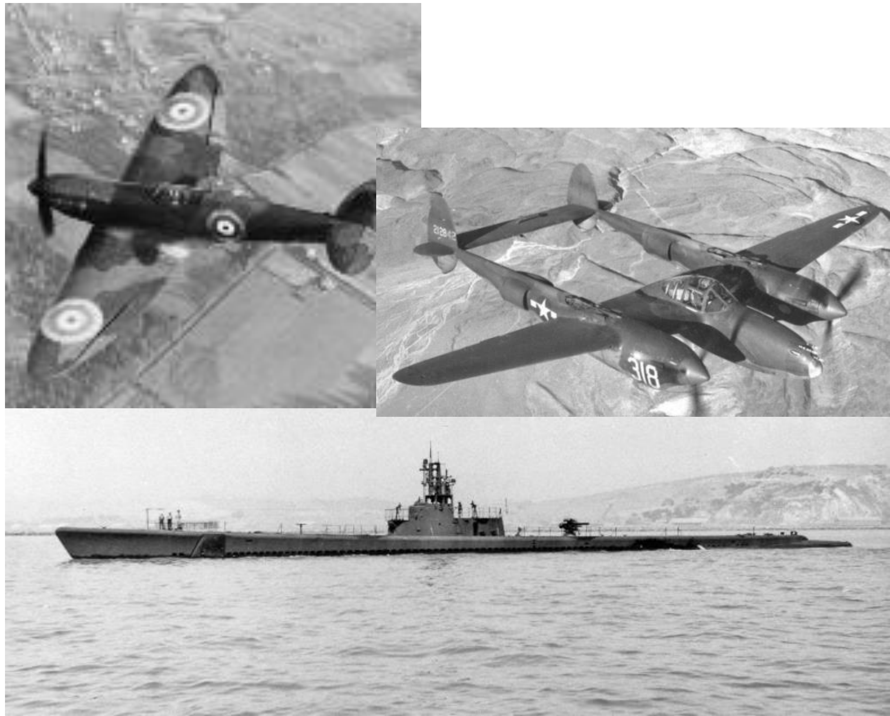
"Caesar cipher left shift of 3" by Matt_Crypto - http://en.wikipedia.org/wiki/File:Caesar3.png. Licensed under Public Domain via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Caesar_cipher_left_shift_of_3.svg#mediaviewer/File:Caesar_cipher_left_shift_of_3.svg

Stage II – Pre-modern cryptography, World War II

- First large-scale use of cryptographic designs in the real-world: the Enigma machine (used by Germans), Purple (used by Japanese), Playfair (used by British forces), etc.
- Fundamental works of Shannon (i.e., information theory) and Turing (i.e., breaking the Enigma machine), etc.

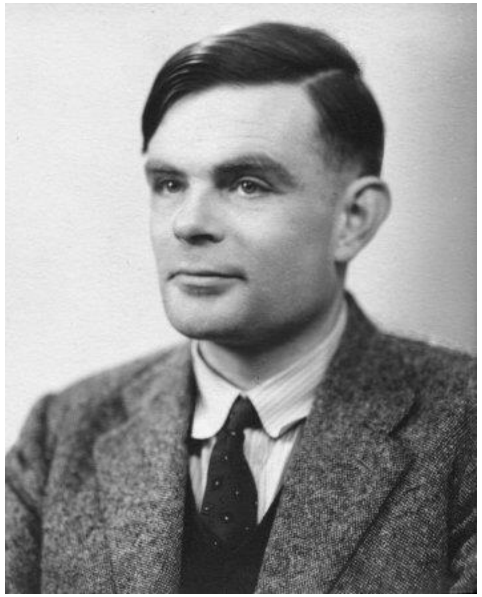
Why was cryptography so important during WW2?

- The most valuable weapons depend on wireless communications, and without secure (encrypted) communications they are useless



Historical figures (early cryptographers)

- Alan Turing (1912-1954)



"Alan Turing photo" by Source. Licensed under **Fair use via Wikipedia** - http://en.wikipedia.org/wiki/File:Alan_Turing_photo.jpg#mediaviewer/File:Alan_Turing_photo.jpg

- Fields: computer science, mathematics, cryptanalysis, computational biology
- Known for: major role in breaking the Enigma machine, first formalization of a general purpose computer (the Turing Machine), first test for machine intelligent behavior (the Turing test)

- Claude Shannon (1916-2001)

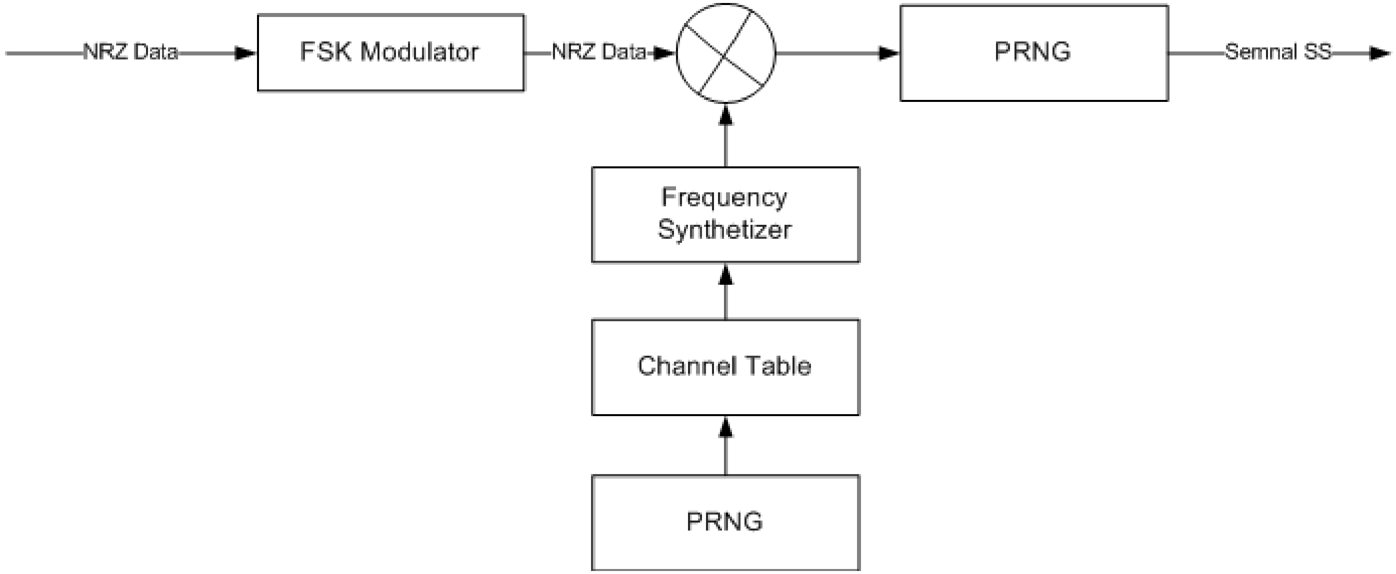
"Claude Elwood Shannon (1916-2001)" by Source. Licensed under **Fair use via Wikipedia** - http://en.wikipedia.org/wiki/File:Claude_Elwood_Shannon_(1916-2001).jpg#mediaviewer/File:Claude_Elwood_Shannon_(1916-2001).jpg

- Fields: computer science, mathematics, cryptanalysis
- Known for: father of information theory, founder of the digital computer and digital circuit design theory

WW2 grade cryptosystems
(more technical details in forthcoming lectures)

The Enigma Machine

Frequency hopping spread spectrum
(discovered among others by Hedy Lamarr)



Stage IV – modern times, mid 1970 - today

- Spectacular growth and indisputable relevance in the digital age
- Dozens of cryptographic designs DES, AES, RSA, DSA, and protocols SSL/TLS, SSH, IPSec, WEP, WPA, etc. – all to be discussed in forthcoming lectures

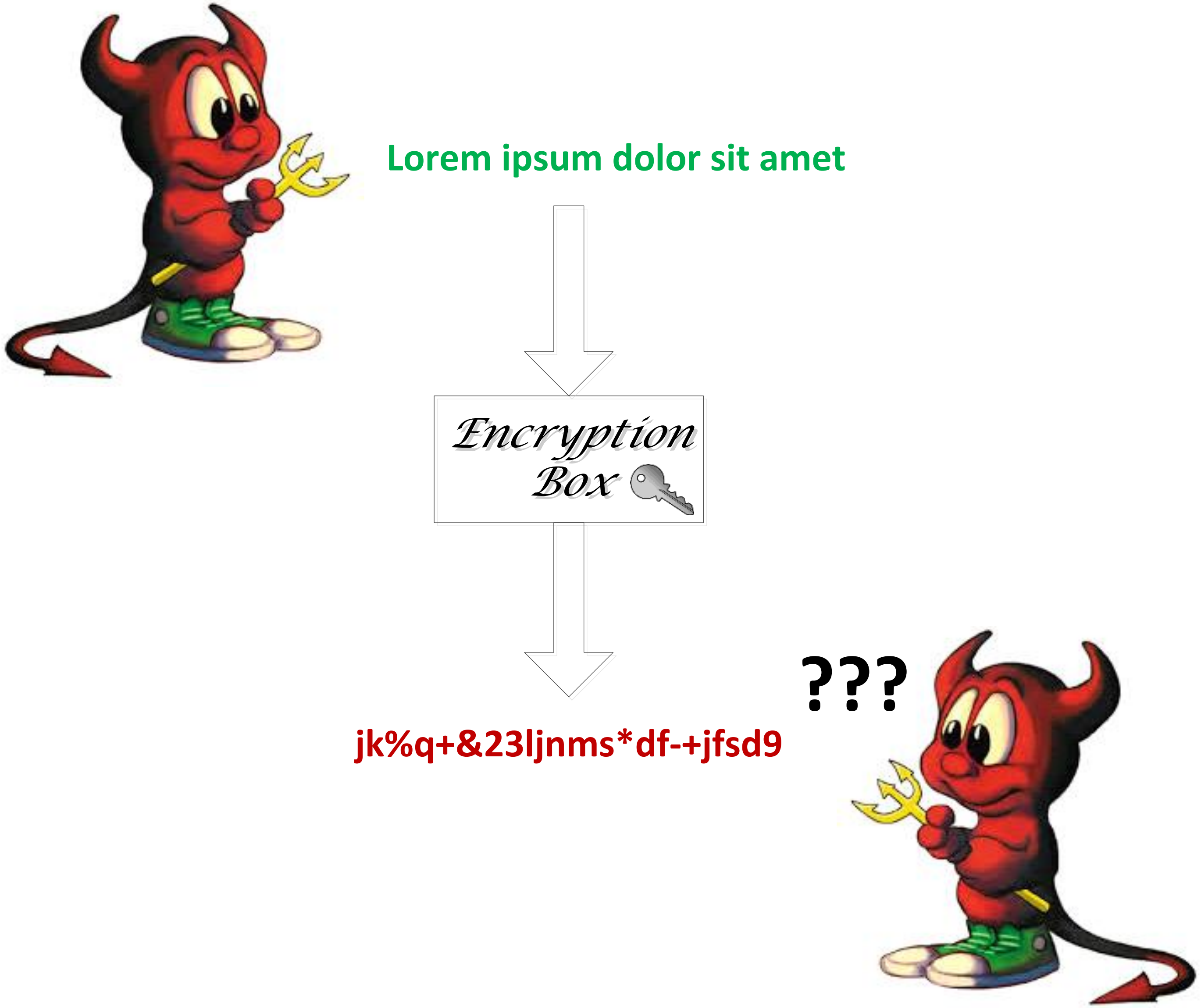
Symmetric Primitives

(block ciphers, stream ciphers, hash functions, keyed hash functions and (pseudo)random number generators)

An informal, yet instructive account of symmetric primitives …

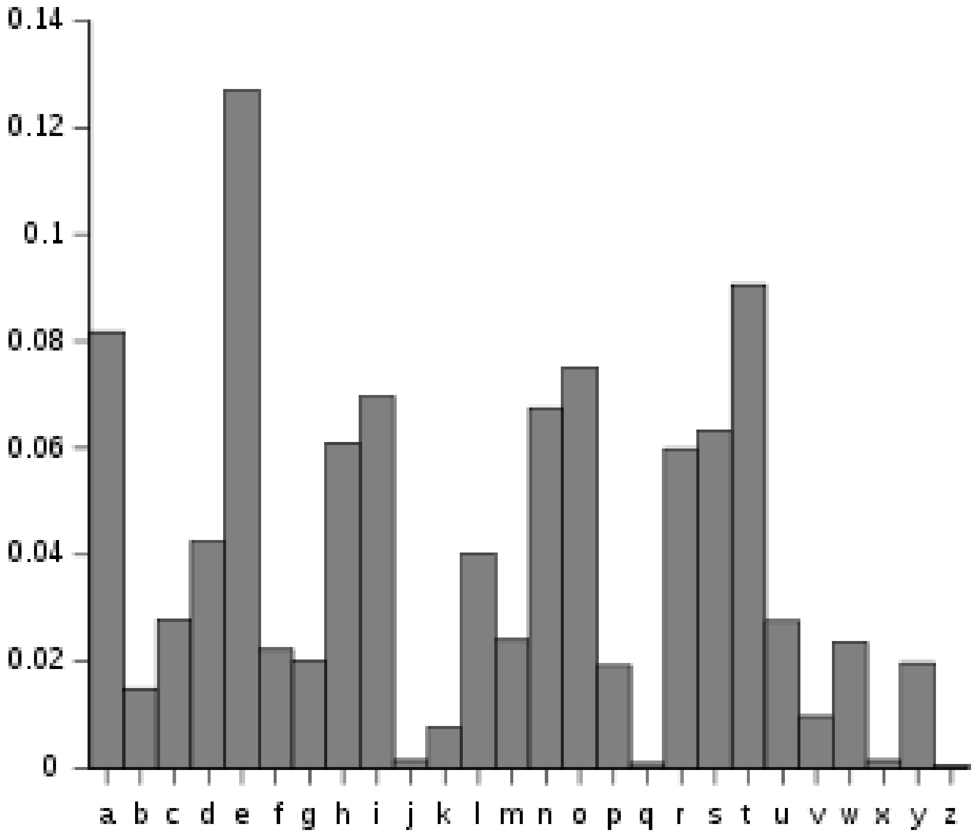
Begin with an informal question

- Question: What do you expect from cryptography?
- (Potentially correct) Answer: Protect your stored data & ongoing communications (let's call this simply protect messages)
- Question: Assume you are given am encryption box (call it symmetric encryption) that encrypts your data with a key. Is your data now protected?
- (At least incomplete) Answer: Yes, as long as the adversary cannot find/guess the key … or maybe not



Monoalphabetical substitutions

- Main idea: replace one symbol with another
- Problem: easy to break by frequency analysis
- Extension: bigram analysis (and beyound)



https://en.wikipedia.org/wiki/Frequency_analysis

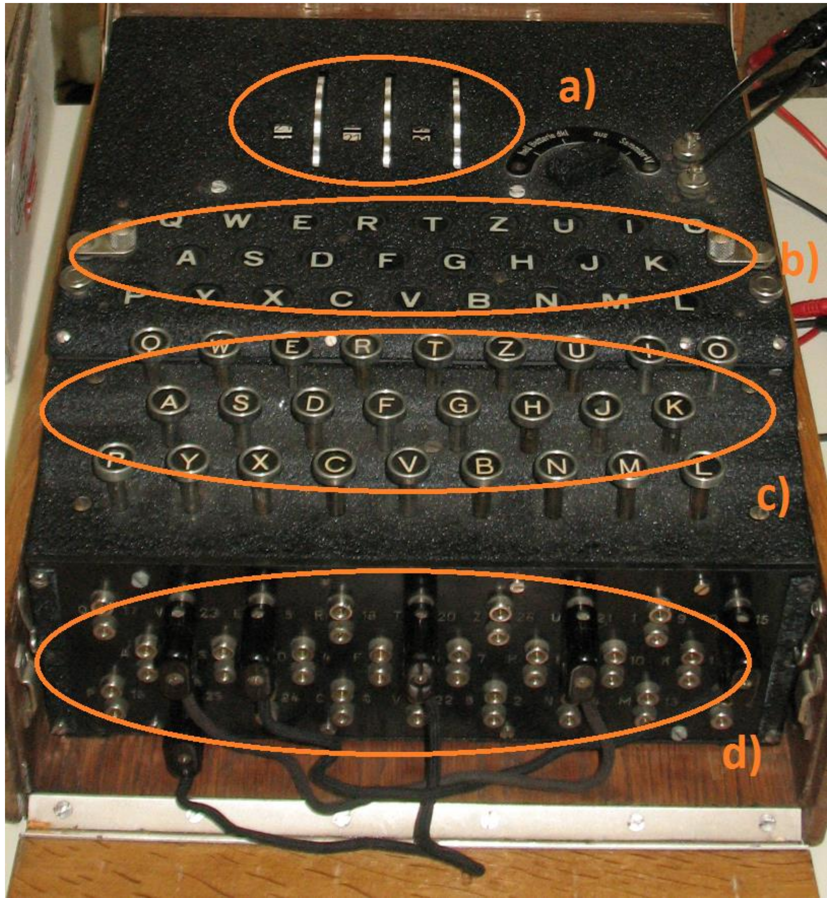
th 1.52	en 0.55	ng 0.18
he 1.28	ed 0.53	of 0.16
in 0.94	to 0.52	al 0.09
er 0.94	it 0.50	de 0.09
an 0.82	ou 0.50	se 0.08
re 0.68	ea 0.47	le 0.08
nd 0.63	hi 0.46	sa 0.06
at 0.59	is 0.46	si 0.05
on 0.57	or 0.43	ar 0.04
nt 0.56	ti 0.34	ve 0.04
ha 0.56	as 0.33	ra 0.04
es 0.56	te 0.27	ld 0.02
st 0.55	et 0.19	ur 0.02

https://en.wikipedia.org/wiki/Bigram

A practical example of polyalphabetical substitution

- The Enigma machine, a rotor cipher (several versions exist), elements:
  - 26 lamps (output, ciphertext) & keys (input, plaintext)
  - 3 or 5 (usually) rotors
  - at most 13 plugs that can connect each two letters on the plug-board (part of the key)

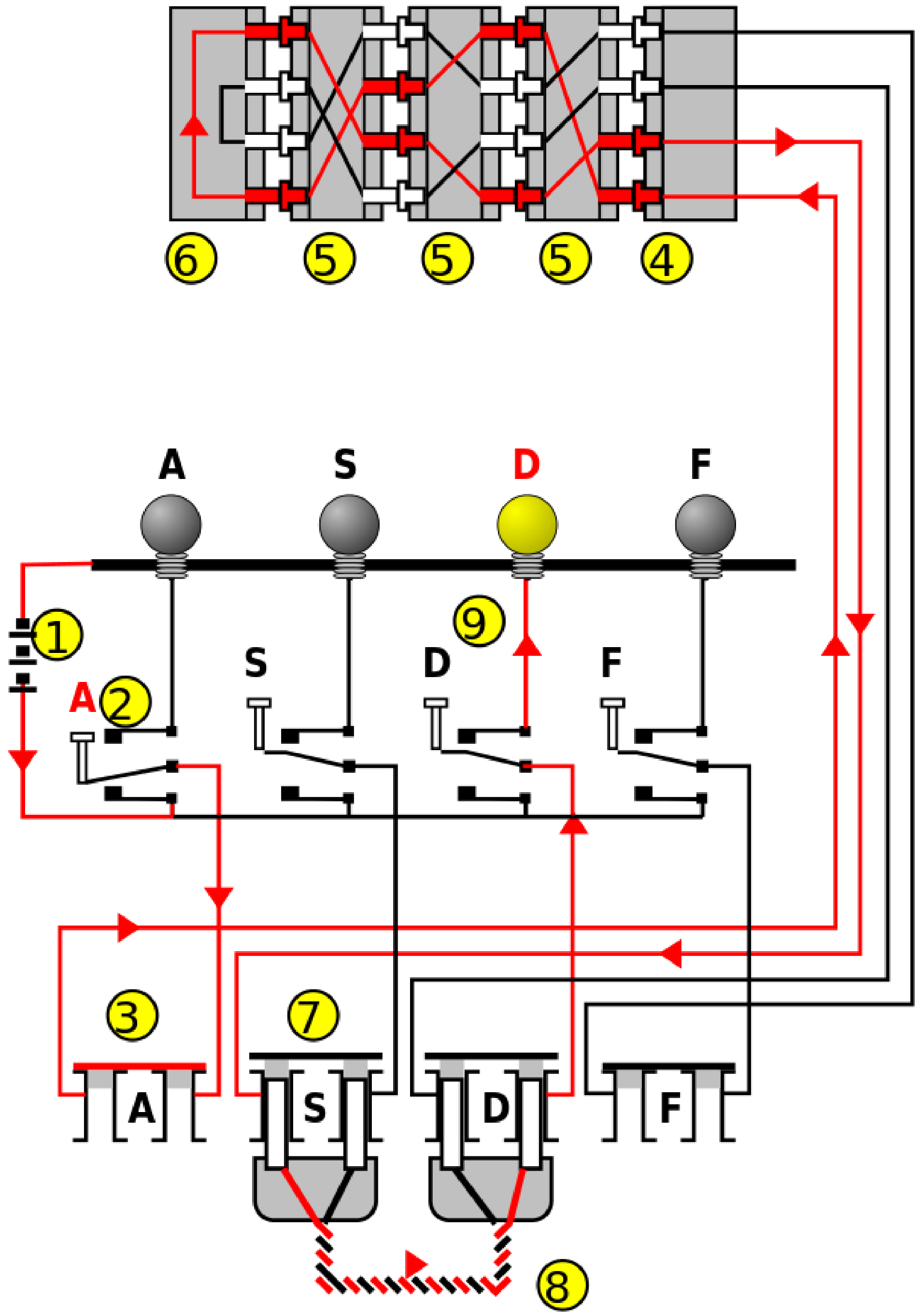
a) rotors (3)
b) lamps (26)
c) keys (26)
d) plugboard (2x13)



How Enigma works

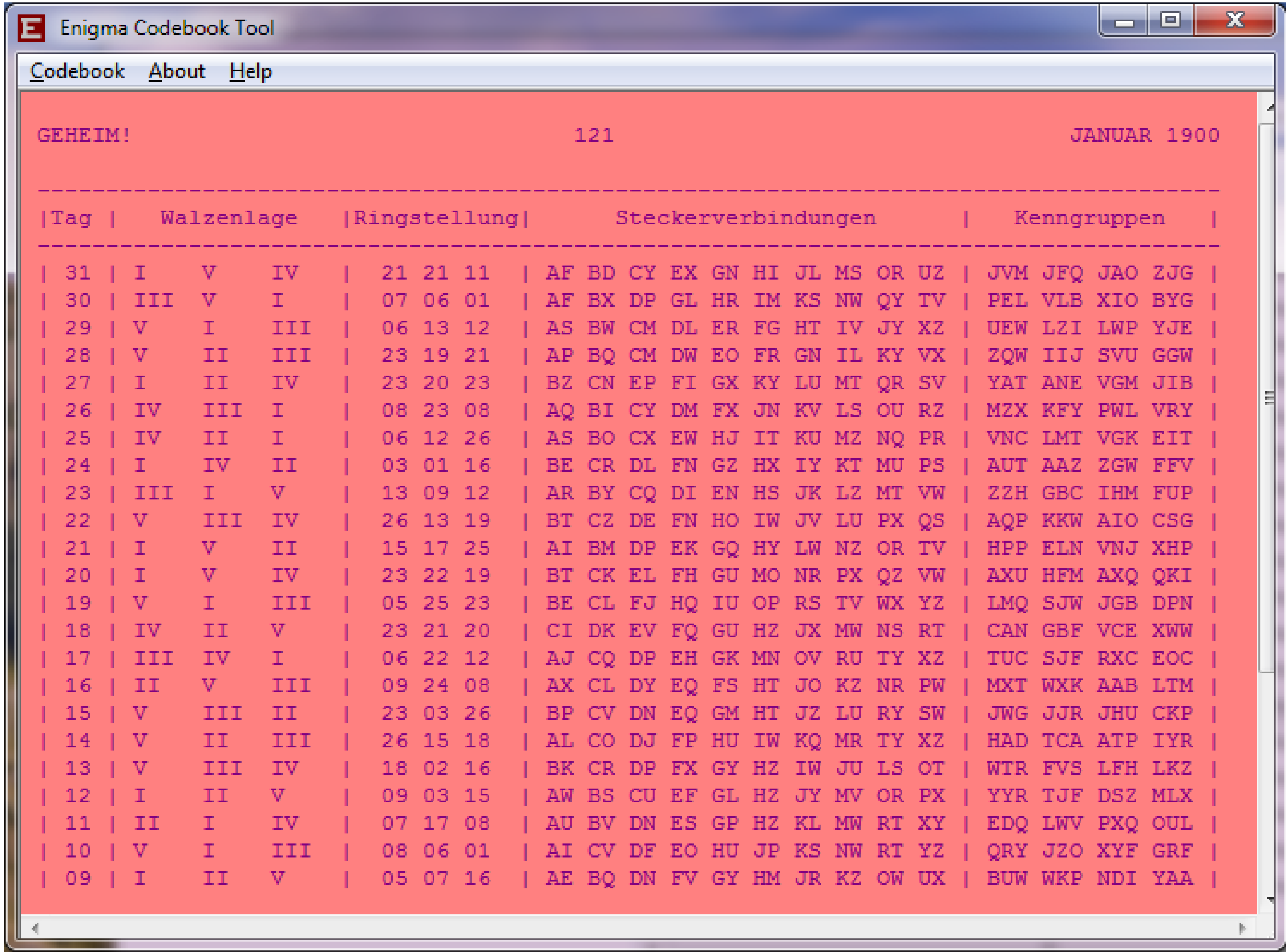
- When one key is pressed (letter of the plaintext selected), circuit is closed under that key, current flows through the plugboard that follows the 3 rotors, returns from the reflector and lightens up the lamp (the letter of the ciphertext)
- Rotors move at each step, thus a character will not always get encrypted to the same character (i.e., a polyalphabetic substitution)

© image from wikipedia.org



- A closer look to the Enigma secret key (depicted in the form of a codebook) may give you more insights on the security of this cryptosystem

- This is a print-screen from a nice tool by Dirk Rijmenants, see http://rijmenants.blogspot.ro/2005/11/enigma-codebook-tool.html



GEHEIM! 121 JANUAR 1900

Tag	Walzenlage	Ringstellung	Steckerverbindungen	Kenngruppen
31	I V IV	21 21 11	AF BD CY EX GN HI JL MS OR UZ	JVM JFQ JAO ZJG
30	III V I	07 06 01	AF BX DP GL HR IM KS NW QY TV	PEL VLB XIO BYG
29	V I III	06 13 12	AS BW CM DL ER FG HT IV JY XZ	UEW LZI LWP YJE
28	V II III	23 19 21	AP BQ CM DW EO FR GN IL KY VX	ZQW IIJ SVU GGW
27	I II IV	23 20 23	BZ CN EP FI GX KY LU MT QR SV	YAT ANE VGM JIB
26	IV III I	08 23 08	AQ BI CY DM FX JN KV LS OU RZ	MZX KFY PWL VRY
25	IV II I	06 12 26	AS BO CX EW HJ IT KU MZ NQ PR	VNC LMT VGK EIT
24	I IV II	03 01 16	BE CR DL FN GZ HX IY KT MU PS	AUT AAZ ZGW FFV
23	III I V	13 09 12	AR BY CQ DI EN HS JK LZ MT VW	ZZH GBC IHM FUP
22	V III IV	26 13 19	BT CZ DE FN HO IW JV LU PX QS	AQP KKW AIO CSG
21	I V II	15 17 25	AI BM DP EK GQ HY LW NZ OR TV	HPP ELN VNJ XHP
20	I V IV	23 22 19	BT CK EL FH GU MO NR PX QZ VW	AXU HFM AXQ QKI
19	V I III	05 25 23	BE CL FJ HQ IU OP RS TV WX YZ	LMQ SJW JGB DPN
18	IV II V	23 21 20	CI DK EV FQ GU HZ JX MW NS RT	CAN GBF VCE XWW
17	III IV I	06 22 12	AJ CQ DP EH GK MN OV RU TY XZ	TUC SJF RXC EOC
16	II V III	09 24 08	AX CL DY EQ FS HT JO KZ NR PW	MXT WXK AAB LTM
15	V III II	23 03 26	BP CV DN EQ GM HT JZ LU RY SW	JWG JJR JHU CKP
14	V II III	26 15 18	AL CO DJ FP HU IW KQ MR TY XZ	HAD TCA ATP IYR
13	V III IV	18 02 16	BK CR DP FX GY HZ IW JU LS OT	WTR FVS LFH LKZ
12	I II V	09 03 15	AW BS CU EF GL HZ JY MV OR PX	YYR TJF DSZ MLX
11	II I IV	07 17 08	AU BV DN ES GP HZ KL MW RT XY	EDQ LWV PXQ OUL
10	V I III	08 06 01	AI CV DF EO HU JP KS NW RT YZ	QRY JZO XYF GRF
09	I II V	05 07 16	AE BQ DN FV GY HM JR KZ OW UX	BUW WKP NDI YAA

How secure is Enigma

- Question: how hard is to break Enigma?
- Answer (not necessarily correct): as hard as to find the key
- Question: how big is Enigma’s key?
- Answer: consider just (the way to place 3 rotors) x (the way to connect 13 plugs)

$$26^3 \times \frac{26!}{13! \times 2^{13}} = 138953282533065000$$

when compared to the number of DES keys $2^{56} = 72057594037927936$ will quickly lead to the conclusion that Enigma (deprecated by the end of WW2) is stronger than DES (deprecated only by the end of the ‘90s)

How secure is Enigma

- Question: imagine you have captured a ciphertext that begins with:

zeyt sadb eiwf dsak sadk jnujj

Could you tell which is the corresponding plaintext from the following:

a) attackatdawnonthewestfront
b) attackatnightonthewestfront
c) attackatduskonthewestfront

- Answer: wrong design decision in Enigma, a letter cannot map to itself! Correct answer is c)

Partial conclusion

- For protecting data by symmetric primitives we need: **clear design principles** (how to build the ciphers) and a **formal treatment of security properties** (what is the exact security they should offer)

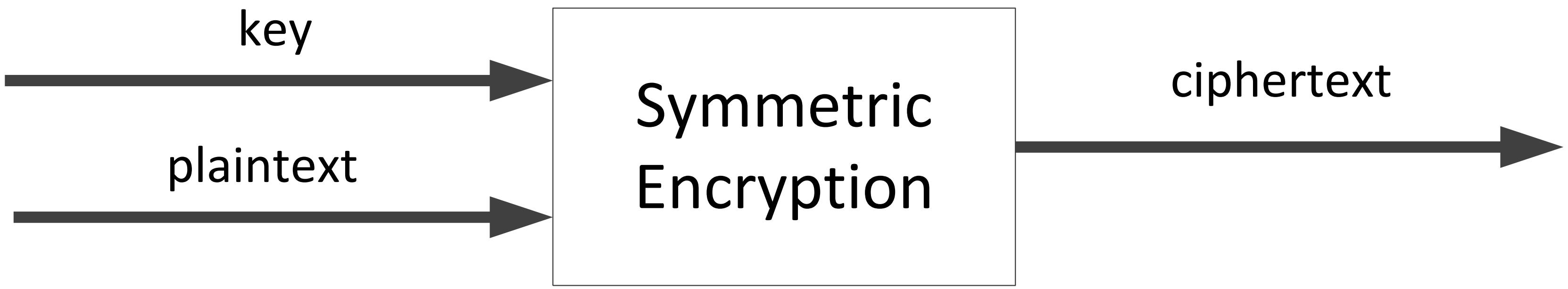
A more formal and constructive account of symmetric primitives …

you should learn:

i. where is the primitive used,
ii. what are the standards,
iii. how is it built,
iv. what are its properties

Type of functions (I) Symmetric encryption schemes

- Description (informal): an algorithm that takes as input a key k and message m called plaintext and returns the encrypted message c called ciphertext (similarly, algorithms for decryption and generating keys are needed)



i.e., $c = e_k(m)$

- Example of use: encrypted tunnels SSL/TLS, IPSEC; encrypted passwords (lmhash in Win XP); encrypted hard drives (TrueCrypt), etc.
- Standards:
  - ➢Not to use: DES, RC4
  - ➢To use: AES (128, 194, 256), 3DES (with 168 bit key, not recommended)

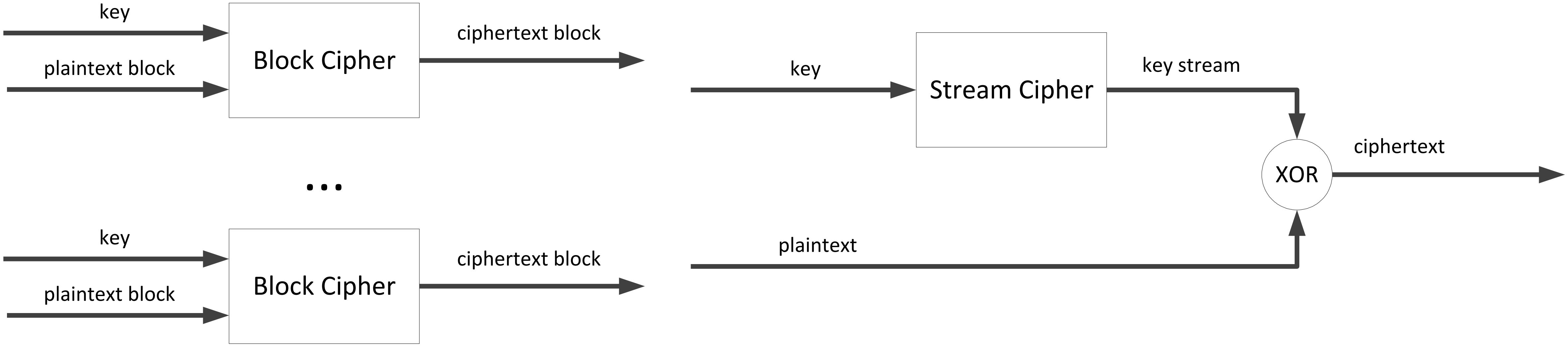
Symmetric encryption: formal definition

- A symmetric encryption scheme is a **triple of algorithms**:
  - ***Gen*** *is the key generation algorithm that takes random coins, a security parameter (l) and outputs the key* $k \leftarrow Gen(1^l)$
  - ***Enc*** *is the encryption algorithm that takes as input the key and some message, then outputs the ciphertext* $c \leftarrow Enc(k, m)$
  - ***Dec*** *is the decryption algorithm that takes as input the ciphertext and the key and outputs the message* $m \leftarrow Dec(k, c)$

- A correctness condition enforces that $Dec\big(k, Enc(k, m)\big) = m$
- In some cases, the encryption and decryption algorithms are allowed to return \null on particular inputs (i.e., they refuse to encrypt/decrypt)

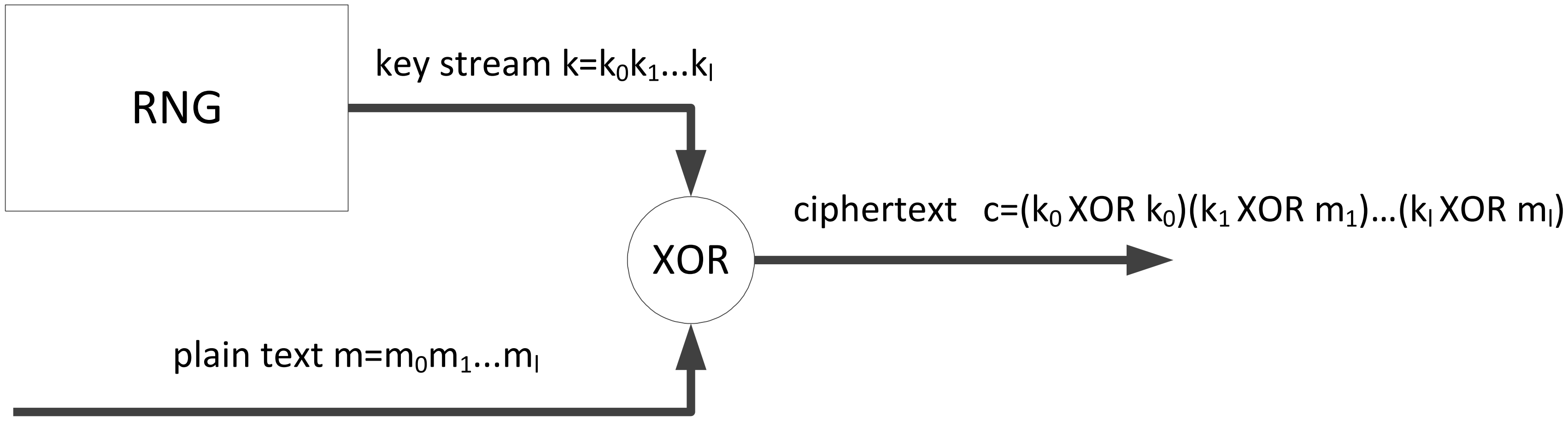
Classification: block ciphers vs. stream ciphers

- **Stream ciphers** – the message is combined via a simple transformation (e.g. XOR) with a keystream (which is a pseudorandom stream generated by a more complex mechanism), operation is done one character (bit) at a time. Examples include RC4 used in SSL/TLS or A5 used in GSM.
- **Block ciphers** – the message is transformed block by block (e.g., 128 bits) via a transformation that is depended on the key. Examples include DES, 3DES, AES.
- Remarks:
  - Block ciphers can be turned into stream ciphers in certain mode of operations, e.g., counter mode (this means that distinction between the two is not always clear)
  - Typically stream ciphers have low hardware complexity, are fast, but practical instantiations such as RC4 are not always secure



Example: the one-time pad (a stream cipher)

- Question: could you build a cipher that cannot be broken regardless of the computational power of the adversary?
- Answer: believe it or not, yes. The one-time pad is information-theoretically secure, i.e., cannot be broken regardless of computational power & ciphertext available.
- Description: generate a random key the same length as the plaintext, then simply XOR it with the plaintext



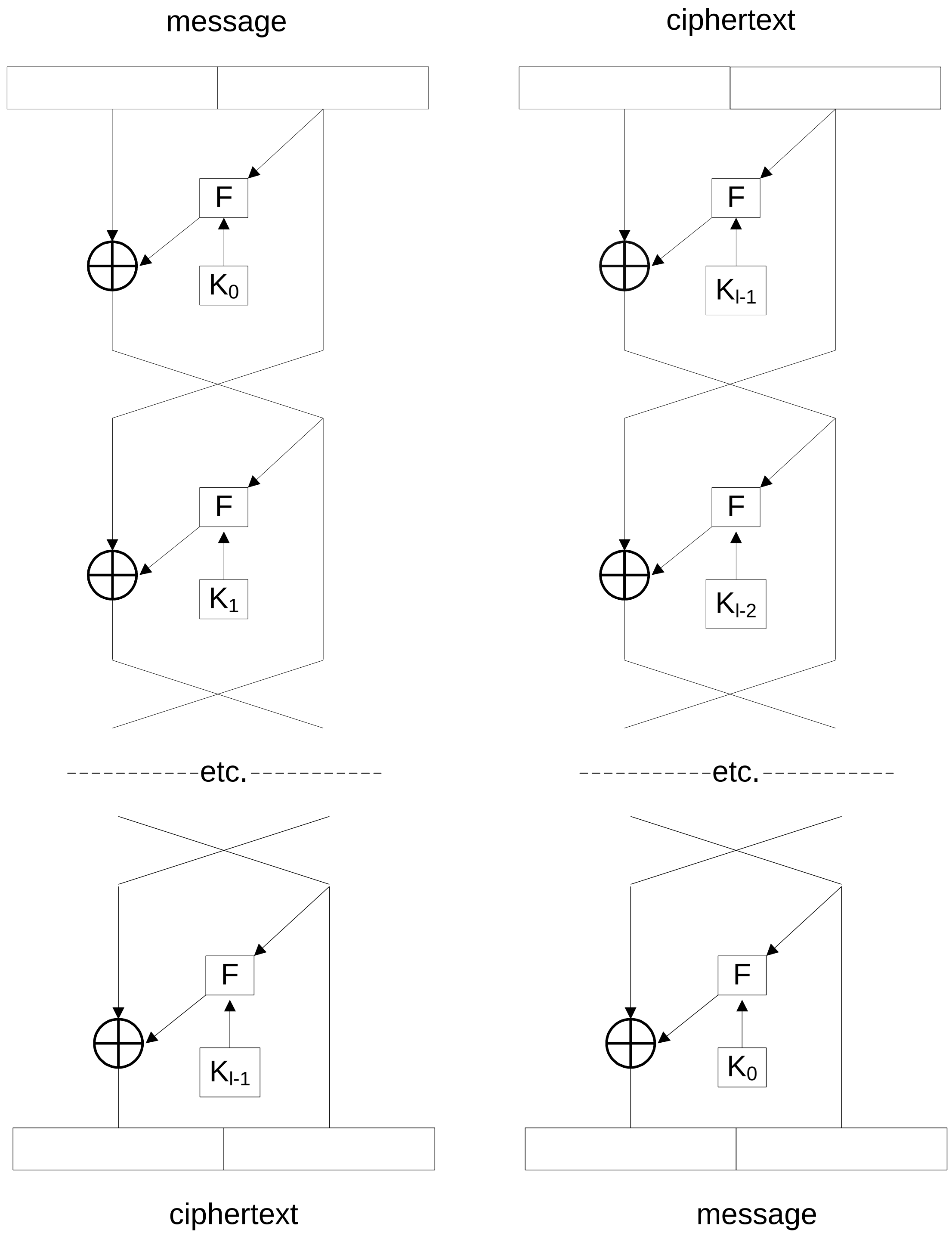
- Problems:
  - requires a random key stream the same length as the plaintext, but in practice you want a key as small as possible
  - Since it's symmetric the key needs to be exchanged a-priori on a secure channel, but then why not simply exchange the plaintext?
- Current status: there are still some practical applications where it's useful, e.g., quantum cryptography, otherwise it is not an efficient solution

Design principle: product ciphers

- Substitutions and transpositions (suggested in the work of Shannon, also used before)
  - ➢ **Substitution (S-Box)** replaces a symbol (or group of symbols) by another symbol – creates confusion
  - ➢ **Permutations (P-Box)** also known as transpositions mixes the symbols inside a block – creates diffusion
- Ciphers that use both substitutions and permutations (S-Boxes and P-boxes) are also called **product ciphers** (sometimes product ciphers denote any cipher that uses more than one transformation, while product ciphers with only S&P are called **SP-networks**)
- Remarks:
  - ➢ DES and AES, the two well known standards are product ciphers
  - ➢ Feistel ciphers are also product ciphers

Design: Feistel networks

- Designed by Horst Feistel in the ’70s at IBM
- SP-networks
- How they work:
    - Variable number of rounds
    - Each block is split into right and left part (if equal in size, then the network is called balanced)
    - Right block is passed through a round function that depends on the round key
    - Round key is derived from the master key (via the key scheduling algorithm)
    - Security/performance trade-off: increasing the number of rounds and the size of the key results in increasing security level
    - Decryption is performed by walking through the circuit in reverse order



Relevant property of the Feistel round

- Note that the Feistel round is invertible regardless of the properties of the round function, so inverting the network is straight forward as follows
  - By definition, deriving the output from the input:

$$L_i = R_{i-1}, R_i = L_{i-1} \oplus f_i(R_{i-1})$$

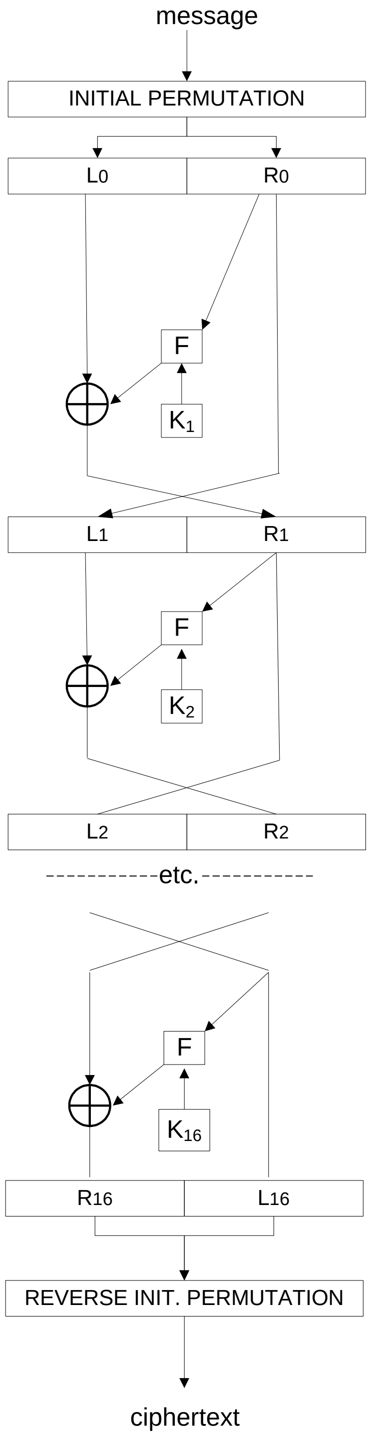
  - Which implies, deriving the input from the output

$$R_{i-1} = L_i, L_{i-1} = R_i \oplus f_i(L_i)$$

Design insights: DES

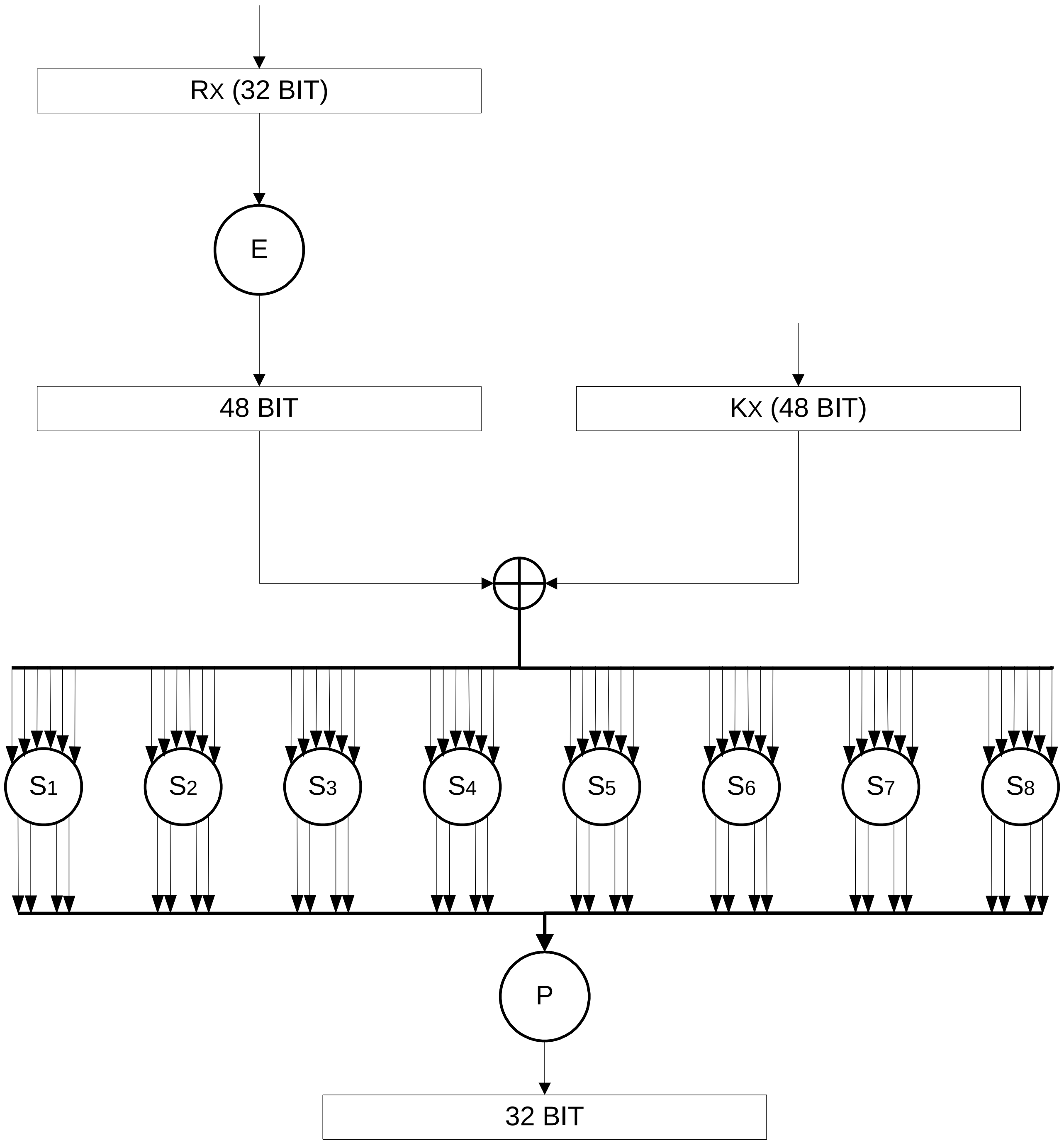
- Some DES facts:
  - ➢Developed in the 70s at IBM based on Feistel's design
  - ➢Standardized with the input from NSA
  - ➢Symmetric encryption standard between 1977-2001
  - ➢Considered insecure since the end of the 90s
  - ➢Replaced by AES (Rijndael) in 2001
  - ➢DES is a 16 round Feistel network
  - ➢DES operates on 64 bit blocks
  - ➢Surprisingly, DES key is only 56 bits

- Some DES oddities:
  - ➢DES has four weak keys: encryption and decryption have the same effect with these keys
  - ➢DES has six pairs of semi-weak keys: encryption with one key from the pair behaves as decryption with the other



DES round function

- How it works: the right half (32 bit) of the message block (64 bit) is expanded (48 bit) then XOR-ed with the round key (48 bit) and each 6 bits are provided as input to 8 x S-Boxes that output only 4 bits resulting in 32 bits that are passed through another permutation P
- This round transformation is applied 16 times, each time with a distinct round key



Examples: E, P and some S-boxes (from the standard)

$$E = \begin{pmatrix} 32 & 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 9 & 10 & 11 & 12 & 13 \\ 12 & 13 & 14 & 15 & 16 & 17 \\ 16 & 17 & 18 & 19 & 20 & 21 \\ 20 & 21 & 22 & 23 & 24 & 25 \\ 24 & 25 & 26 & 27 & 28 & 29 \\ 28 & 29 & 30 & 31 & 32 & 1 \end{pmatrix}$$

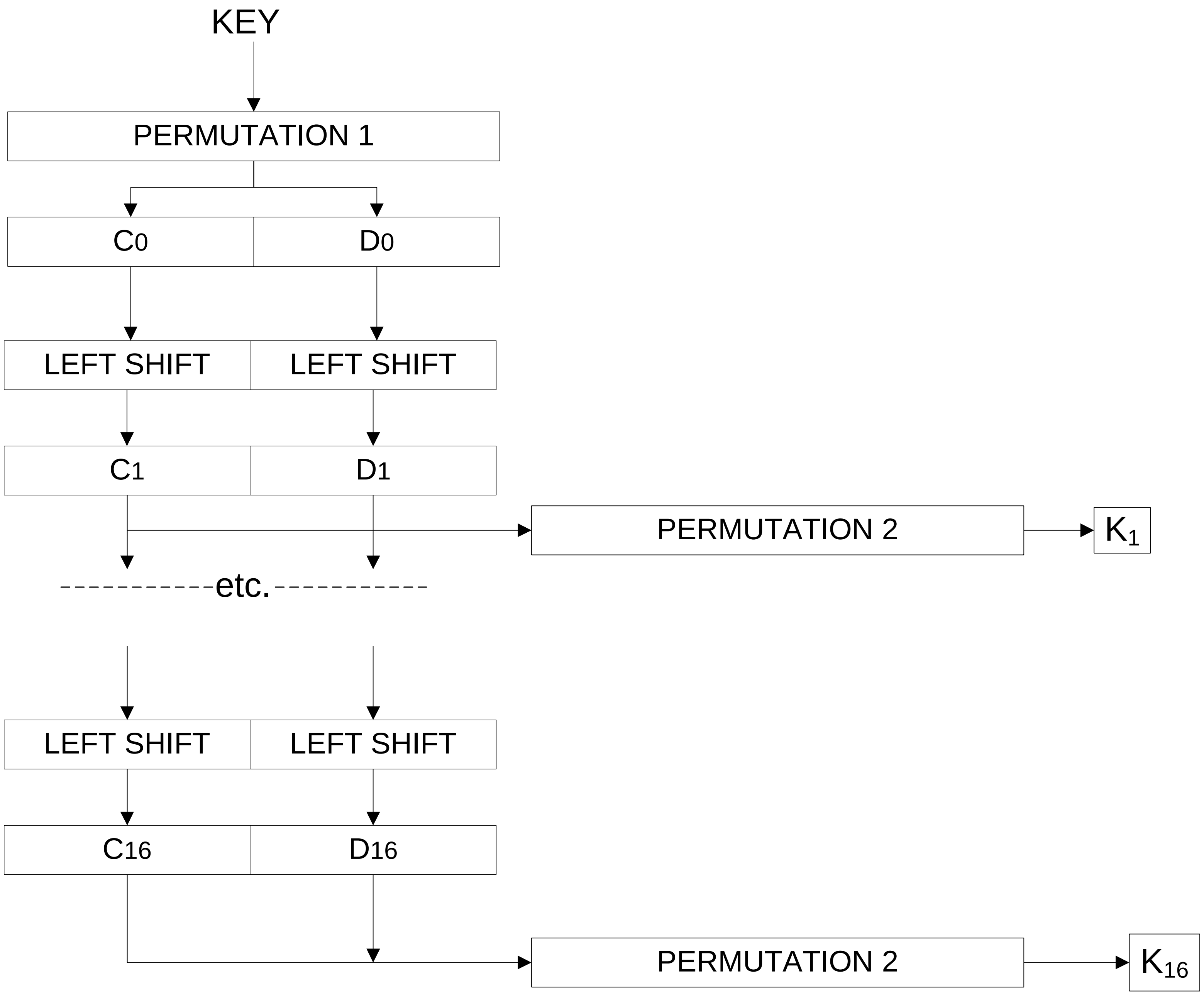
$$S_1 = \begin{pmatrix} 14 & 4 & 13 & 1 & 2 & 15 & 11 & 8 & 3 & 10 & 6 & 12 & 5 & 9 & 0 & 7 \\ 0 & 15 & 7 & 4 & 14 & 2 & 13 & 1 & 10 & 6 & 12 & 11 & 9 & 5 & 3 & 8 \\ 4 & 1 & 14 & 8 & 13 & 6 & 2 & 11 & 15 & 12 & 9 & 7 & 3 & 10 & 5 & 0 \\ 15 & 12 & 8 & 2 & 4 & 9 & 1 & 7 & 5 & 11 & 3 & 14 & 10 & 0 & 6 & 13 \end{pmatrix}$$

$$S_2 = \begin{pmatrix} 15 & 1 & 8 & 14 & 6 & 11 & 3 & 4 & 9 & 7 & 2 & 13 & 12 & 0 & 5 & 10 \\ 3 & 13 & 4 & 7 & 15 & 2 & 8 & 14 & 12 & 0 & 1 & 10 & 6 & 9 & 11 & 5 \\ 0 & 14 & 7 & 11 & 10 & 4 & 13 & 1 & 5 & 8 & 12 & 6 & 9 & 3 & 2 & 15 \\ 13 & 8 & 10 & 1 & 3 & 15 & 4 & 2 & 11 & 6 & 7 & 12 & 0 & 5 & 14 & 9 \end{pmatrix}$$

$$P = \begin{pmatrix} 16 & 7 & 20 & 21 \\ 29 & 12 & 28 & 17 \\ 1 & 15 & 23 & 26 \\ 5 & 18 & 31 & 10 \\ 2 & 8 & 24 & 14 \\ 32 & 27 & 3 & 9 \\ 19 & 13 & 30 & 6 \\ 22 & 11 & 4 & 25 \end{pmatrix}$$

DES key scheduling

- Derives each of the round keys from the master key



Designs: 3DES

- 3 DES keys K1, K2, K3 in the following transformation:

$$c = E_{K3}\left(D_{K2}\left(E_{K1}\left(m\right)\right)\right), \qquad m = D_{K1}\left(E_{K2}\left(D_{K3}\left(c\right)\right)\right)$$

- Considered to be secure so far (given that all three keys are random and independent) but it is slower than AES (thus no serious reasons for use in practice)
- Has 3 keying options: i. independent keys, ii. K1 and K2 independent but K3=K1, iii. all keys are equal K1=K2=K3 (this is DES)
- Main reason for practical persistence may be the electronic payment industry

Designs: AES

- AES facts:
  - Designed by Vincent Rijmen and Joan Daemen
  - Selected by public competition from the 5 finalists: MARS, RC6, **Rijndael**, Serpent, and Twofish
  - The new standard as of 2001
  - Not a Feistel network
  - Available with 3 key lengths: 128, 192, 256 bits
- How AES works
  - Operates on a 4x4 matrix of bytes (128 bit blocks) called state
  - Has 10, 12 or 14 rounds according to the key size
  - Each round has 4 transformations: SubBytes (a substitution) is non-linear substitution where each byte is replaced via a look-up table, ShiftRows (a permutation) the last three rows are shifted, MixColumns the four bytes of each column are combined via a linear transformation, AddRoundKey each byte of the state is combined with the round key via a XOR operation

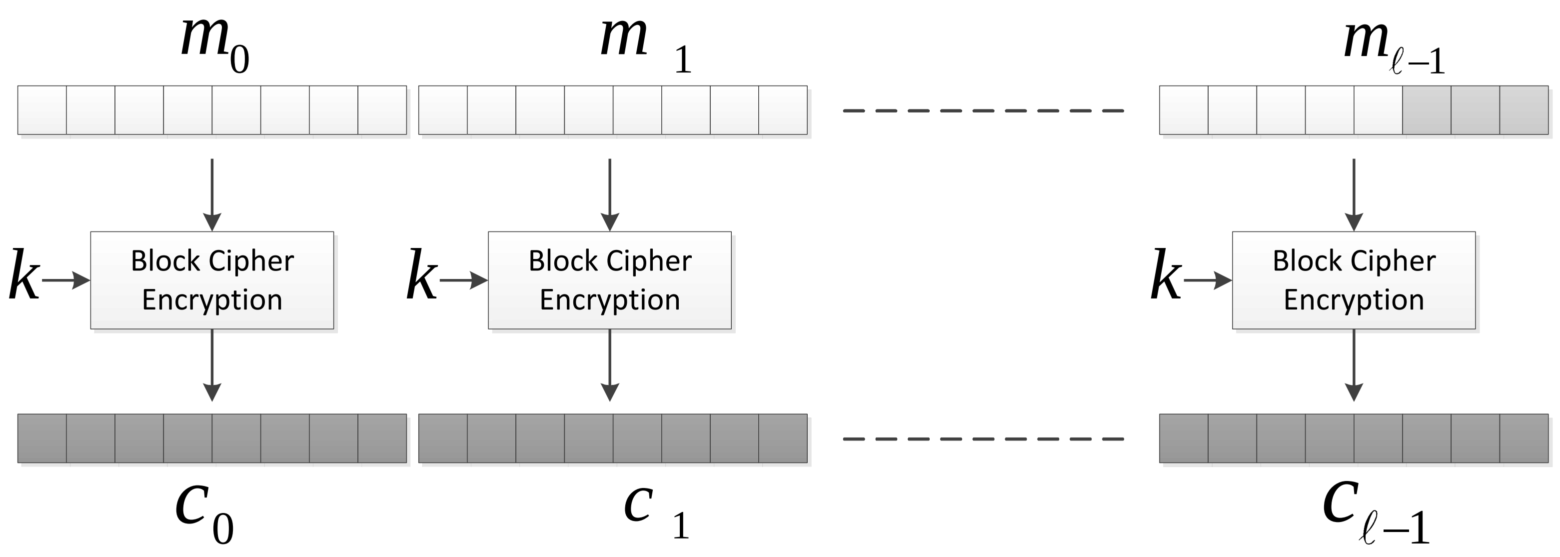
AES_Encrypt_Round(State, Key)
{
SubBytes(State) ;
ShiftRows(State);
MixColumns(State);
AddRoundKey(State, Key);
}
	AES_Decrypt_Round(State, Key)
	{
	AddRoundKey^{-1}(State, Key);
	MixColumns^{-1} (State);
	ShiftRows^{-1} (State);
	SubBytes^{-1} (State) ;
	}

Block Ciphers use in practice

- Question: block ciphers work on single blocks of message, how do you extend them to multiple blocks?

Electronic Code Book (ECB)

- The message is parsed into blocks and each block is encrypted with the secret key
- Decryption is done by reversing this operation

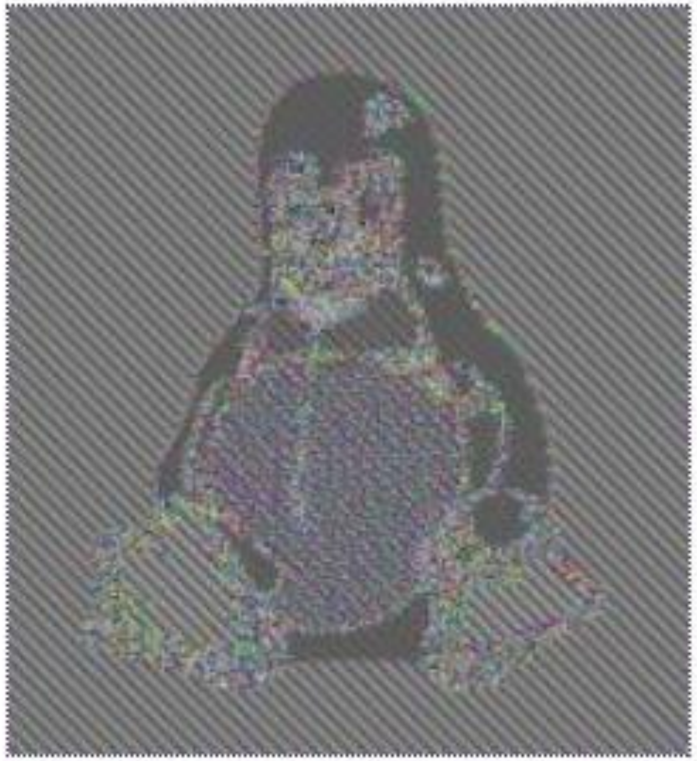


- Question: assuming that the block cipher is secure, is this construction secure?

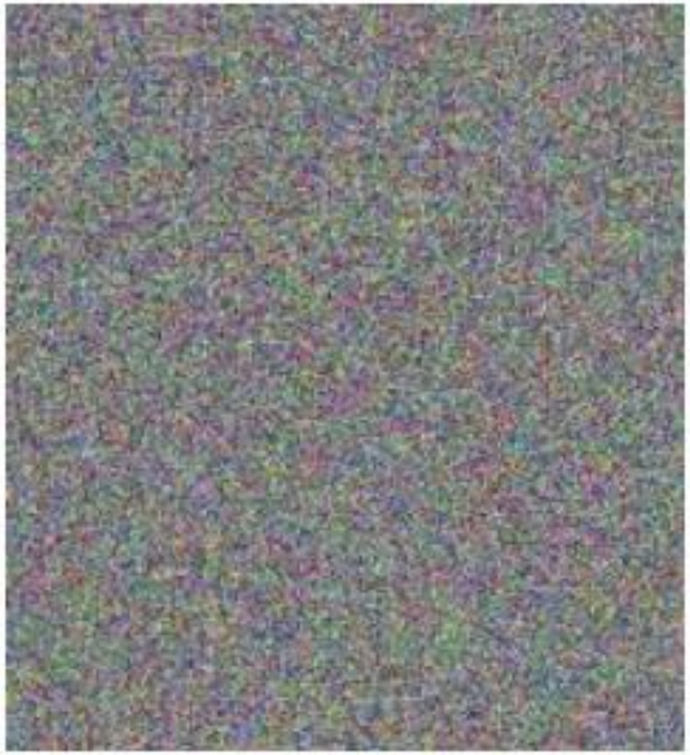
• Answer: No. Do not use ECB.

© image from wikipedia.org

Original



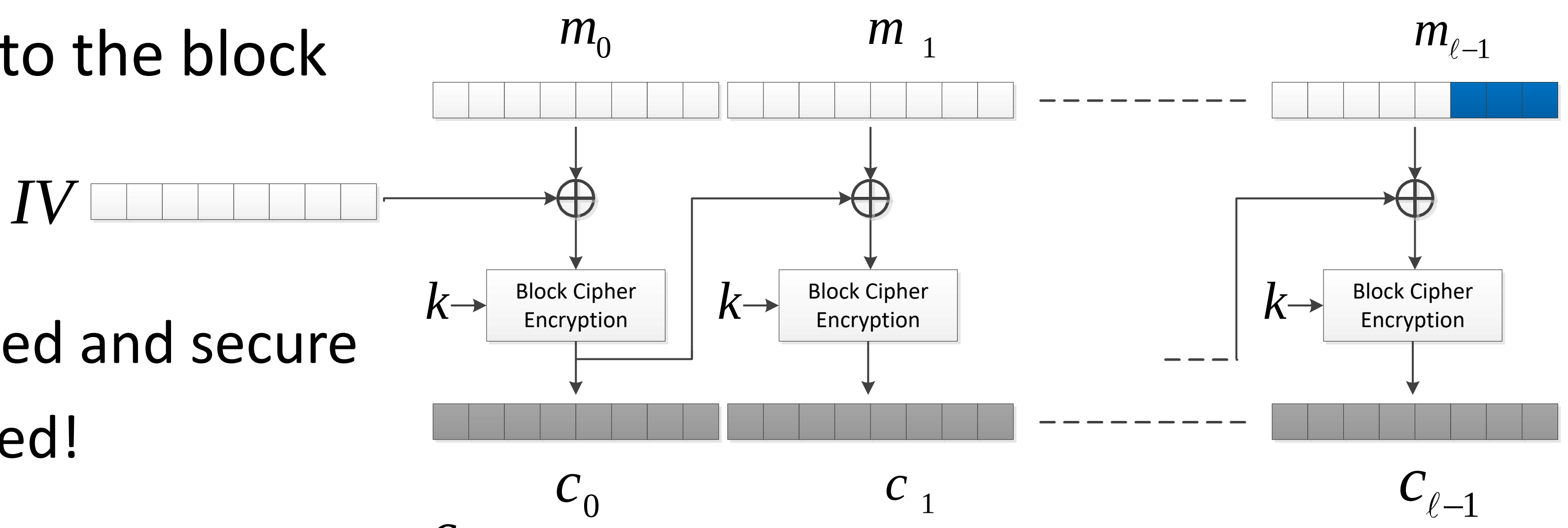
Encrypted using ECB mode



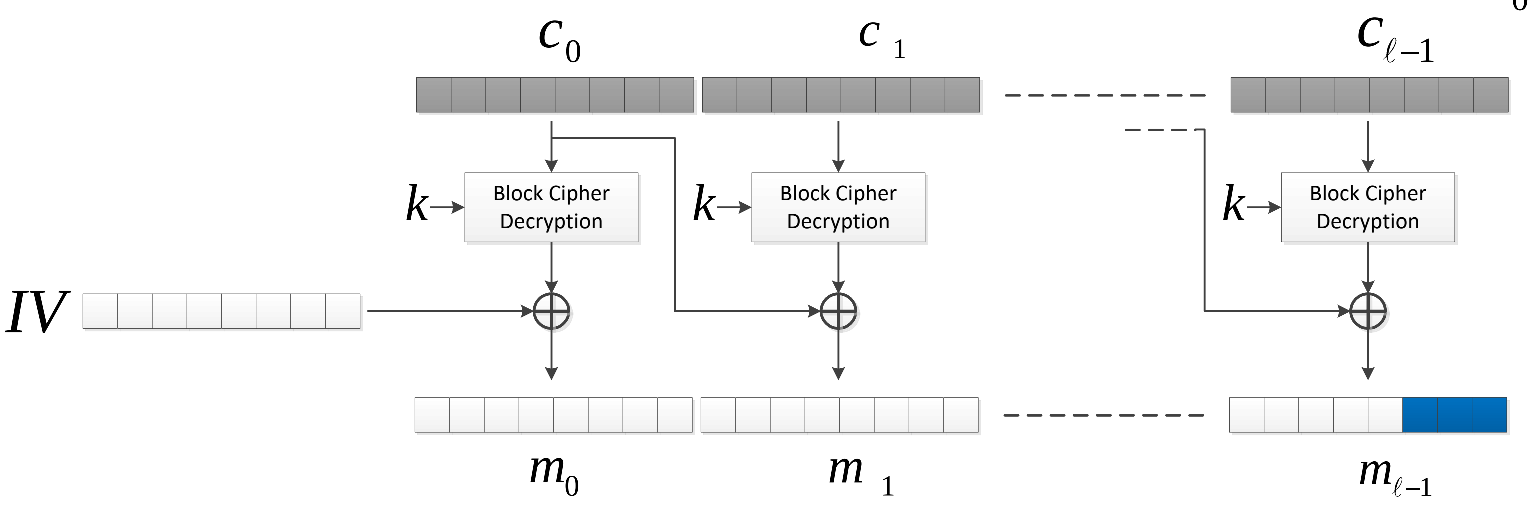
Other modes than ECB results in pseudo-randomness

Cipher Block Chaining (CBC)

- Initialization Vector (IV) is a non-secret random value used for randomization of the first output block
- Last message chunk is padded to the block length



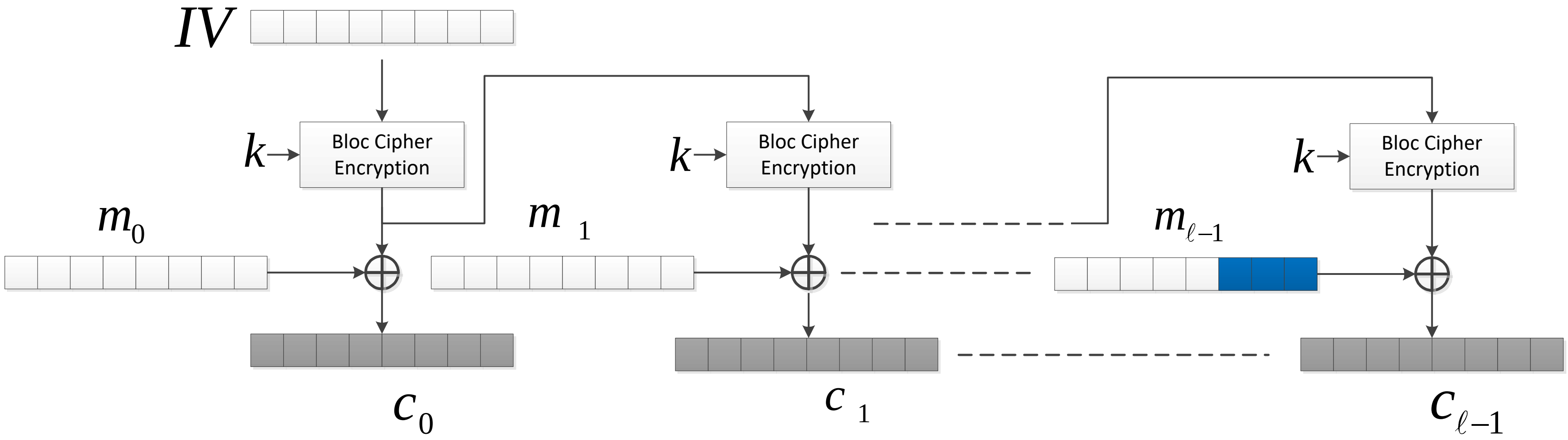
- Pros: encryption is fully randomized and secure
- Pros: decryption can be parallelized!

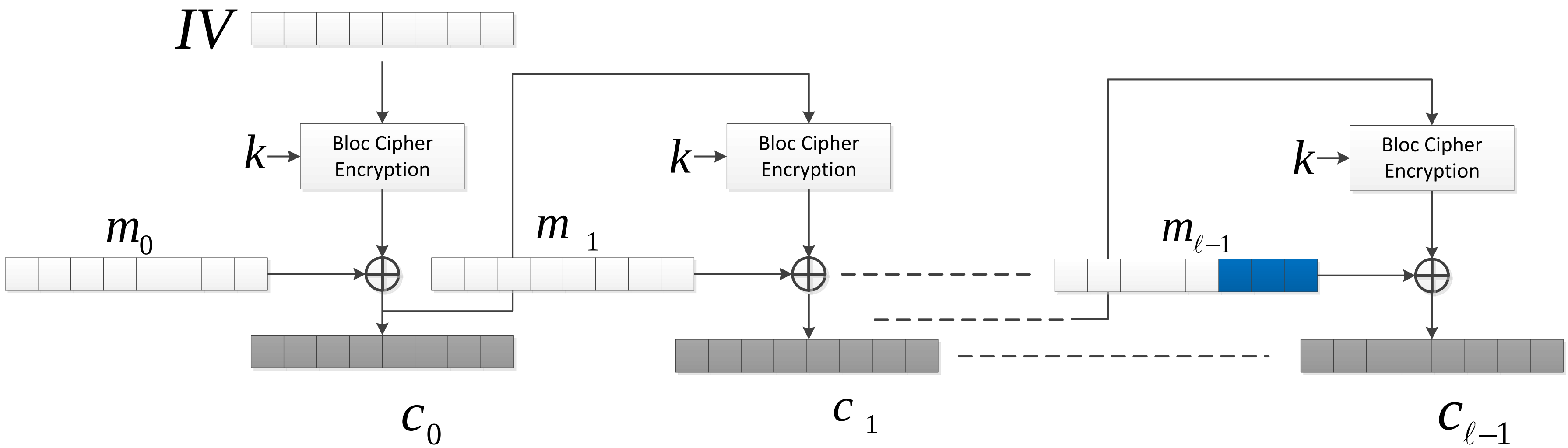


- Cons: if one of the blocks is lost, decryption cannot be performed on the next
- Question: if one cipher text block has a 1 bit error and all other blocks are fine, how much do you lose when decrypting?

Some variations: Output-feedback (OFB) and Cipher Feedback (CFB)

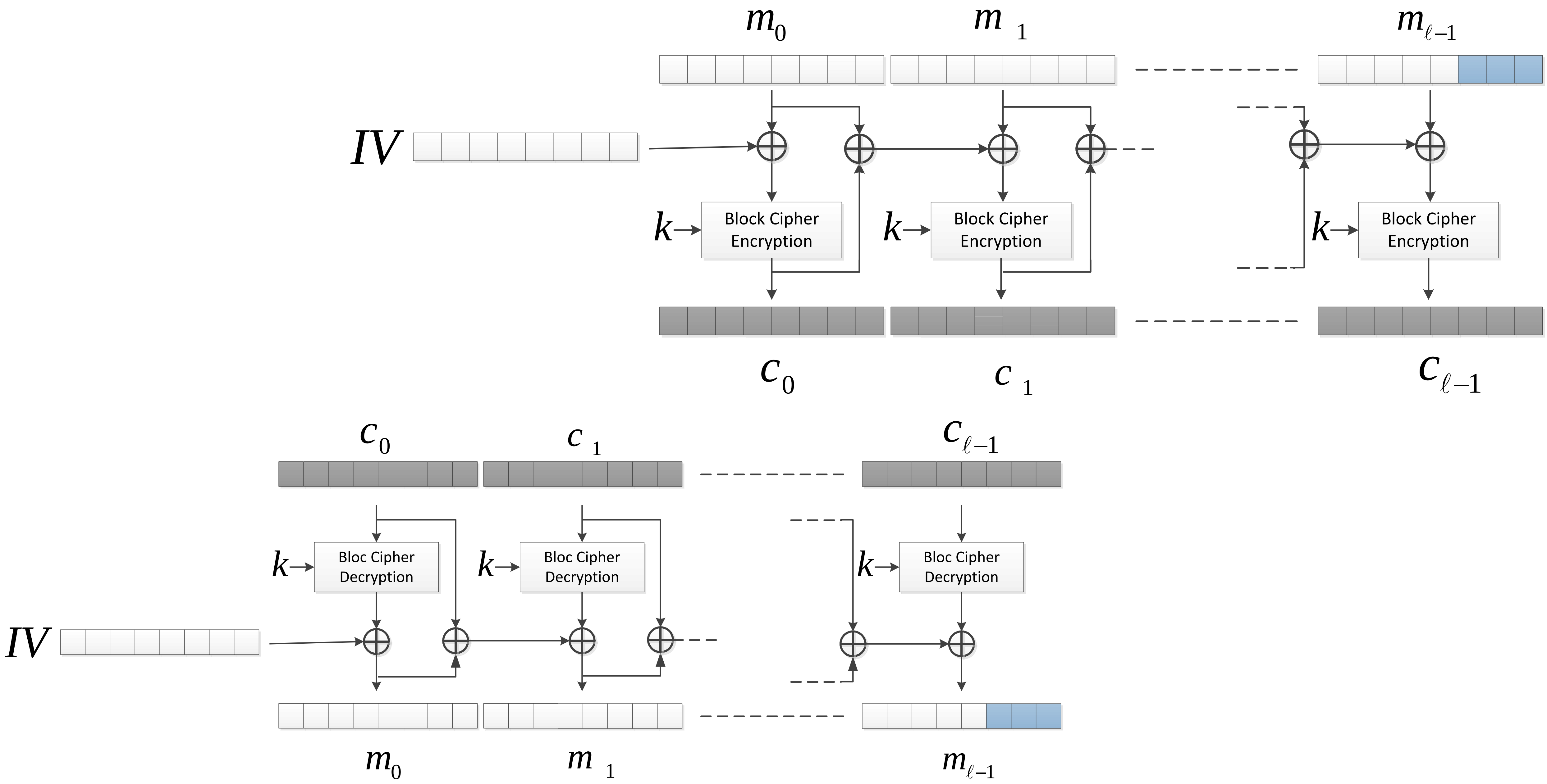
- Pros: OFB allows decryption even when message blocks are lost, it also allows pre-computation of the key stream
- Pros: CFB and OFB are stream modes and don’t require padding





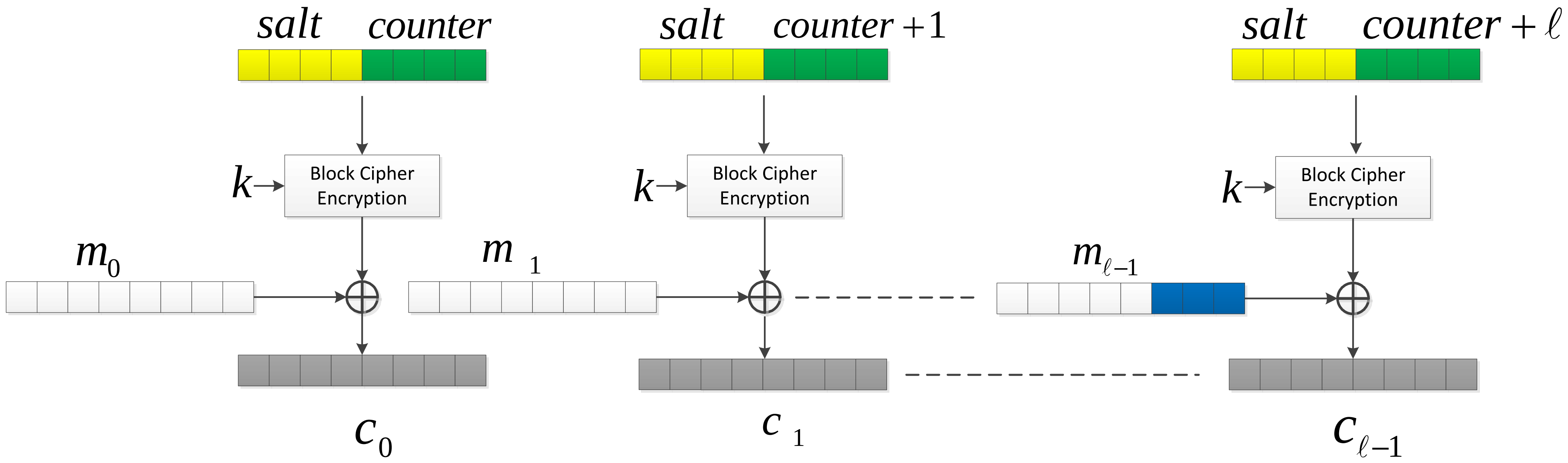
- Pros: CFB allows decryption to be parallelized

Another variation: Propagating Cipher Block Chaining (PCBC)



Counter Mode

- A counter is incremented and encrypted for each block, then XORed with the message
- Pros: decryption can still be performed if bocks are lost, key-stream can be pre-computed
- This mainly converts the block cipher into a stream cipher



Additional notes on padding

- To make the encryption plaintext an integer multiple of the block size some padding is needed
- According to PKCS#7, the most commonly used padding for symmetric encryptions, padding is in whole bytes and the value of each added byte is the number of bytes that are added, e.g.
  - 01 – for a message of 15 bytes (assuming 128 bit blocks)
  - 02 02 – for a message of 14 bytes (assuming 128 bit blocks)
  - 03 03 03 – for a message of 13 bytes (assuming 128 bit blocks)
  - etc.
- Question: what is the padding value when the plaintext is already a multiple of the block size?

Adversary capabilities (informal) – what the adversary can do?

- **CPA** – chosen plaintext adversary, an adversary that has access to a black-box that encrypts plaintexts at the adversary choice
- **CCA** – chosen ciphertext adversary, an adversary that has access to a black-box that decrypts cyphertexts at the adversary choice

- **Adaptive vs. non-adaptive** – is an additional flavour that can be added to both CPA and CCA meaning that the adversary can continue (adaptive) or not (non-adaptive) to query the encryption/decryption box after he received the target ciphertext that he is required to break (obviously the adversary is not allowed to query the target ciphertext to the decryption box)

Security notions (informal)

- semantic security (SS) (Goldwasser & Micali 1982)

Any information that can be efficiently computed with the ciphertext, can be also computed without the ciphertext

- indistinguishability of ciphertexts (IND)

Given two messages selected by the adversary and the encryption of one of them chosen at random (without adversary's knowledge) the adversary cannot decide which is the encrypted message

- real or random indistinguishability (RoR)

Given a message selected by the adversary and the encryption of either this message or some complete random message (not known to the adversary) the adversary cannot decide if the ciphertext corresponds or not to its chosen plaintext

- Question: which of the previous properties is the strongest?
- Answer: under proper formalization they are all equivalent, see Goldreich – Foundations of Cryptography, vol II, p.383
- Question: which is easier to prove?
- Answer: generally IND or RoR are easier to prove and are the standard tool in proving security

How to prove equivalences?

- Security reductions, proving that a cryptosystem that has one property has the other (or the reverse, if it doesn't have one property it doesn't have the other)

Example, security reductions: IND → RoR & IND ←RoR

- Proof to be done as exercise during laboratory hours

Type of functions (II) Hash functions

- Description: an algorithm that takes as input a message of any length and turns it into a constant size output (usually referred as tag or simply hash)

i.e., $tag = H(m)$

- Example of use: assure integrity of software downloads/updates, protect stored passwords, etc.

e.g., downloading images from ubuntu.com

14.10

(Utopic Unicorn): October 2014 (Supported until July 2015)

md5 Hash	Version
08494b448aa5b1de963731c21344f803	ubuntu-14.10-desktop-amd64.iso
4a3c4b8421af51c29c84fb6f4b3fe109	ubuntu-14.10-desktop-i386.iso
91bd1cfba65417bfa04567e4f64b5c55	ubuntu-14.10-server-amd64.iso

- Standards:
  - Not to use MD5, SHA1 (not resistant to collisions)
  - To use SHA2 (mostly 256, 384 and 512 are somewhat slow)
  - Future use: SHA3 (Keccak the winner of the competition)
  - Alternatives: BLAKE is a lightweight design, one of the SHA3 finalists

Security properties for hash functions

- The following properties are mandatory for hash functions:
  - **Pre-image resistance** – given the hash of some message it is infeasible to find the message

    i.e., $h(m) \xrightarrow{?} m$

  - **Secondary pre-image resistance** – given the hash of a message and the message it is infeasible to find a second message that has the same hash value

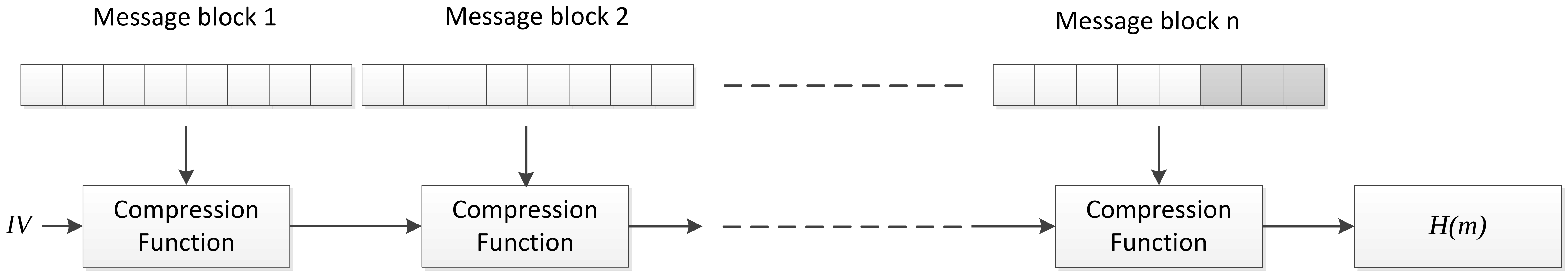
    i.e., $m_1, h(m_1) \xrightarrow{?} m_2 \ s.t. \ h(m_1) = h(m_2)$

  - **Collision resistance** – it is infeasible to find two messages that have the same hash

    i.e., $\xrightarrow{?} m_1, m_2 \ s.t. \ h(m_1) = h(m_2)$

Design principle

- The Merkle-Damgard construction provides a method for turning a collision-resistant one-way functions into a collision-resistant hash functions
- This design stands behind MD5, SHA1 and SHA2
- The IV is fixed (not random like in block ciphers modes of operation)



Design insights: MD5

- 4 IV's defined as follows

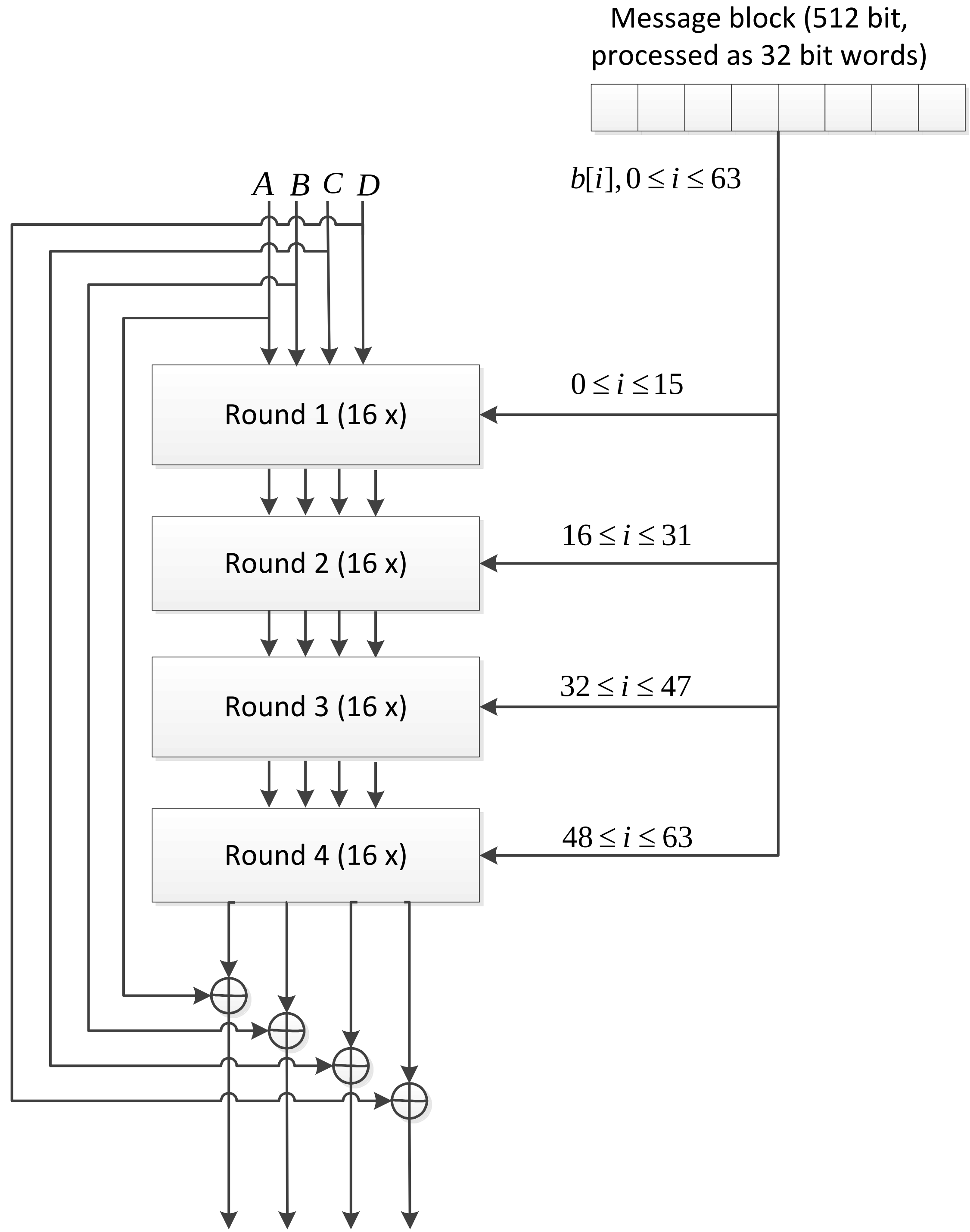
A= 0x67452301,

B= 0xefcdab89,

C= 0x98badcfe,

D= 0x10325476.

- Message is processed in blocks of 512 bits that are further split in 128 bit chuncks and propagated as IVs for the next block to be hashed (i.e., Merkle-Damgard construction)



MD5 round function

- Each round proceeds with the following transformation (A, B, C, and D are the IV's, K and S are fixed constants and M is the message):

$$D \leftarrow C,$$
$$C \leftarrow B,$$
$$B \leftarrow B + ((A + FR(B, C, D) + M + K) <<< S),$$
$$A \leftarrow D.$$

- Round function is distinct for each round (still, all round functions consist in simple logic operations AND, OR, XOR and NOT):

$$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z),$$
$$G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z),$$
$$H(X, Y, Z) = X \oplus Y \oplus Z,$$
$$I(X, Y, Z) = Y \oplus (X \vee \neg Z).$$

Padding

- According to RFC 6234 one byte \x80 followed by as many 0s as needed, followed by the length in bits of the message as a 64 bit integer
- Example: “Hello world\x80\x00\x00\x00\x00 ….. \x00\x00\x00\x00\x0B”

Test vectors as per RFC 1321

- Examples of what you get after you hash

```
MD5 ("")            = d41d8cd98f00b204e9800998ecf8427e
MD5 ("a")           = 0cc175b9c0f1b6a831c399e269772661
MD5 ("abc")         = 900150983cd24fb0d6963f7d28e17f72
MD5 ("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5 ("abcdefghijklmnopqrstuvwxyz")=c3fcd3d76192e4007dfb496cca67e13b
MD5 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789")
    = d174ab98d277d9f5a5611c2c9f419d9f
MD5("1234567890123456789012345678901234567890123456...2345678
90") = 57edf4a22be3c955ac49da2e2107b67a
```

Type of functions (I) Keyed Hash Functions (or MACs)

- Description (informal): an algorithm that takes a message of arbitrary length and and a key then outputs a tag

i.e., $tag = MAC_k(m)$

- Example of use: assuring message authentication, i.e., binding a message with the identity of a principal that knows a key
- Standards:
  - ➢Not to use: simple concatenation of key to a message is in general insecure
  - ➢To use: HMAC or NMAC with one of the previous hash functions
  - ➢Future use: N/A

Message Authentication Codes formal definition

- A message authentication code is a **triple of algorithms**:
  - ***Gen*** *is the key generation algorithm that takes random coins, a security parameter l and outputs the key* $k \leftarrow Gen(1^l)$
  - ***Mac*** *is the tag-generation algorithm that takes as input the key and some message, then outputs the tag* $tag \leftarrow MAC(k, m)$
  - ***Ver*** *is the verification algorithm that takes as input the key, the tag and the message and outputs 1 if the tag is valid or 0 otherwise* $\{0,1\} \leftarrow Ver(k, tag, m)$

- A correctness condition enforces that $Ver(k, MAC(k, m), m) = 1$

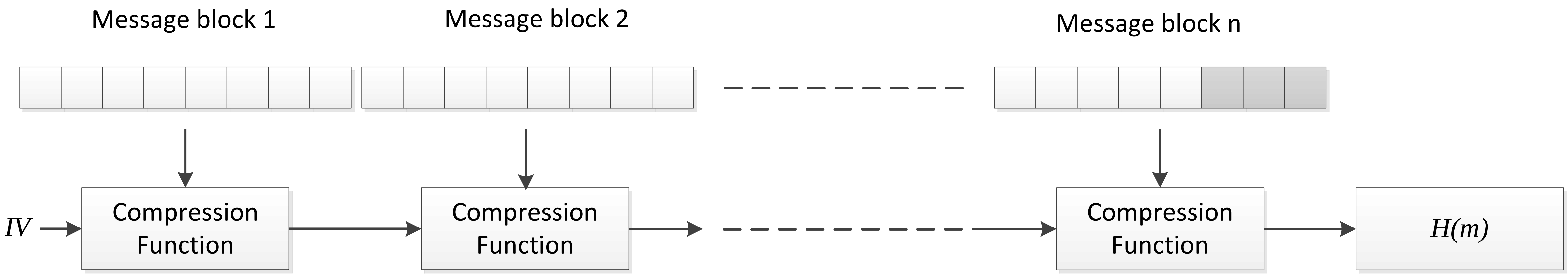
Desired Properties for MACs

- Fortunately, there is only one strong definition of security (of course, this can be refined in several ways)
- MACs must have **(existential) unforgeability under chosen message attacks**, that is, an adversary that receives any number of valid message-tag pairs (i.e., pairs that are computed with the MAC algorithm) is unable to output a new message-tag pair that will successfully pass through the verification algorithm

What not to use

- Question: based on the previous security definition for MAC code, is the simple concatenation of message to key, i.e., H(k||m), secure?

- Answer: No. Concatenation attacks are possible due to the construction of some hash functions (revisit MD5 and the Merkle-Damgard construction)



HMAC

- Simple and secure
- The application of a hash function twice with an inner-padding (ipad) and outer-padding (opad)
- ipad is B blocks of 0x36 and opad is B blocks of 0x5C, where B is the byte size of the block to be processed (e.g., B=64 in case of MD5 that uses blocks of 512bits)

$$HMAC(K,m) = H((K \oplus \mathrm{opad}) \,||\, H((K \oplus \mathrm{ipad}) \,||\, m))$$

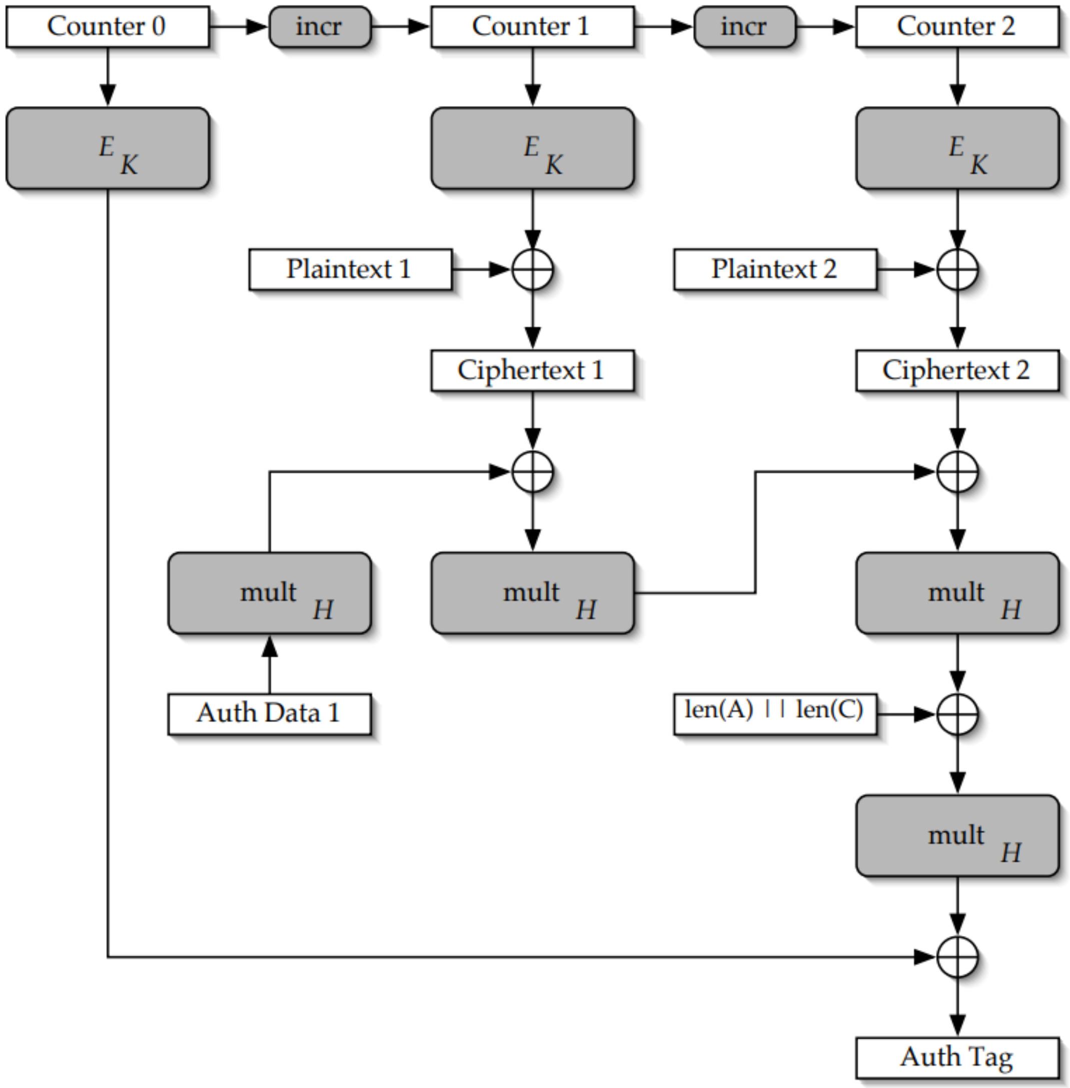
- Can be paired with any hash function, e.g., HMAC-MD5, HMAC-SHA256, etc.
- NMAC (Nested MAC) is as simple as HMAC, however it requires changing the IV which is less handy when implementing

Various paradigms of combining MACs with encryptions

- A frequent application of MAC functions is in **authenticated encryption**, i.e., assuring that an encrypted ciphertext indeed originates from the source (note that block ciphers are not designed for this)
- There are three paradigms employed in practice:
  - **Encrypt-and-MAC**, i.e., $E_k(m)||MAC_k(m)$, used in SSH
  - **MAC-then-encrypt** , i.e., $E_k\big(m||MAC_k(m)\big)$, used in SSL/TLS
  - **Encrypt-then-MAC**, i.e., $E_k(m)||MAC_k\big(E_k(m)\big)$, used in IPSec
- **Encrypt-then-MAC** has better security than the previous two and should be the desired alternative in practice
- For details, see Bellare & Namprempre, "Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm", 2000

Widely-adopted: Galois Counter Mode (GCM)

- Key points:
  - Auth Data 1 represents additional authenticated data (not encrypted), e.g., headers, etc.
  - Requires multiplication in a Galois Field GF(2^{128}) using field polynomial $f = 1 + \alpha + \alpha^2 + \alpha^7 + \alpha^{128}$
  - See NIST specification for additional info



https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf

Type of functions (IV) RNGs and PRNGs

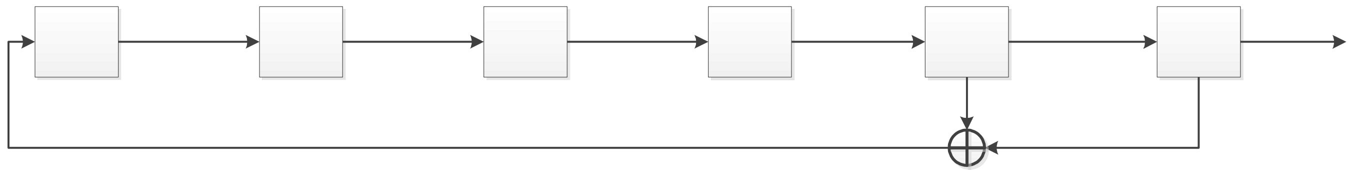
- Random numbers stay at the core of any cryptosystem since you need randomness for the secret keys
- Description (informal):
  - **TRNG** – True random-number generators output random sequences based on physical processes that are hard/infeasible to model, i.e., white noise from a Zenner diode, oscillator drift, SRAM state at power-up, etc.
  - **PRNGs** – deterministic algorithms that generate a random sequence based on a value called seed (they all have cycles but this does not mean they are insecure, computationally secure PRNGs exist)
- Example of use: used in any handshake SSL/TLS, IPSec, etc. that needs to generate a fresh session key

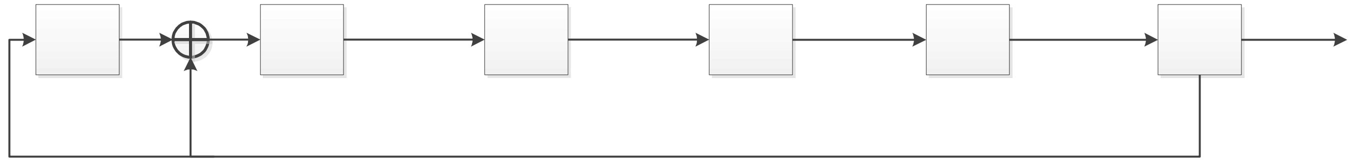
PRNG examples

- The linear congruential generator, an insecure and yet common solution

$$X_{i+1} = aX_i + c \bmod n \quad (X_0 \text{ is the seed})$$

- Galois or Fibonacci LFSR (Linear Feedback Shift Register) are another common, insecure alternative





- Bloom-Bloom-Shub is cryptographically secure but requires a large modulus n and is computationally expensive, thus almost absent in practice (X_0 is the seed)

$$X_i = X_{i-1}^2 \bmod n \quad (X_0 \text{ is the seed})$$

- Block ciphers in counter mode or stream ciphers provide secure instantation of PRNGs (as long as the cipher is secure)

Testing for randomness

- Various statistical tests are usually employed, none is perfect but may provide some degree of confidence, for example:
  - Entropy, e.g., ideally a random string will have 8 bits/byte

$$H(X) = -\sum_{x \in X} p(x)\log(p(x))$$

  - Compression rate, e.g., a random string will have 0% compression rate
  - Chi square distribution – the difference between the expected and observed frequency of occurrence

$$\chi^2 = \sum \frac{(o_i - e_i)^2}{e_i}$$

  - Arithmetic mean, e.g., random bytes will have the arithmetic mean at 127.5
  - Monte Carlo Pi – estimates the value of π by considering a quadrant inscribed in a unit square (the number of points inside the circle divided by all the number of attempts converges to π/4)
  - Serial correlation coefficient – totally uncorrelated data will lead to a value of 0

Test suites for randomness

- Dieharder is a battery of tests used by many enthusiasts or professionals http://www.phy.duke.edu/~rgb/General/rand_rate.php

Questions?

Asymmetric Primitives

(public key encryptions and digital signatures)

Real World RSA Keys

- 2048 bit RSA key from RSA factoring challenge (offered 200.000$ for its factors)

25195908475657893494027183240048398571429282126204032027777137836043662020707595556264018525880784406918290641249515082189298559149176184502808489120072844992687392807287776735971418347270261896375014971824691165077613379859095700097330459748808428401797429100642458691817195118746121515172654632282216869987549182422433637259085141865462043576798423387184774447920739934236584823824281198163815010674810451660377306056201619676256133844143603833904414952634432190114657544454178424020924616515723350778707749817125772467962926386356373289912154831438167899885040445364023527381951378636564391212010397122822120720357

- Question: Consider to factor by exhaustive search? What is the expected number of steps?

- Need more motivation? The following rewards were withdrawn by RSA, but still …

RSA-768	**$50,000 USD**	**(factored December 12, 2009)**
RSA-896	**$75,000 USD**	
RSA-1024	**$100,000 USD**	
RSA-1536	**$150,000 USD**	
RSA-2048	**$200,000 USD**	

Real World RSA Keys

- 2048 bit RSA key from RSA factoring challenge (offered 200.000$ for its factors)

  25195908475657893494027183240048398571429282126204032027777137836043662020707595556264018525880784406918290641249515082189298559149176184502808489120072844992687392807287776735971418347270261896375014971824691165077613379859095700097330459748808428401797429100642458691817195118746121515172654632282216869987549182422433637259085141865462043576798423387184774447920739934236584823824281198163815010674810451660377306056201619676256133844143603833904414952634432190114657544454178424020924616515723350778707749817125772467962926386356373289912154831438167899885040445364023527381951378636564391212010397122822120720357

- Question: Consider to factor by exhaustive search? What is the expected number of steps?

- You should take a look at the following:
  - *electrons in universe:*
    $8{,}37*10^{77} \approx 837000000000000000000000000000000000000000000000000000000000000000000000000000$
  - age of solar system: $1{,}89*10^{17} \approx 189000000000000000$

- Need more motivation? The following rewards were withdrawn by RSA, but still …

RSA-768	**$50,000 USD**	**(factored December 12, 2009)**
RSA-896	**$75,000 USD**	
RSA-1024	**$100,000 USD**	
RSA-1536	**$150,000 USD**	
RSA-2048	**$200,000 USD**	

An informal, yet instructive account of asymmetric primitives …

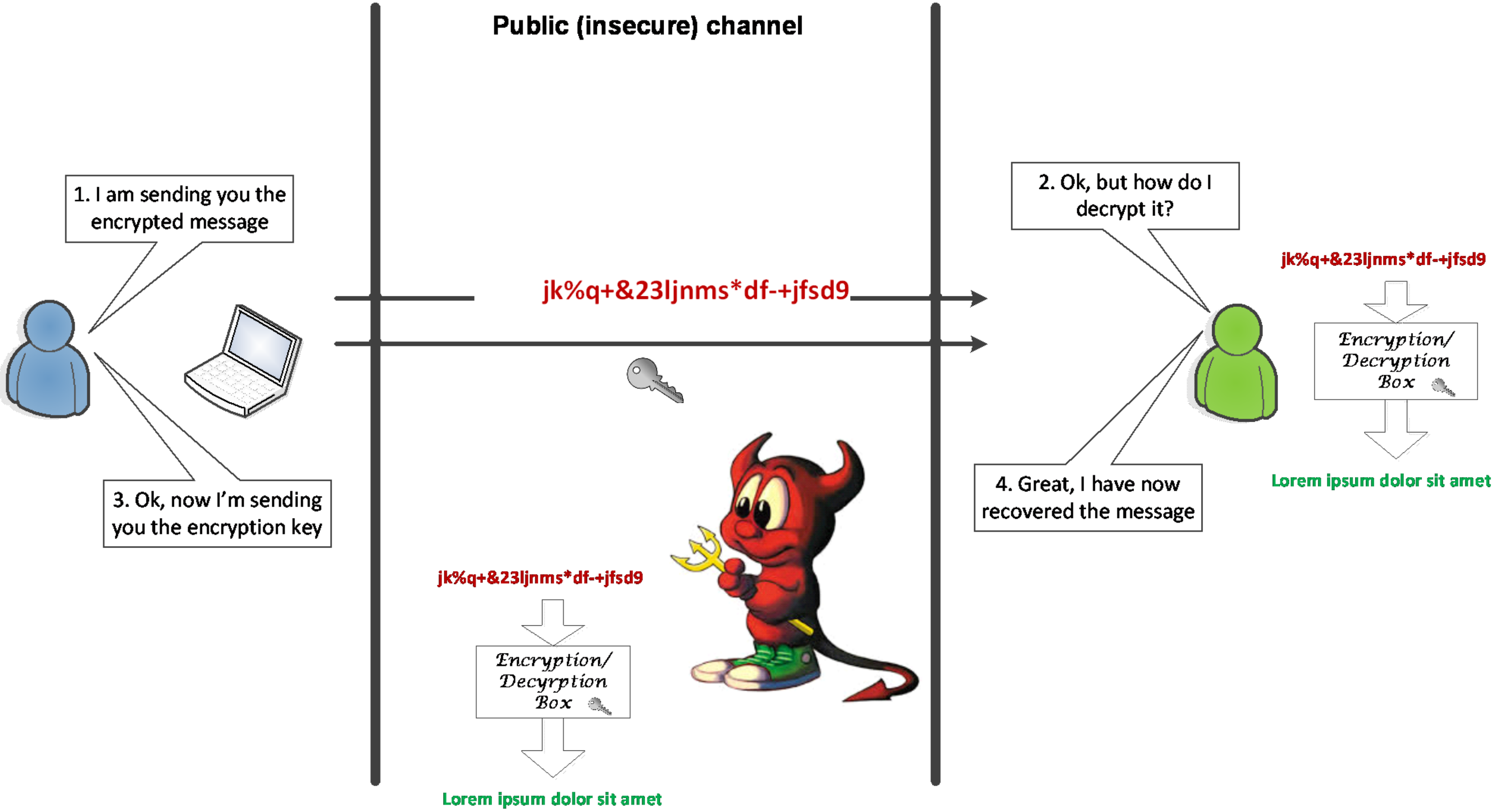
Timeline of the invention of public-key cryptography

- *1970-1974 British cryptographers James Ellis and Clifford Cocks from GCHQ invent the possibility of non-secret key encryption and the RSA*
- 1974 Ralph Merkle invented a public-key agreement that was published only in 1978
- 1976 Withfield Diffie and Martin Hellman, influenced by Ralph Merkle's work, published a method for public-key agreement (known as Diffie-Hellman key exchange, or Diffie-Hellman-Merkle key exchange)
- 1977 Ron Rivest, Adi Shamir and Leonard Adleman invent the RSA, published in 1978
- 1979 Michael O. Rabin publishes the Rabin cryptosystem, a public key cryptosystem with security equivalent to factoring
- 1985 Taher ElGamal published a method for encrypting and signing based on DHM key exchange
- 1985 Neal Koblitz and Victor Miller independently and simultaneously introduce elliptic curve cryptography

Why we need public-key cryptography?

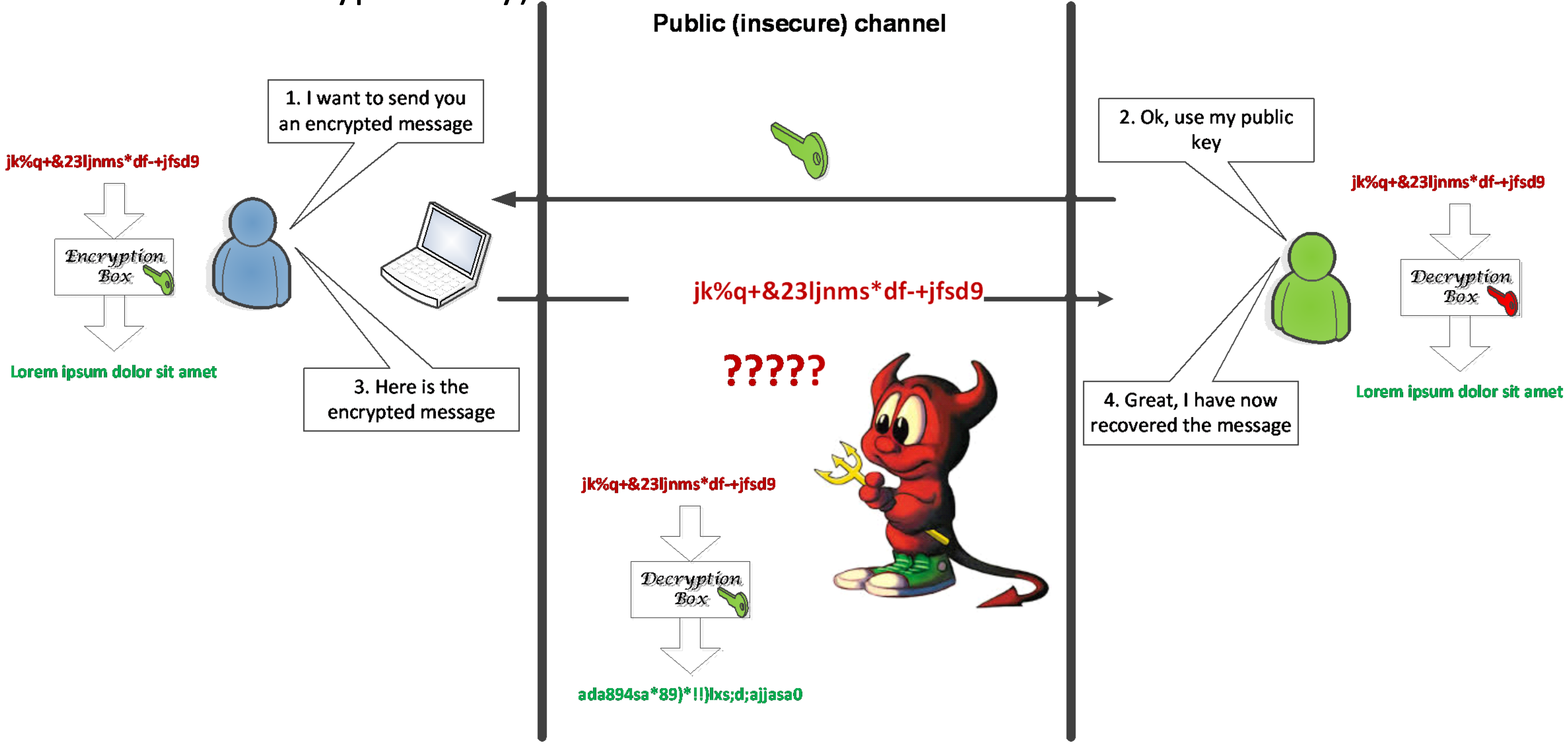
- Answer: Exchanging information securely over an insecure channel in the absence of a secretly shared key

- All symmetric key cryptosystems require a key to be shared between parties
- But in the real-world communication happens spontaneously between parties that did not interact before (i.e., previously shared secrets do not exist) and exchanging a secret key securely over a public channel (e.g., Internet) is not possible



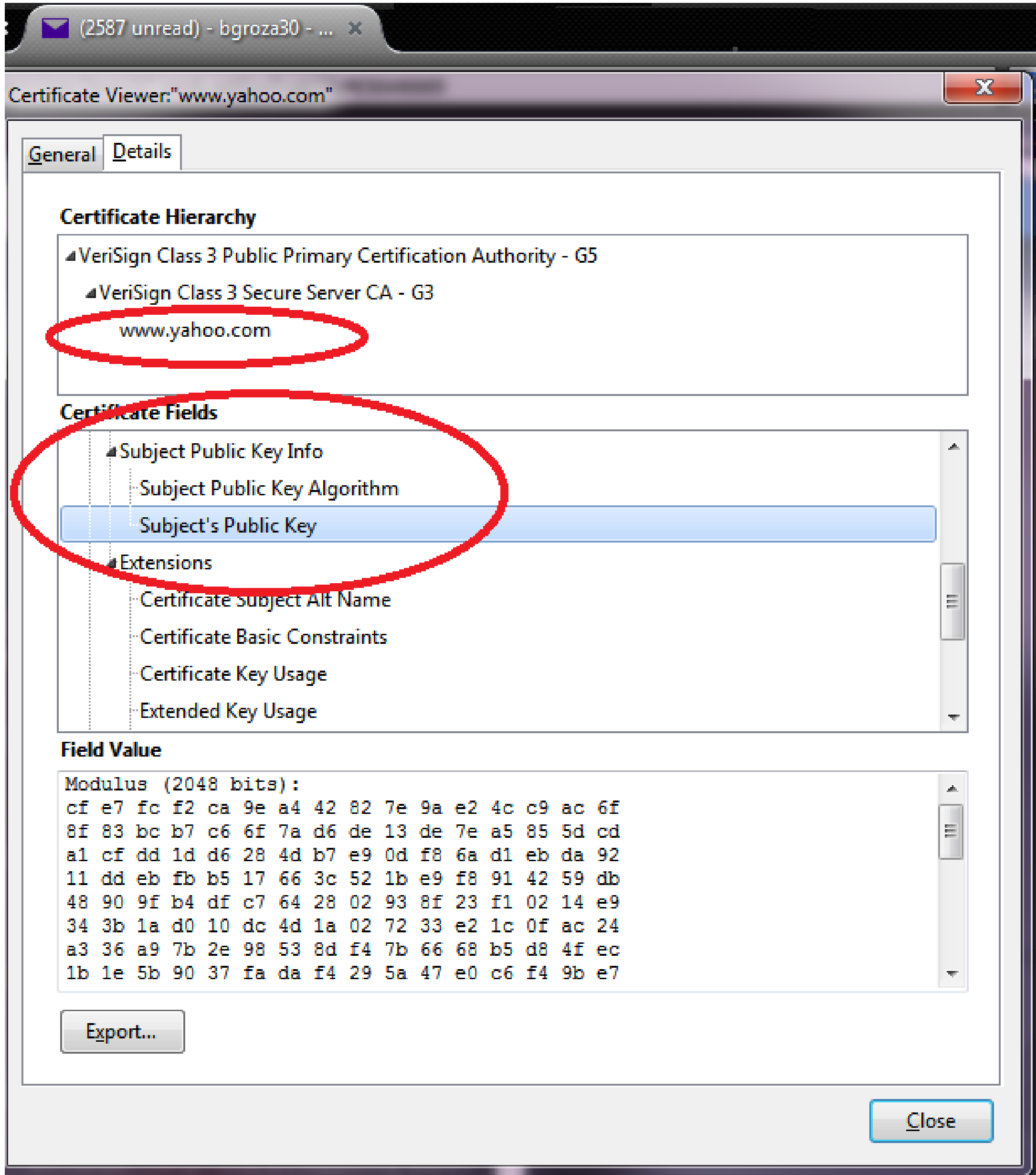
How public-key encryption works (informal)

- Use separate key for encryption and decryption (note that the decryption key must not be recoverable from the encryption key)



Where is public-key encryption used?

- Used everywhere, examples:
  - In your browser: HTTPS, or HTTP over SSL/TLS, whenever you are using the Hypertext Transfer Protocol Secure (HTTPS) to privately read your e-mail, browse, chat or whatever …
  - Behind your routers: IPSEC
  - Etc.



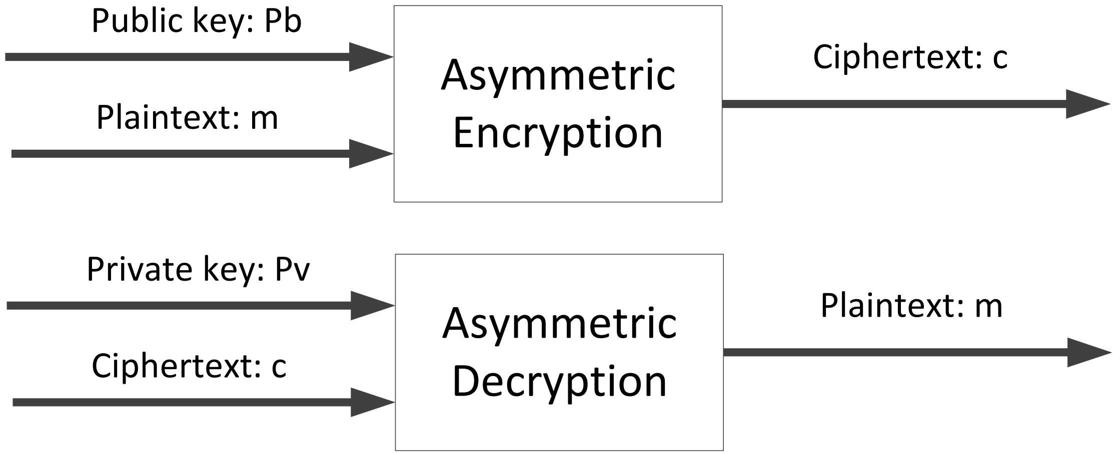
A more formal and constructive account of asymmetric primitives …

you should learn:

i. where is the primitive is used,
ii. what are the standards,
iii. how is it built,
iv. what are its properties

Type of functions (I) Asymmetric encryption schemes

- Description (informal): an algorithm that takes as input a public key (Pb) and message (m, called plaintext) and returns the encrypted message (c, called ciphertext), and a decryption algorithm that takes as input a private key (Pv) and ciphertext (c) and returns the message (m) (a key generation algorithm is also needed)



i.e., $c = e_{Pb}(m)$

i.e., $m = d_{Pv}(c)$

- Example of use: key-exchange for encrypted tunnels SSL/TLS, IPSEC, etc.
- Standards:
  - ➢ To use: RSA (2048 bit or above), Diffie-Hellman (with or without ECC)
  - ➢ Not to use: small key versions or unpadded (textbook) versions of the above
  - ➢ Future use: ECC to completely replace RSA (?)

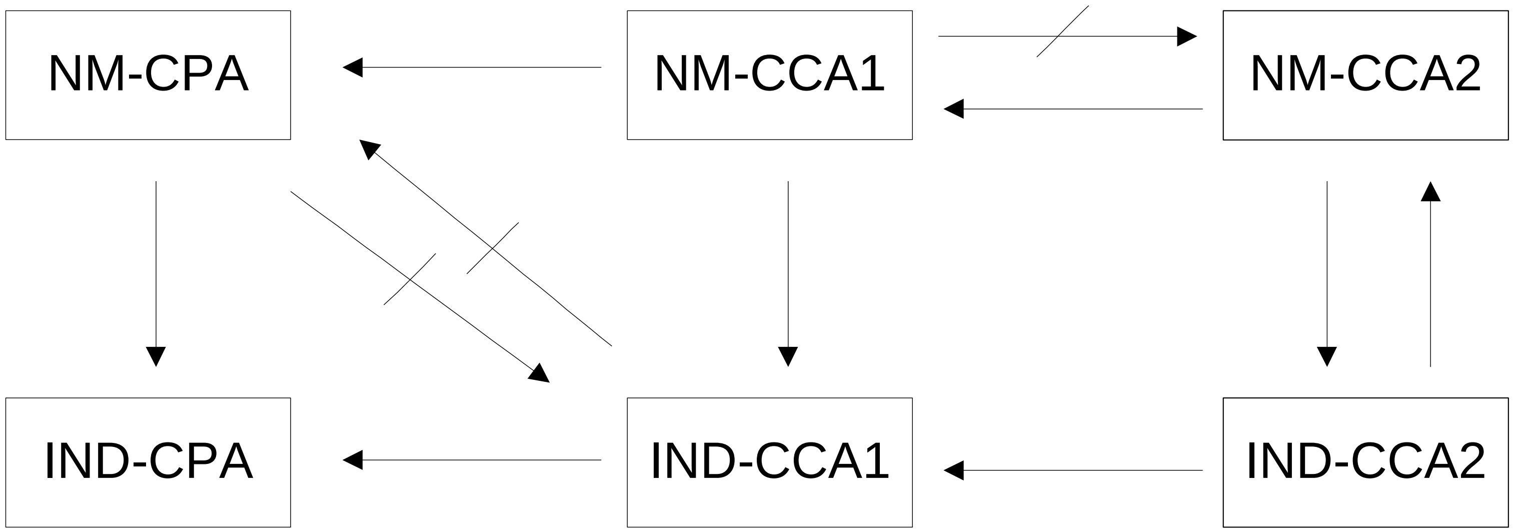
Asymmetric encryption: formal definition

- A symmetric encryption scheme is a triple of algorithms:
  - *Gen is the key generation algorithm that takes the security parameter l, random coins and outputs the public and private key* $(Pb, Pv) \leftarrow Gen(1^l)$
  - *Enc is the encryption algorithm that takes as input the public key and the message, then outputs the ciphertext* $c \leftarrow Enc(Pb, m)$
  - *Dec is the decryption algorithm that takes as input the ciphertext and the private key and outputs the message* $m \leftarrow Dec(Pv, c)$

- A correctness condition enforces that $Dec\big(Pv, Enc(Pb, m)\big) = m$
- A security condition enforces that given the public key Pb it is infeasible to compute the private key Pv, but this is not enough (remember SS/IND/NM security properties)

What are the desired security properties for PKC?

- Similar to what we defined in case of symmetric encryptions: active adversaries (CPA/CCA) and IND/NM:
  - IND – indistinguishability of ciphertexts – what you already know from symmetric cryptosystems
  - NM – non-malleability of ciphertexts – the adversary cannot modify a given challenge ciphertext such that it decrypts to a valid plaintext
- Pictured below are relations among security notions for PKC as proved by Bellare, Desai, Pointcheval & Rogaway '1998



Fundamentals - Number Theory (in 1 slide)

- Definition: A set A together with some operation $\times$ forms an abelian group if the operation $\times$ is:
  i. associative, i.e., $(\mathrm{a} \times \mathrm{b}) \times \mathrm{c} = \mathrm{a} \times (\mathrm{b} \times \mathrm{c})$,
  ii. comutative, i.e., $\mathrm{a} \times \mathrm{b} = \mathrm{b} \times \mathrm{a}$,
  iii. there exists an identity element e such that $\mathrm{e} \times \mathrm{a} = \mathrm{a} \times \mathrm{e} = \mathrm{a}$,
  iv. each element a has an inverse b such that $\mathrm{a} \times \mathrm{b} = \mathrm{b} \times \ \mathrm{a} = \mathrm{e}$.
- $Z_n = \{0,1,2, \dots, n-1\}$ is called the set of integers modulo n, i.e., remainders mod n, then $(Z_n, +)$ forms an abelian group
- $Z_n^* = \{x \in Z_n | \gcd(x, n) = 1\}$ is the set of integers modulo n that are relatively primes to n, then $(Z_n,*)$ forms an abelian group
- The Euler's totient function function is defined as $\varphi(n) = |Z_n^*|$, that is $\varphi(n) = n\left(1-\frac{1}{p_1}\right) \dots \left(1-\frac{1}{p_r}\right)$ where $p_1, \dots, p_r$ are the prime factors of n
- Euler's Theorem – strong result that builds the RSA trapdoor

$$\forall x \in Z_n^*, x^{\varphi(n)} \equiv 1 \bmod n$$

Tools: Computational Number Theory (in 1 slide)

- The following computational problems make public key trapdoors possible, to build public key trapdoors we need both problems that can be efficiently solved (encryption and decryption, i.e., the cryptosystem is efficient) and problems that cannot be efficiently solved (finding the private key from the public key, i.e., breaking the cryptosystem is hard)

Efficiently Computable	Requires (If)
Elementary operations in Z_n^*: -, +, *, /, a^x	-
Greatest common divisor (GCD) and multiplicative inverse, i.e., x^{-1}	-
Primality testing	-
Square root in Z_n^*, i.e., $\sqrt[2]{x} mod n$	**If and only if** factorization known
e-th root in Z_n^*, i.e., $\sqrt[e]{x} mod n$	**If** factorization known
Systems of simultaneous congruences over co-primes (Chinese Remaindering Theorem)	-

Not Efficiently Computable	Requires (If)
Logarithms, i.e., $log_a(a^x) mod\ \mathrm{p}$	Order of the group sufficiently large
Factorization of an integer	Large integers with non-trivial factors
Square root in Z_n^*, i.e., $\sqrt[2]{x} mod n$	If factorization is not known
e-th root in Z_n^*, i.e., $\sqrt[e]{x} mod n$	If factorization is not known

RSA public key cryptosystem

- **Key generation**

1. Generate two random primes p,q
2. Compute $n=pq$, $\varphi(n)=(p\text{-}1)(q\text{-}1)$
3. Choose e relatively prime to $\varphi(n)$
4. Compute d such that $ed\equiv 1 mod\ \varphi(n)$
5. Public key is $Pb=(n,e)$ and private key $Pv=(n,d)$

- **Encryption**

1. Obtain the public key $Pb=(e,n)$
2. Compute $c=m^e \bmod n$, (note that the message must be represented as integer mod n)

- **Decryption**

1. Receive the encrypted message c
2. Compute $m=c^d \bmod n$ by using the private key Pv

- Example (with artificially small numbers)

- **Key generation**

$$p=11,\ q=13,$$
$$n=p\cdot q=143, \phi(n)=(p-1)\cdot(q-1)=120$$
$$e=7,\ d=103,$$
$$Pb=(7,143), Pv=(103,143)$$

- **Encryption**

$$m=5$$
$$c=m^e \bmod n=5^7 \bmod 143=47$$

- **Decryption**

$$c=47$$
$$m=c^d \bmod n=47^{103} \bmod 143=5$$

Real World RSA Example, e.g., 2048 bit

p=8771790518033453833017725332787159777554132254638805970997737935800125548681187458989251920752757442913599538967290134422204254464987500209794754996228353355043342889170014650091827064074824228282220649533618300604182762534874587197130742810668322972641036801070096428526988411279367560818649799455463660957

q=35854546003977174565510740065268320064652149413063859625641831730127272711465083113461063828495283882464922609911879425279857210780829161482812728546396118096771447737562161014412999014651809527526115364454639300101116444258051893130616890759713361315636239668678324095023104457779566373583973773185341848137

n=3145085666660812421371212830252628201231454745470010028666451733216644680210723064697610581964025286025381168821678505106502877475125973775982616968206216778567184889284360224141175515609820975143283443761155573339595327933374389334538451641646301931628708789331972800805576939076262518928316668528736277298123601348376805406893557725503439946830084802329864534893977812775991800171411897316501358120184949585247426231754317892299571256572382692094062287379459464485753380952291568996685349589503706309590173800607443930375509595894933377633658450815877191855015707320956169677690099046904961042512131905014350087109

phi=3145085666660812421371212830252628201231454745470010028666451733216644680210723064697610581964025286025381168821678505106502877475125973775982616968206216778567184889284360224141175515609820975143283443761155573339595327933374389334538451641646301931628708789331972800805576939076262518928316668528736277298000028897192493693997869732110300028989891330370412615558358601687863271973134939613147775084162091273646508232169537122797671501141678528513302008870779812838548504323029261482180523510777072553781837786619186329670210389102006897305910817245495507566738430851207749154140006177846027108109508332373544578016

e=65537

d=804639384971876068009385927412695378946973491574158996454680504308153887927326451100673461217187420400499962733007057007808318150440777000807185844727613969608556831082150050783146770072323546854867227696466049684681857019071492225341505083776851923152090578920749926775822947914791547597099128449903431061164937127412401265960637224747751660681348061954631861773761366472380540475968590757795577228682206759315058722992914368035591468325411964044471592882438395438667359994876659402653780882634224567942689994480124161764810986070057366787131035487666862617628259915501172315904992349114917321843723792802649220129

m=100

c=2548976845452449395879597636565818040429249216911776554340786808269869607248582578109482386937930738029539853835271441835300964929299252492801427967763066935011758480260571665283197031694055236443568768373001705826149336572950881281964314788586957659272876801920455855808566073459620866409255904768797598183549291311221205768157523067139811766553103059906211473702387598423211448621690479755865645281861912934219969090063032346816096468223524632470559925217068733964098762893583266796609734868085590529178186029803475510680587805155014963979543880268829325102999518057911194673397595580615913766442151206798081405953

Algorithms (I) Extended GCD (Greatest Common Divisor)

- Based on the Euclidean Algorithm for GCD
- For 2 integers a and b, computes 2 integers x and y such that ax + by = gcd(a, b)
- Scope: it essentially computes the modular inverse, e.g., replace a with e and b with phi(n) in RSA

  - Input: a, b (two integers)
  - Output: oldx, oldy (two integers such that oldx × a + oldy × b = gcd(a, b))
  - Complexity: $O((\lg n)^2)$

1. oldr ← a; r ← b;
2. oldx ← 1; x ← 0;
3. oldy ← 0; y ← 1;
4. While r != 0 do
   4.1. q ← Floor[oldr/ r];
   4.2. aux ← r; r ← oldr - q × r; oldr ← aux;
   4.3. aux ← x; x ← oldx - q × x; oldx ← aux;
   4.4. aux ← y; y ← oldy - q × y; oldy ← aux;
5. Return (oldx, oldy)

Algorithms (II) Modular Exponentiation (RSM)

- Input: m, n and e (as binary representation on k bits, i.e., $e=e_k e_{k-1} \ldots e_1 e_0$)
- Output: $m^e \bmod n$
- Complexity: $O(\log_2 e)$

1. Res $\leftarrow$ 1
2. Acu $\leftarrow$ m
3. If $e_0 = 1$ Res $\leftarrow$ Acu
4. For i=1 to k do:
   3.1. Acu $\leftarrow$ Acu^2 mod n
   3.2. If $e_i = 1$ then Res $\leftarrow$ Res x Acu mod n

4. Return Res

Algorithms (III) Random Prime Generation

- To generate a *k*-bit random prime, generate an odd random k-bit integer *p* and apply a primality test to *p*
- Probabilistic primality testing: provides partial information whether an integer is prime or not
- Example: **Fermat testing**, if a prime candidate fails the test, then the integer is composite, otherwise, it can be a prime:
  - How to do it: for any candidate prime p, pick a random x ϵ (1,p-1) and check that x^{p-1} = 1 mod p (if not, the number is not prime, if yes, repeat several times)
- More efficient tests exist: see HAC, Chapter 4, https://cacr.uwaterloo.ca/hac/

RSA Computational requirements in brief

- Generating keys is the most intensive computational step as generation of two random primes requires: generating a random integer + testing for primality (there are ~x/ln(x) prime numbers up to x, so probability of success is ~1/ln(x))
- Encryption is usually the most efficient step since one can choose special form exponents: 3, 5, 65537 (note that primes of the form 1000…0001 are preferred)
- Decryption is always more computationally intensive than encryption because the decryption exponent is in the order of the modulus n
- Questions: why are exponents of the form 100…001 preferred? Why is the decryption exponent in the order of n?

RSA CRT speed-up

- For faster computations, RSA decryption is usually performed with Chinese-Remaindering-Theorem
- This allows performing decryptions modulo p and q then combines them to get the result

$$\begin{cases} m_1 = c^{d_1} mod\ p \\ m_2 = c^{d_2} mod\ q \end{cases} \Longrightarrow m = m_1 q(q^{-1} mod\ p) + m_2 p(p^{-1} mod\ q)$$

- where $d_1 = d\ mop\ (p - 1)$ and $d_2 = d\ mop\ (q - 1)$
- Questions: why is the decryption exponent reduced mod p-1 and q-1? Why this works faster than standard decryption?
- Note: there are alternative ways for doing the same, e.g., see in .NET implementation

Mathematical security & properties (or vulnerabilities?)

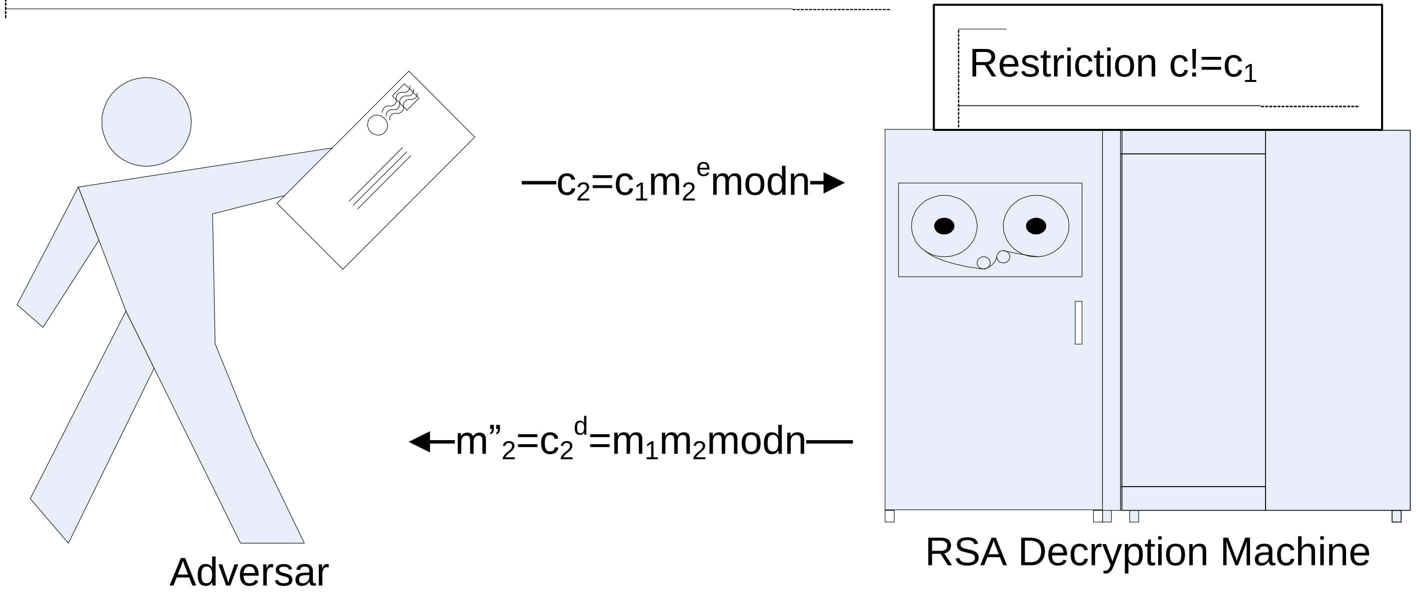
- Relation between RSA and Factoring: no proof of equivalence between breaking RSA and factoring exists so far, some facts:
  - ❖Factoring obviously leads to breaking the RSA
  - ❖Computing a private-public RSA key pair also leads to factoring (discussed in laboratory exercises)
  - ❖Proving that RSA decryption leads to factoring seems to be hard (or maybe this equivalence is not true after all)

- Many interesting properties behind the text-book RSA trapdoor, some of them opening door for attacks (all these will be discussed in laboratory exercises):
  - ❖Small messages
  - ❖Small encryption exponents
  - ❖Small decryption exponents
  - ❖Messages that do not encrypt

Why text-book RSA fails in front of active adversaries?

- Question: Consider IND (indistinguishability) as security property, is textbook RSA secure under this property?
- Answer: No, in fact no deterministic public key cryptosystem is.
- Question: Consider an CCA adversary, can the adversary recover the full plaintext in case of textbook RSA?
- Answer: Yes, textbook RSA is completely insecure under CCA adversaries

Adversary has target ciphertext c_1



Target message recovered $m_1=m''_2m_2^{-1}modn$

Introducing RSA-PKCS#1

- RSA encryption according to PKCS#1 (Public-Key Cryptography Standards)
- Before encryption, message is padded as:

$$(00...00 \,||\, 00...10 \,||\, random \,||\, 00...00 \,||\, m)^e \bmod n$$

- Note: the random number below has k-3-|m| bytes (at least 8) where k is the byte length of the modulus

- Good news: previous CCA attacks does not work, can be (somewhat) securely used in practice
- Bad news: there are some attacks for special cases (small exponents, special messages, etc.), and more, there is no proof that RSA-PKCS#1 is secure
- Good news: newer versions of PKCS#1 include RSA-OAEP as improved encryption/decryption method

RSA encryption example (OpenSSL)

- Create a file

  echo "Something secret ..." > secret.txt

- Generate RSA key

  openssl genrsa -out privatekey.pem 2048

- Encrypt the file

  openssl rsautl -encrypt -pkcs -inkey privatekey.pem -in secret.txt -out ciphertext.bin

- Decrypt as raw (padding not removed) and print the output

  openssl rsautl -decrypt -inkey privatekey.pem -in ciphertext.bin -raw -hexdump

```
[10/27/23]seed@VM:~/Desktop$ openssl rsautl -decrypt -inkey privatekey.pem
 -in ciphertext.bin -raw -hexdump
0000 - 00 02 37 44 4b 4a 12 ed-2b e6 5d eb 05 31 2a 7c   ..7DKJ..+.]..1*|
0010 - f4 e4 ed 4d b2 0f 78 4e-6c e3 e5 72 82 3f de 1f   ...M..xNl..r.?..
0020 - 97 9c e7 95 59 03 3a eb-fe 0e 83 f4 40 c8 37 c2   ....Y.:.....@.7.
0030 - 4c 96 7d 0d 0d b0 40 35-5d b3 5d e1 90 34 aa cc   L.}...@5].]..4..
0040 - 04 fe ce d1 c3 07 d1 74-59 38 8a d1 61 aa 85 8a   .......tY8..a...
0050 - 0b be 3d 80 e9 1f 5a e8-c6 f6 79 ab ca bd a8 6d   ..=...Z...y....m
0060 - 7b fb 46 cb 7c ba 06 92-52 f3 56 39 07 cd 92 dd   {.F.|...R.V9....
0070 - 10 23 ed f3 f5 e8 0f cc-42 4c cd fa 42 1c e5 e0   .#......BL..B...
0080 - 94 2c 6b 44 71 16 bd f4-a0 f3 50 da 06 3c fb 0b   .,kDq.....P..<..
0090 - df b4 b7 eb 98 48 da 74-46 ec 7f 8c 92 77 b4 4e   .....H.tF....w.N
00a0 - a7 a0 86 c6 ee a5 d6 1a-90 5f 4b 3a f5 f0 e0 86   ........._K:....
00b0 - 01 0d 6e 77 30 fd 5a 10-9f 40 70 43 c2 e5 2e 42   ..nw0.Z..@pC...B
00c0 - 84 e2 44 08 57 4b 30 9b-be 37 2e 6b b2 45 3f ce   ..D.WK0..7.k.E?.
00d0 - 74 fd d5 1f d5 df 59 56-36 01 34 76 a3 44 f2 1e   t.....YV6.4v.D..
00e0 - 95 fd 95 40 9d 98 36 96-0e ed 00 53 6f 6d 65 74   ...@..6....Somet
00f0 - 68 69 6e 67 20 73 65 63-72 65 74 20 2e 2e 2e 0a   hing secret ....
```

- For more rsautl commands please see:

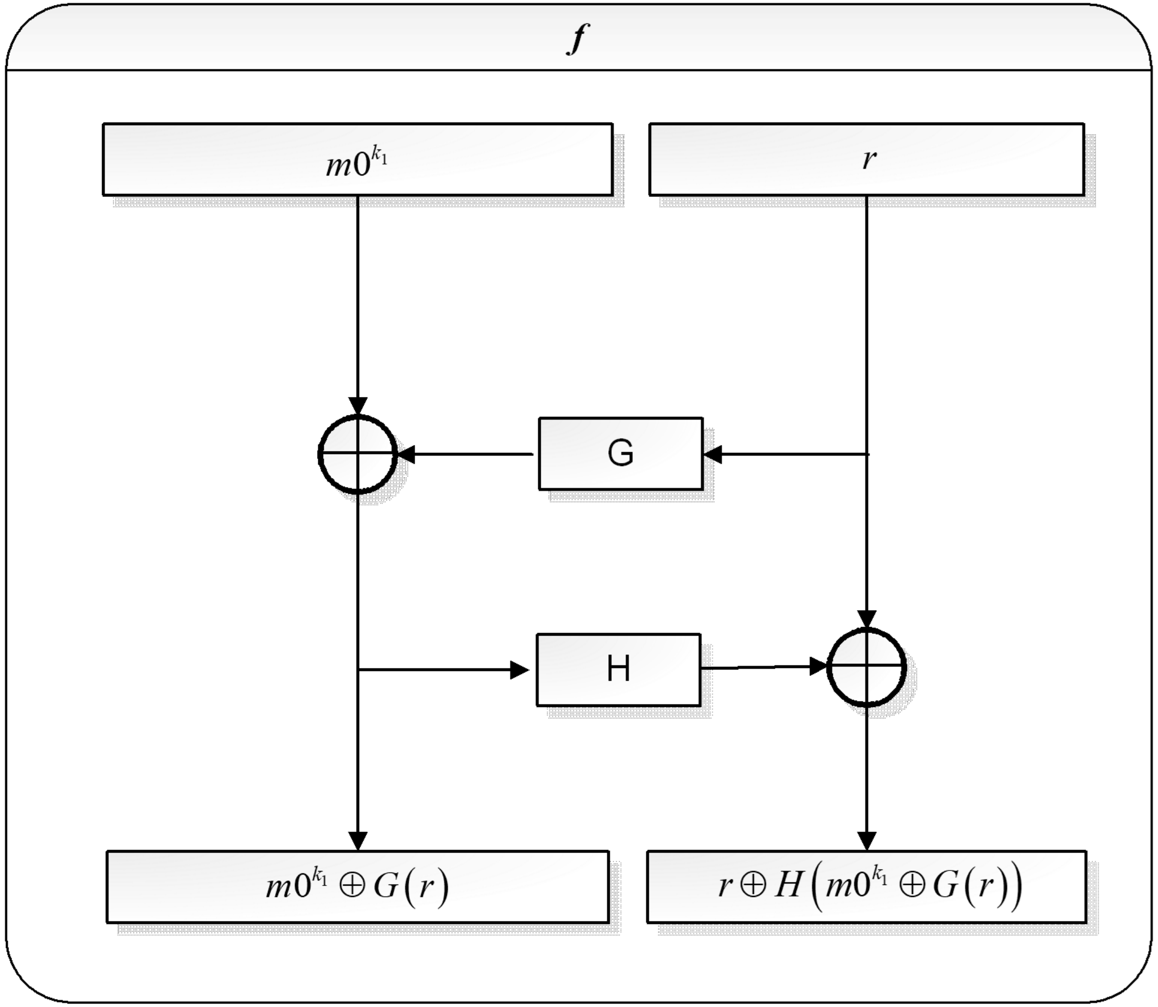
  https://www.openssl.org/docs/man1.0.2/man1/rsautl.html

Secure versions of RSA: RSA-OAEP

- Bellare & Rogaway 1991
- Main idea: embed a Feistel network under RSA:

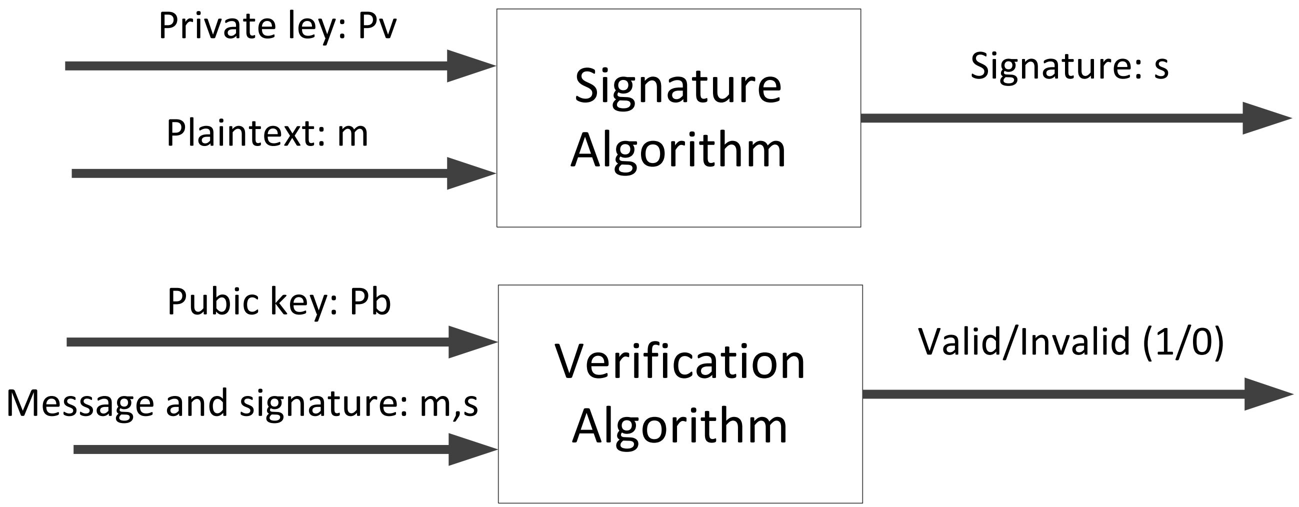
$$E(x) = f\left(x \oplus G(r) \,\|\, r \oplus H\left(x \oplus G(r)\right)\right)$$

- OAEP has provable NM/IND security under CCA adversaries
- Some historical turnarounds for OAEP:
  - Bellare & Rogaway proved that OAEP gives security on any trapdoor
  - Shoup proved they were wrong
  - Fujisaki & Okamoto proved that security holds for RSA
  - All proofs are in the Random Oracle Model but hash functions in practice are not random oracles



Type of functions (II) Digital signatures

- Description (informal): the electronic "equivalent" of a handwritten signature, the signing algorithm takes the private key and message and returns a signature, the verification algorithm takes the public key, message and signature and checks if the input is genuine. (a key generation algorithm is also needed)



i.e., $s = Sig_{Pv}(m)$

- Example of use: document signing, driver signing, public-key certificate signing, SSL/TLS, etc.
- Standards:
  - To use: RSA-PSS, RSA-FDH, RSA-PKCS
  - Not to use: small key versions of the above or unpadded (textbook) versions
  - Future use: N/A

Digital signatures: formal definition

- A symmetric encryption scheme is a triple of algorithms:
  - *Gen is the key generation algorithm that takes random coins, the security parameter l and outputs the public and private key* $$(Pb, Pv) \leftarrow Gen(1^l)$$
  - *Sig is the signing algorithm that takes as input the private key and the message, then outputs the signature* $$s \leftarrow Sig(Pv, m)$$
  - *Ver is the verification algorithm that takes as input the signature and the public key and outputs the 1 if the signature is valid or 0 otherwise* $$\{0,1\} \leftarrow Ver(Pb, s, m)$$

- A correctness condition enforces that $Ver\big(Pb, Sig(Pv, m)\big) = 1$
- A security condition enforces that given the public key Pb it is infeasible to compute the private key Pv, but this is not enough (see security properties)

What do we mean by breaking a signature?

- *Existential forgery* – find a valid message-signature without controlling the message
- *Selective forgery* – forge signature over messages that have a particular structure
- *Universal forgery* – forge signatures over any kind of messages (without knowing the private key)
- *Total break* – recover the private key (sign anything)

What are the adversary capabilities?

- *Key-only* – adversary knows only the public key
- *Known-messages* – adversary has valid messages-signature pairs but not at his choice
- *Chosen message* – adversary has messages-signature pairs at his choice (adaptive chosen-message is a flavor of this notion where the adversary is allowed to chose messages after fixing the target to be forged)

To sum up: **unforgeability under chosen-message attacks** is the desired property (adversary cannot forge signatures, even if he has full access tot the signing oracle)

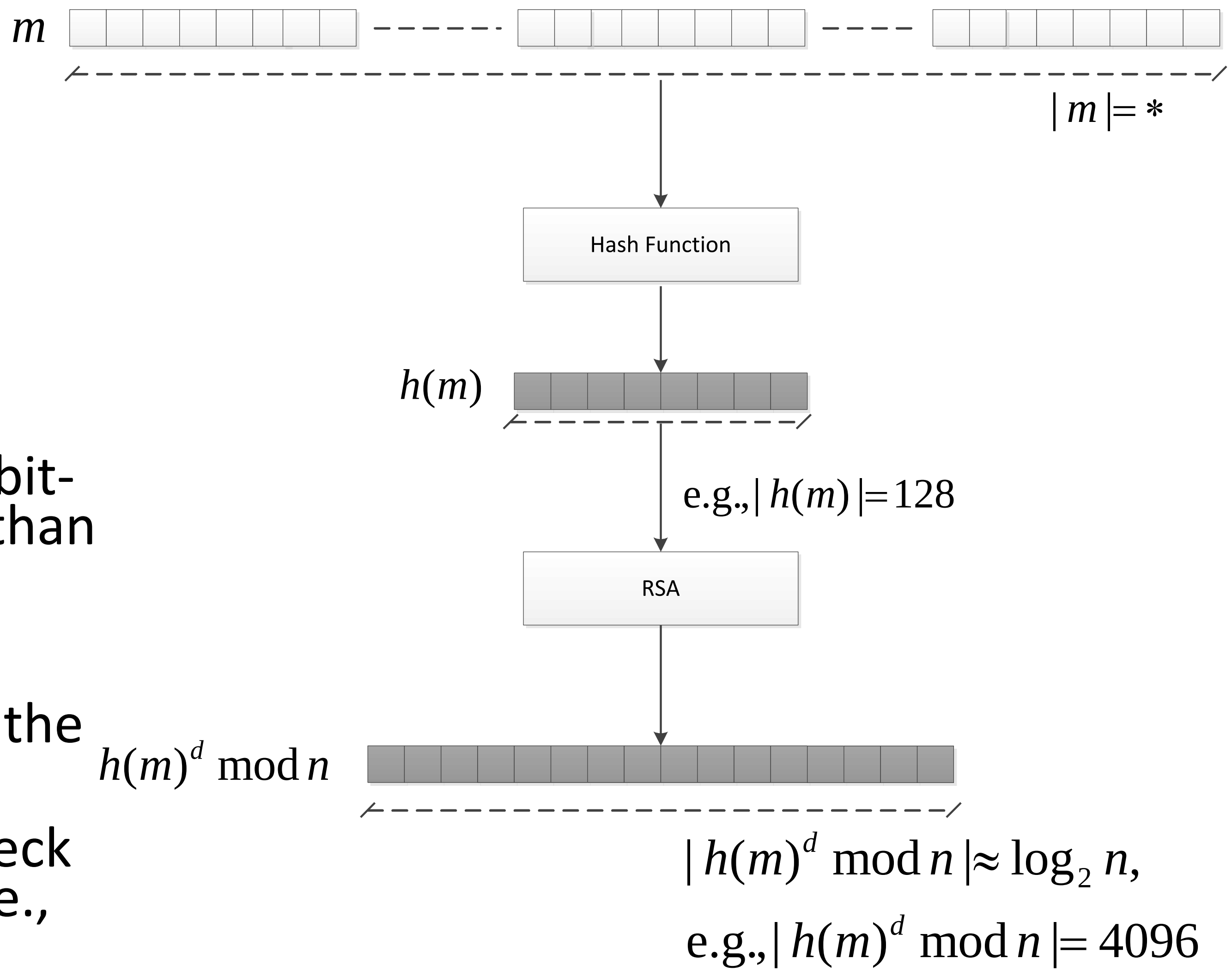
The textbook RSA signature (hash then sign)

- Principle:
  - ➢To sign: hash the message then use the private key to sign the hash
  - ➢To verify: use the public key to recover the hash then compare it to the hash of the original message
- **Sign**

1. Compute $s=H(m)^d \bmod n$, (note that the bit-length of the hash must be less or equal than that of the modulus n)

- **Verify**

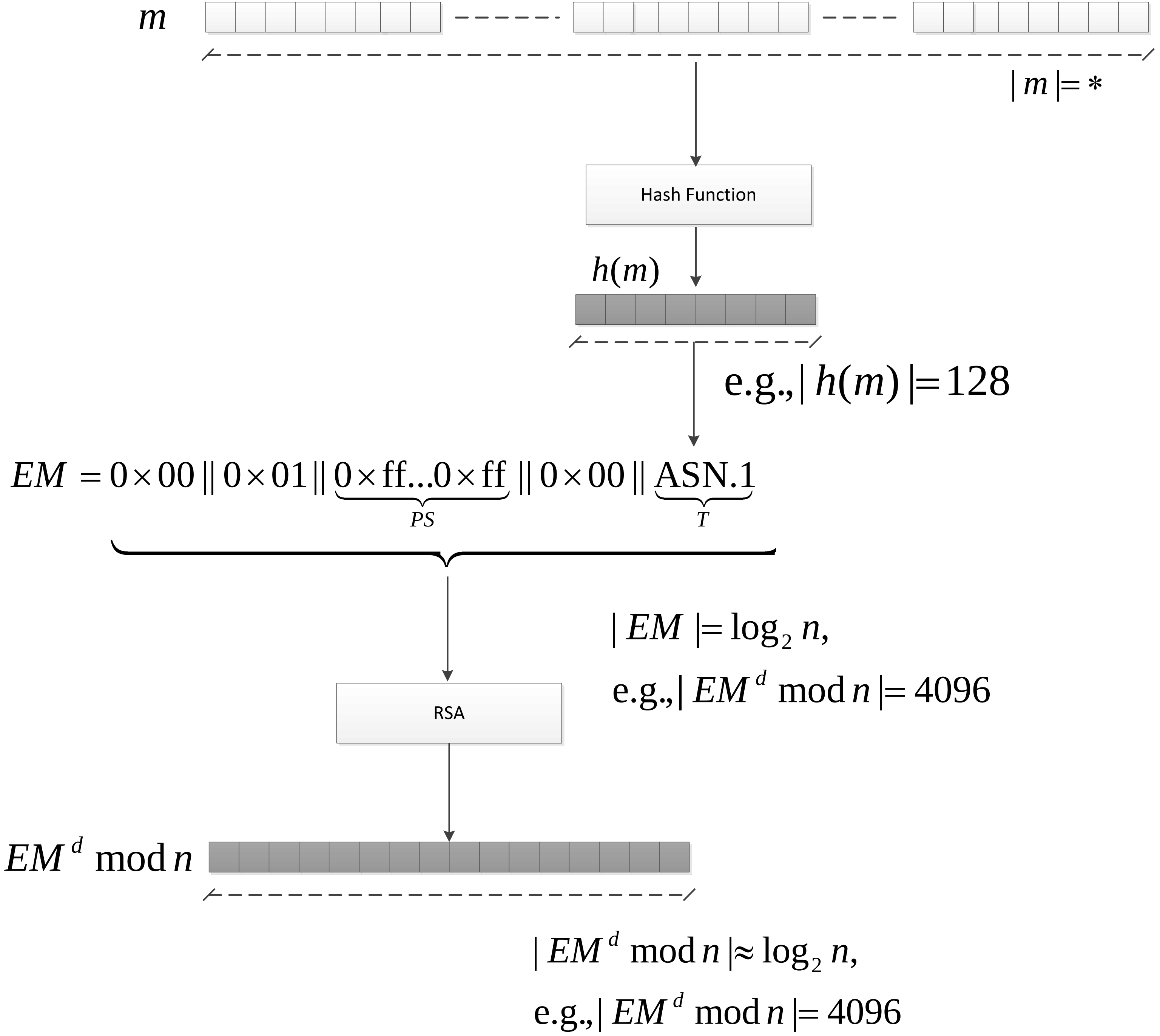
1. Recover the hash from the signature with the help of the public key $h' = s^e \bmod n$
2. Compute the hash of the message and check that it is equal with the recovered hash, i.e., $h' = H(m)$



- Note: in case of RSA the signing algorithm is the reverse of encryption algorithm, this leaves the impression that in general signing is the reverse of encryption, but turns out not to be the case for many other public key cryptosystems, e.g., ElGamal

RSA – PKCS v.1.5

- Standard published by RSA laboratories as of 1991, current version is from 2012



RSA signing examples (OpenSSL)

- Generate an RSA key

  openssl genrsa -out privatekey.pem 2048

- Compute the digest of a txt file:

  openssl dgst -sha256 -binary -out sha256.dgt hello_world.txt

- Sign the file with the private key:

  openssl rsautl -sign -inkey privatekey.pem -in sha256.dgt -out sha256_signed.dgt

- Verify the signature as raw (padding not removed) and print the output

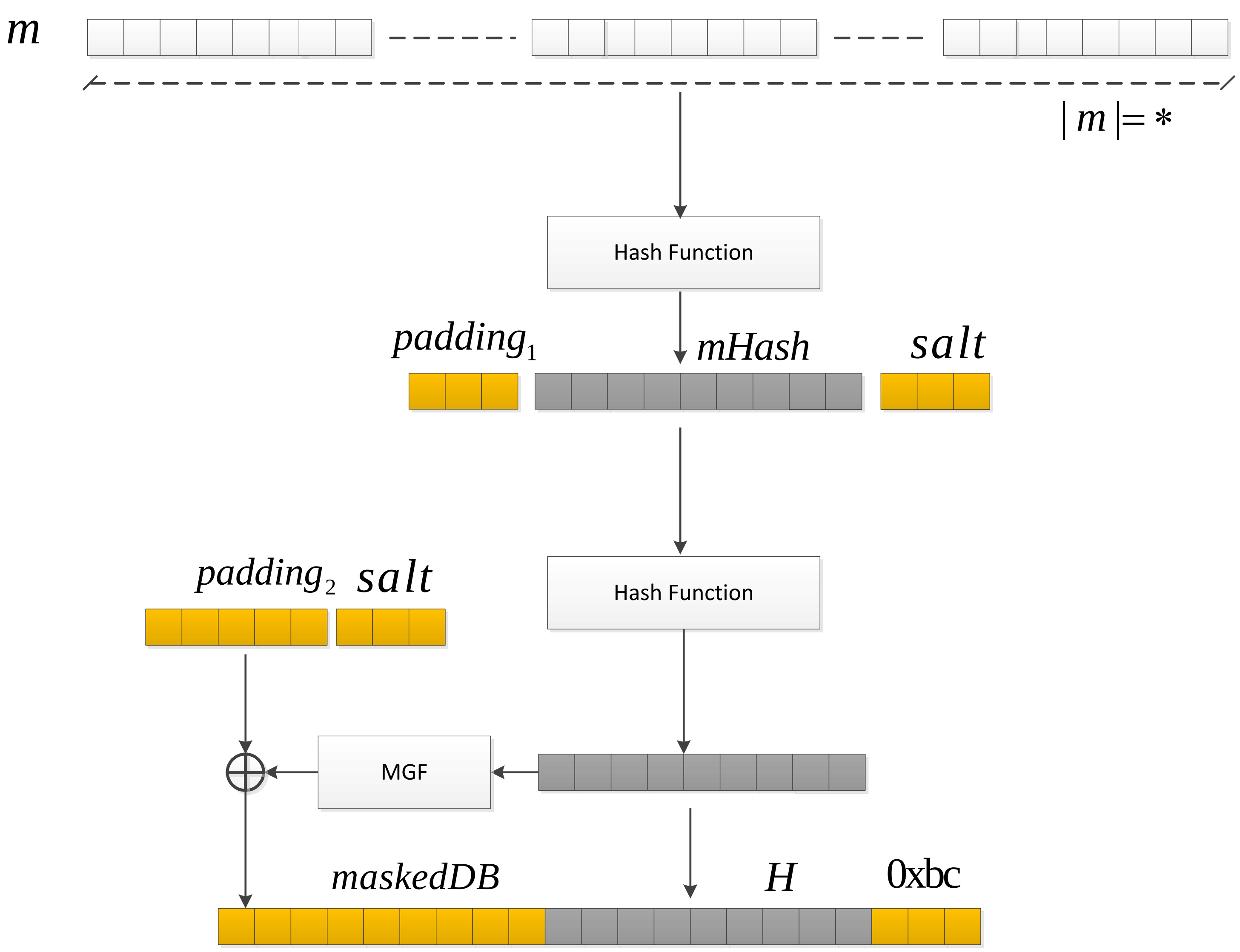
  openssl rsautl -verify -in sha256_signed.dgt -inkey privatekey.pem -raw -hexdump

```
[10/27/23]seed@VM:~/Desktop$  openssl rsautl -verify -in sha256_signed.dgt
 -inkey privatekey.pem -raw -hexdump
0000 - 00 01 ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0010 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0020 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0030 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0040 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0050 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0060 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0070 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0080 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
0090 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
00a0 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
00b0 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
00c0 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff   ................
00d0 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff 00   ................
00e0 - 79 fc e7 ea fe 1c 6d bc-d5 38 22 a0 b9 7f 83 22   y.....m..8"...."
00f0 - ef 9b e5 45 df c8 c0 64-40 dc 3a 5d 9f 0e d0 f4   ...E...d@.:]....
```

- For more rsautl commands see: https://www.openssl.org/docs/man1.0.2/man1/rsautl.html

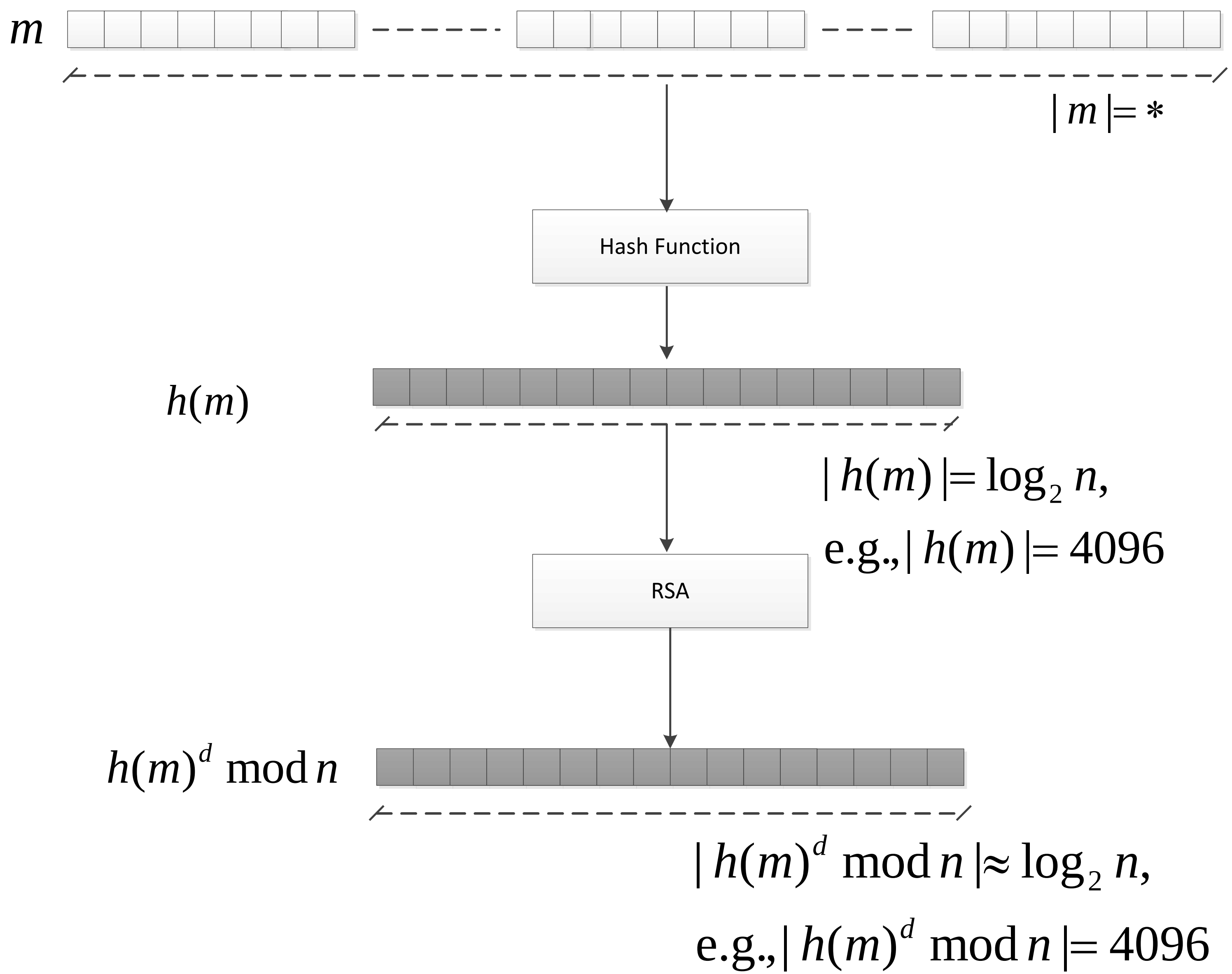
RSA – Probabilistic Standard Signature (PSS)

- Designed by Bellare & Rogaway, also included in newer versions of PKCS



RSA – Full Domain Hash (FDH)

- Principle: use a hash function that spans over the entire domain of the modulus
- Security: RSA-FDH is provable secure in the Random-Oracle-Model



The Rabin cryptosystem

- Published in '79 by M.O. Rabin

- Key generation

1. Generate two random primes p,q
2. Fix e = 2
3. Public key is $Pb=(2,n)$ and private key $Pv=(p,q)$

- Encryption

1. Obtain the public key $Pb=(2,n)$
2. Compute $c=m^2 \bmod n$

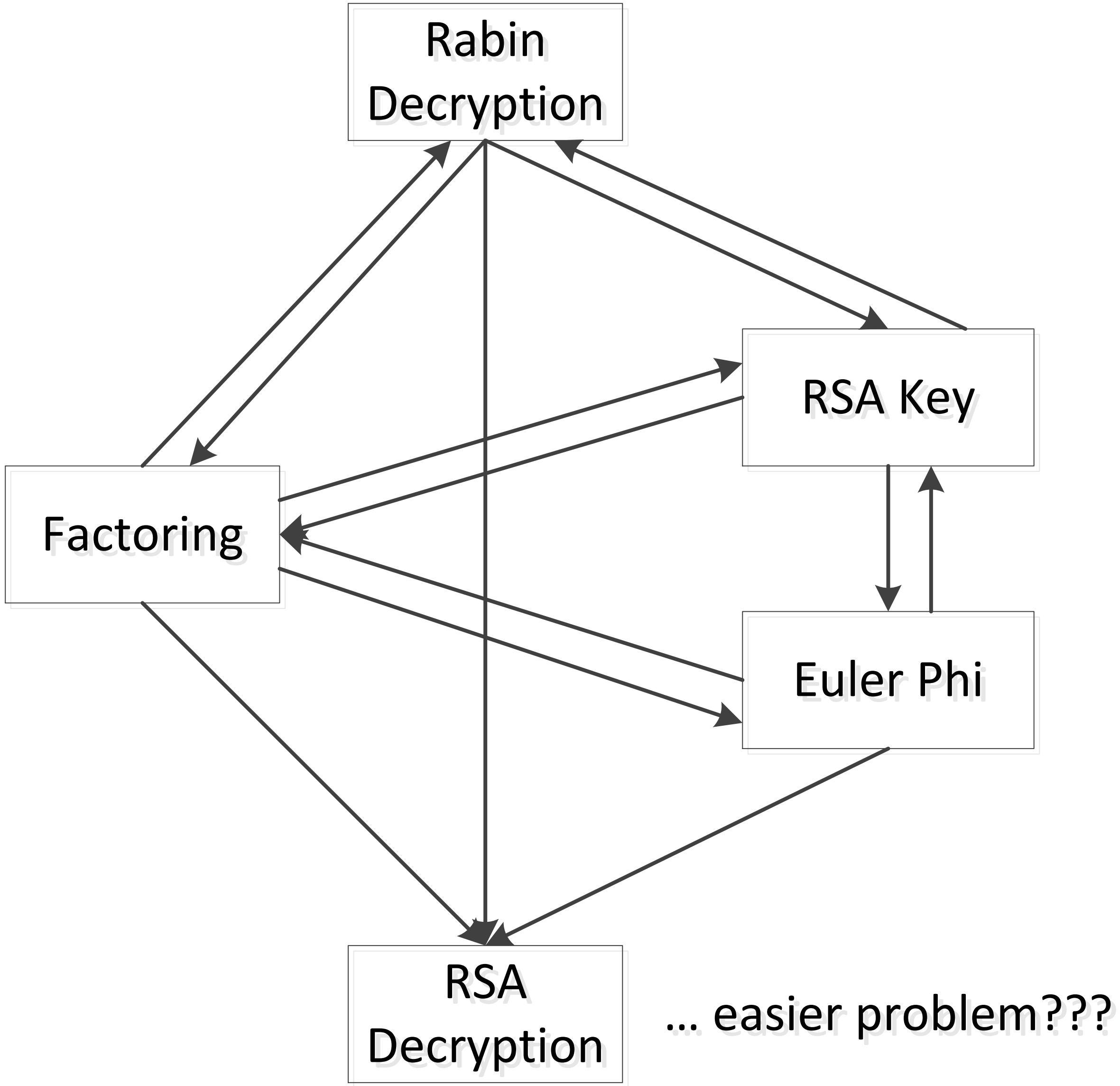
- Decryption

1. Compute m as the square root of c

- Notes:
  - Rabin is not a particular case of RSA, 2 cannot be an RSA encryption exponent
  - Requires padding similar to the RSA to be secure
  - If the modulus is the product of two primes then there are 4 square roots (need redundancy/padding to decide which of them was the message)
- Question: why 2 cannot be an RSA exponent? Why are there 4 roots?

Recap: computational problems behind factoring based schemes

- All problems seem to nicely reduce one to another: Factoring, Rabin Decryption, RSA Key Generation and Euler Phi computation
- Is just RSA Decryption for which there is no proof that it will allow solving the others



- Note: arrow from P1 to P2 means that if you could solve P1, you can solve P2

The Diffie-Hellman-Merkle Key exchange – The Discrete Logarithm Terrain

- Method for securely exchanging a key over an insecure channel between two parties

- Key setup

1. Fix a prime p
2. Choose a generator g of Z_p

- Exchange

1. $A \rightarrow B$: $g^a mod\ p\ n$ (a is a fresh secret random value)
2. $B \rightarrow A$: $g^b mod\ p$ (b is a fresh secret random value)

Where

- Compute

1. A computes $(g^b)^a mod\ p = g^{ba} = g^{ab}$
2. B computes $(g^a)^b mod\ p = g^{ab}$

- Notes:
  - The protocol above is vulnerable to a man-in-the-middle attack (but it’s trivial to derive secure versions of it)
  - The order of the group Z_p must have a large prime factor, usually one works with $p = 2q + 1$(this is usually called a safe prime)

ElGamal encryption

- Key generation

1. Generate a random prime p
2. Choose a generator g
3. Choose a random value $a \in (1, p-2)$
4. Compute $g^a mod\ p$
5. Public key is $\mathrm{Pb} = (\mathrm{p}, \mathrm{g}, g^a)$ and private key is $\mathrm{Pv} = (\mathrm{p}, \mathrm{g}, a)$

- Encryption

1. Obtain the public key $\mathrm{Pb} = (\mathrm{p}, \mathrm{g}, g^a)$
2. Choose a random value $\mathrm{k} \in (1, p-2)$
3. Compute $c_1 = g^k mod\ p$, $c_2 = m(g^a)^k\ mod\ p$
4. Send $c = (c_1, c_2)$

- Decryption

1. Receive the encrypted message c
2. Recover the message as $\mathrm{m} = {c_1}^{-a} c_2$

- Remark:
  - Same remark for the order of the group as in the case of Diffie-Hellman
  - When computing $c_2 = m(g^a)^k\ mod\ p$ multiplication is used to conceal the message, but you can use other operations as well (XOR, symmetric encryption, etc., with the Diffie-Hellman key)

ElGamal Signature

- Published by Taher ElGamal in ‘84 (dlogs were used in crypto since the ’76 work of Diffie&Hellman, but a dlog signing scheme eluded for many years)

- Key generation

1. Generate a random prime p
2. Generate a random integer a $\in (1, p-2)$
3. Compute $y = g^a mod\ p$
4. Public key is $Pb = (g, y, p)$ private key is $Pv = (g, a, p)$

- Sign

1. Generate random $\mathrm{k} \in (0, p-1)$
2. Having h the hash of the messge, compute $r = g^k\ mod\ p$ and $\mathrm{s} = k^{-1}(h - ar) mod\ (p-1)$
3. Output the pair (r, s) as the signature

- Verify

1. Compute the hash of the message h
2. Verify that $r \in (0, p)$ and $s \in (0, p-1)$ return 0 if not
3. Verify that $g^h = y^r r^s$ return 1 if so or 0 otherwise

- Remarks:
  - Key generation is cheaper than for RSA (only one prime needed), more, the prime field can be a global parameter, i.e., more entities can use the same fixed p
  - Signing requires more computations but these are done over a prime p that is usually smaller than the RSA modulus, therefore its faster
  - Verification is slower than for RSA (if special public exponents are used, i.e., 65537, etc.)

ElGamal – notes on security

- So far there exist no security reductions (proofs) for ElGamal signatures, nor for DSA (next), Schnorr signature is the simplest dlog based signature that has a security reduction to the dlog problem but is quite absent in practice
- Selecting a random k is mandatory for the security of the ElGamal signature, if k is not random then the secret key is trivial to recover:

Let the first signature be

$$\{r_1 = g^k \, mod \, p, s_1 = k^{-1}(h_1 - \boldsymbol{a} r_1) mod \, (p-1)\}$$

and the second

$$\{\mathrm{r}_2 = \mathrm{g}^{\mathrm{k}} \, \mathrm{mod} \, p, \; s_2 = k^{-1}(h_2 - \boldsymbol{a} r_2) mod \, (p-1)\}$$

then

$$\mathrm{k} = (s_1 - s_2)/(h_1 - h_2)$$

and now $\boldsymbol{a}$ can be recovered from s_1 or s_2

Schnorr Signature

- Published by Peter Schnorr in “Efficient Signature Generation by Smart Cards”, Journal of Cryptology, p. 161-174, 1991.

- Key generation

1. Generate a random prime of the form $p = k * q + 1$ where q is a smaller prime (usually a 160-bit prime)
2. Let g be a generator of the subgroup of order q from Z_p
3. Generate a random integer $\mathrm{x} \in (1, q-1)$
4. Compute $y = g^a mod\ p$
5. Public key is $Pb = (g, y, p)$ private key is $Pv = (g, x, p)$

- Sign

1. Generate random $k \in (0, q-1)$
2. Compute $r = g^k\ mod\ p, e = H(r||m)$ and $\mathrm{s} = k - ae\ mod\ q$
3. Output the pair (s, e) as the signature (note that this pair is computed modulo q and thus more compact than in the regular ElGamal signature)

- Verify

1. Compute $r_v = g^s y^e\ mod\ p$ and $e_v = h(r_v||m)$
2. Verify that $e_v = e$ return *true* otherwise return *false*

- Remarks:
  - Note that the pair (s, e) is 2x160 bits (assuming q is 160 bits) and thus much more compact compared to the regular ElGamal signature

The Digital Signature Algorithm - DSA

- Also known as DSS – Digital Signature Standard, standardized by NIST
- It is a variation of the ElGamal signature, all previous remarks apply here as well
- It differs from ElGamal mostly at key generation and verification, resulting in smaller signatures (a small but true practical advantage)

- Key generation

1. Generate a random prime p such that another prime q of 160 bits divides $p-1$
2. Select a generator g of order q
3. Generate random a $\in (0, q-1)$
4. Compute $y = g^a mod\ p$
5. Public key is $Pb = (g, y, p)$ private key is $Pv = (g, a, p)$

- Sign

1. Generate random $\mathrm{k} \in (0, q-1)$
2. Having h the hash of the messge, compute $r = g^k\ mod\ p\ mod\ q$ and $\mathrm{s} = k^{-1}(h + ar) mod\ q$
3. Output the pair (r, s) as the signature

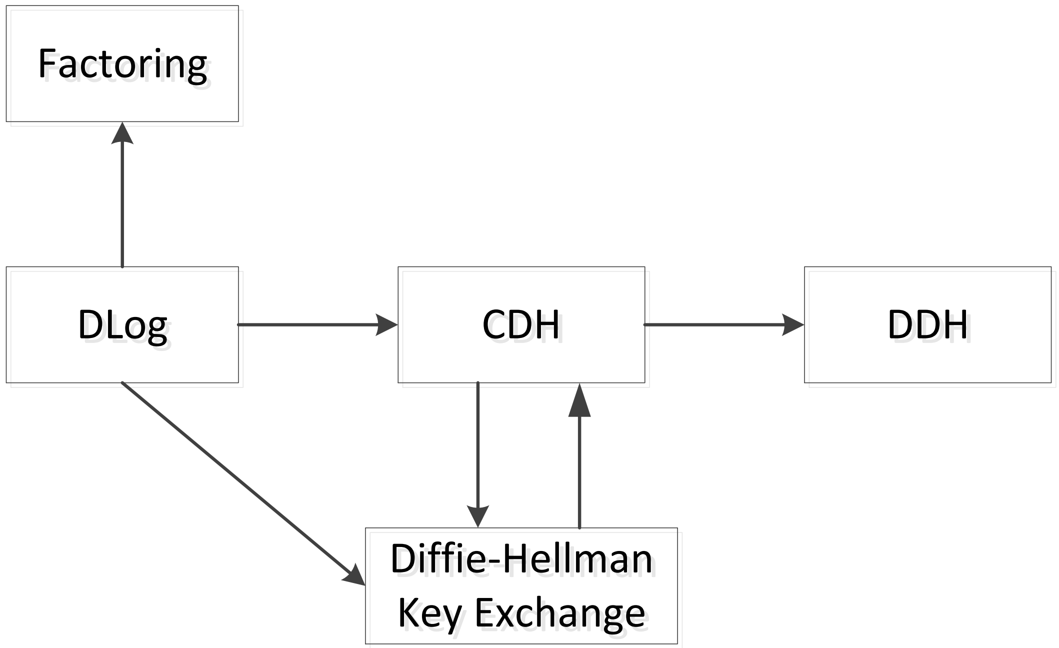
- Verify

1. Compute the hash of the message h
2. Verify that $r \in (0, q)$ and $\mathrm{s} \in (0, q)$ return 0 if not
3. Verify that $v = r$ and return 1 if so or 0 otherwise, where $v = (\ g^{u_1} y^{u_2} mod\ p) mod\ q$, $u_1 = wh\ mod\ q, u_2 = rw\ mod\ q$, $\mathrm{w} = s^{-1}\ mod\ q$

- Remark: parameter q here is fixed at 160 bits according to the output size of SHA1, it can be set to 224 and 256 for SHA2 (see FIPS 186-3)

Computational problems behind DLog based schemes

- All of the previous are apparently based on the difficulty of computing discrete logarithms, but there are three flavors of this problem:
    - Decisional Diffie-Hellman problem (DDH) – let $y_0 = g^{ab}, y_1 = r$, and β a random bit, given g^a, g^b, y_β find β (that is, distinguish between a complete random value and a DH key)
    - Computational Diffie-Hellman problem (CDH) – given g^a, g^b compute g^{ab}
    - Discrete Logarithms (DLog) – given g^a compute a
- The security of the Diffie-Hellman key exchange is equivalent to CDH (and at most as hard as DLog)
- If DLog can be computed Factoring is easy



More on digital signatures: message recovery

- All of the previous signatures worked with the hash of the message, these are usually called signatures with appendix
- Signatures with message recovery also exist, for example with RSA if the message is smaller than the modulus one can sign directly on the message, then recover it from the signature

  - Sign: compute $s = m^d \bmod n$, (note that the message must be smaller than the modulus n)
  - Verify: recover the message from the signature with the help of the public key $m = s^e \bmod n$

- Question: show an existential forgery on the above RSA signing scheme (to avoid such forgeries padding must be used).

Questions?