Ch. 1 - Introduction

Secure software development and web security

R. Absil

Haute École Bruxelles-Brabant
École supérieure d'Informatique

September 23, 2021

Table of contents

Introduction

Before we start

- Since WW2, there is a necessity to protect computer systems
  - Different types of threat
  - Different types of protection

Problem

- Strong security is heavier

Solution

- Analyse each risk and its probability to occur, and adapt your strategy

Types of risks

- Different types of risks
  - Access a restricted resource or service
  - Usurp somebody else identity
  - Access confidential data
  - Falsification
- In this course, *only* these risks are covered
- For each of those risks
  - Several possible attacks
  - Several counter-measures available

"When in doubt, use brute force"

- Many security system rely on hard to solve mathematical problems
  - At least NP-Hard, we assume $P \neq NP$
  - We have no quantum computers

Example

- Big numbers factorisation
- Invert functions
- Combinatorial optimisation
- Stochastic optimisation

- We're interested in this type of constraints
  - Maths left aside

Objectives

1. Availability : make sure a service is available for authorized users
2. Authentication : make sure only authorized users have access to restricted resources and services
3. Non repudiation : make sure a contract between *x* and *y* *really* comes from *x* and is *really y* being concluded with *y*. User can't deny having taken actions
4. Confidentiality : make sure confidential data are confidential for anyone not authorized
5. Integrity : make sure data are left unaltered during transit

Secure software development

What is security

Building Secure Software, J. Viega, G. McGraw

- Security is enforcing a policy that describes rules for accessing resources

- Resource is data, devices, the system itself (*i.e.*, its availability)
- A system is as secure as its weakest component

Secure by design

- Security is not a functionality, not a feature
- Software must be *designed* to be secure
- The attacker chooses the time, place and method
- The defender must protect against all possible attacks
  - Currently know, and yet to be discovered

Kerckhoff's principles

- Introduced in 1883 by Auguste Kerckhoff, in military context

Principles

1. The system must be materially and mathematically unbreakable without the key
2. The system must not require any secret (beyond the key), and must not be compromised should it fall into enemy hands
3. The system must be "easy" to use

- Security *cannot* rely on secrecy

Summary

- Secrecy is a weakness
  - Arguments such as "your system is unsafe because we can see the source code" or "my system is safer than yours because my source code is hidden" are naive, and invalid
- It is hard (if not impossible) to bypass security without the proper codes
- These codes must be easily modified
- The security system must be analysed by experts

What about us ?

- Theses principles can be in particular applied to cryptography, the codes on which it relies being the keys

Summary

- The opponent *knows* the cryptographic system

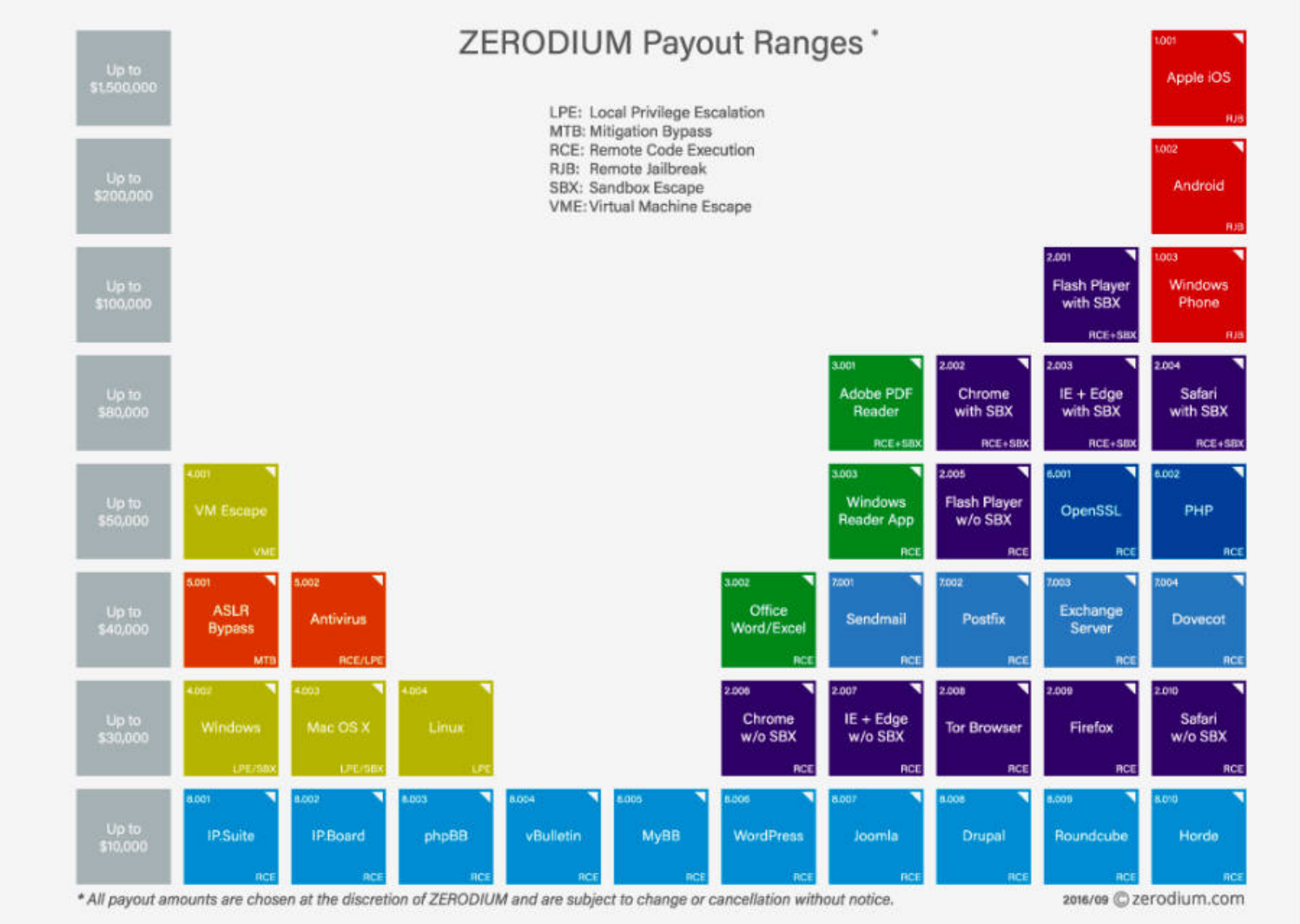
Consequence

- A system's security must rely on the key *only*, not on the system itself

What software should we secure ?

- *Everything*
  - Operating systems
  - Web browser
  - Web / mobile application
  - Desktop application
  - Router software
  - Plugins
  - Videoconf systems

Bounty hunting



Cryptographic hash functions

Characteristics

- Let $f : M \mapsto \mathbb{N}$, f is a cryptographic hash function if
  1. computing $f(x)$ is easy for each $x \in M$
  2. f is resistant to pre-image
     - it is hard to invert f,
  3. f is resistant to second pre-image
     - it is hard to change a message without its hash
  4. f is resistant to collisions
     - it is hard to generate two different messages with the same hash
- It is currently unknow if a function having the first two characteristics actually exists

Good hash functions

- In practice, we require several other characteristics
  - The hash function must be public (no key, no secret)
  - The output must have a fixed size, whatever the size of the input
    - Makes practical uses easy
    - Makes you vulnerable to birthday attacks
    - Based on the Merkle-Damgård construction and one-way compression functions
  - Images are uniformly distributed
    - The hash function "shuffles the input well"

Applications : "encrypted" password

- A user create an account (login,password)
- The system has a hashtable indexed by logins
- We don't want to cipher the table, or each entry separately
- Instead of storing a password, we store its hash
  - Hash function f
- Looking at the table is useless (first pre-image)
- When the system gets an authentication request `(login,password)`, we check if $t\,[login] = hash(password)$
  - If it is, we grant access
- Use of salt
- Multipass hashing

Application : data integrity

Academic example

- A billionaire writes his will
- "When I die, my wealth will be equally shared between my three sons. However, George will receive nothing"
- The father leaves the testament at some notary, and gives the hash to another one

- If Georges corrupts / tortures the notary who has the message, falsification is detected with the hash
- It is also impossible to find a message having the same hash as the one the second notary has

Cipher algorithms

Symmetric ciphers

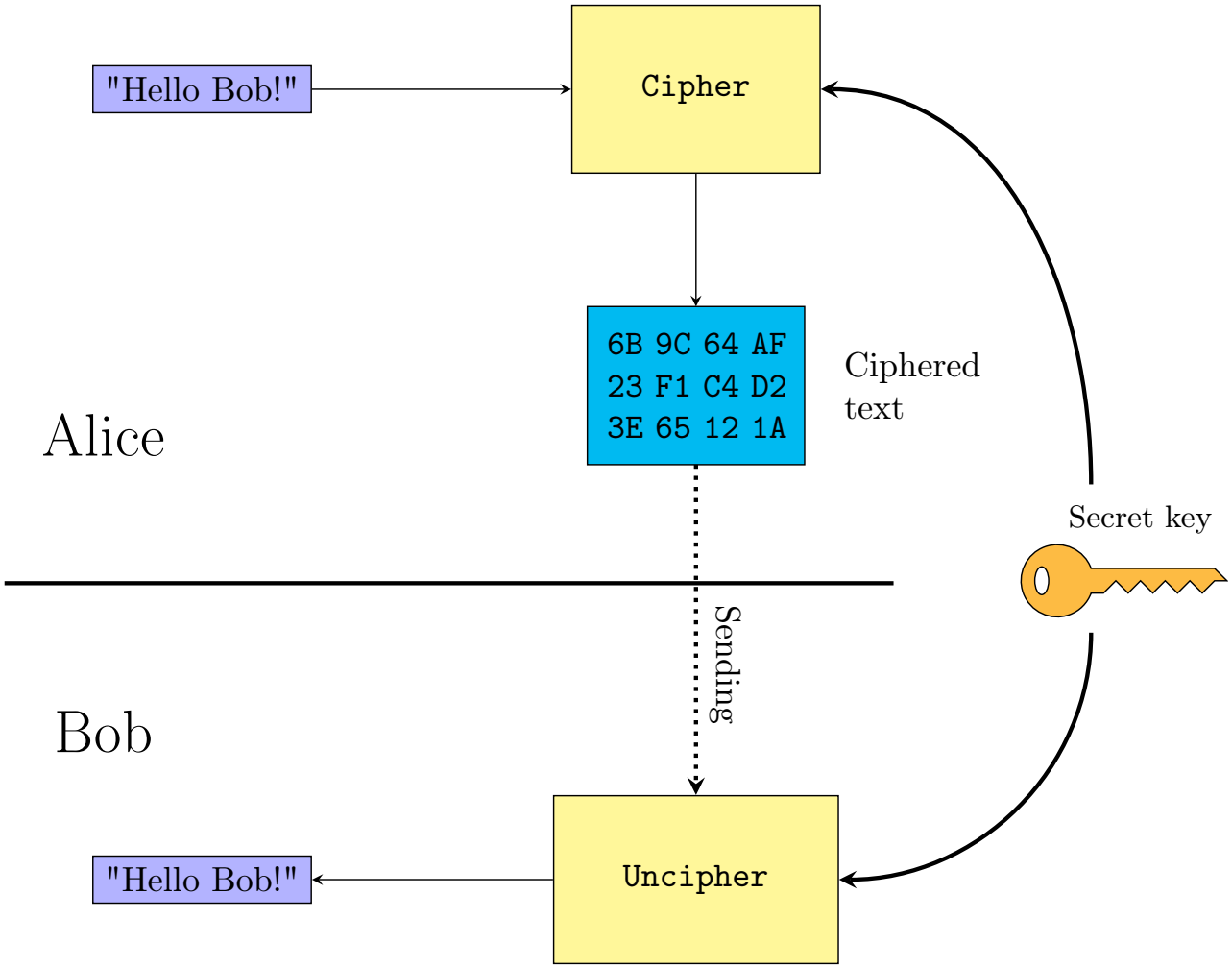
- A single key, solely known by the actors, is used to cipher and uncipher
- Alice ciphers her text with the key and sends it to Bob
- Bob uses the key to decipher the message

Problems

- Safety and key exchange
- For n actors, $\frac{n(n-1)}{2}$ keys are required

- Advantage : *fast*

Illustration



Key exchange

Problem

- A problem arises when we transmit the key : it *cannot* be intercepted
- Either we use a secure channel (hard in practice), or meet the actor one face-to-face

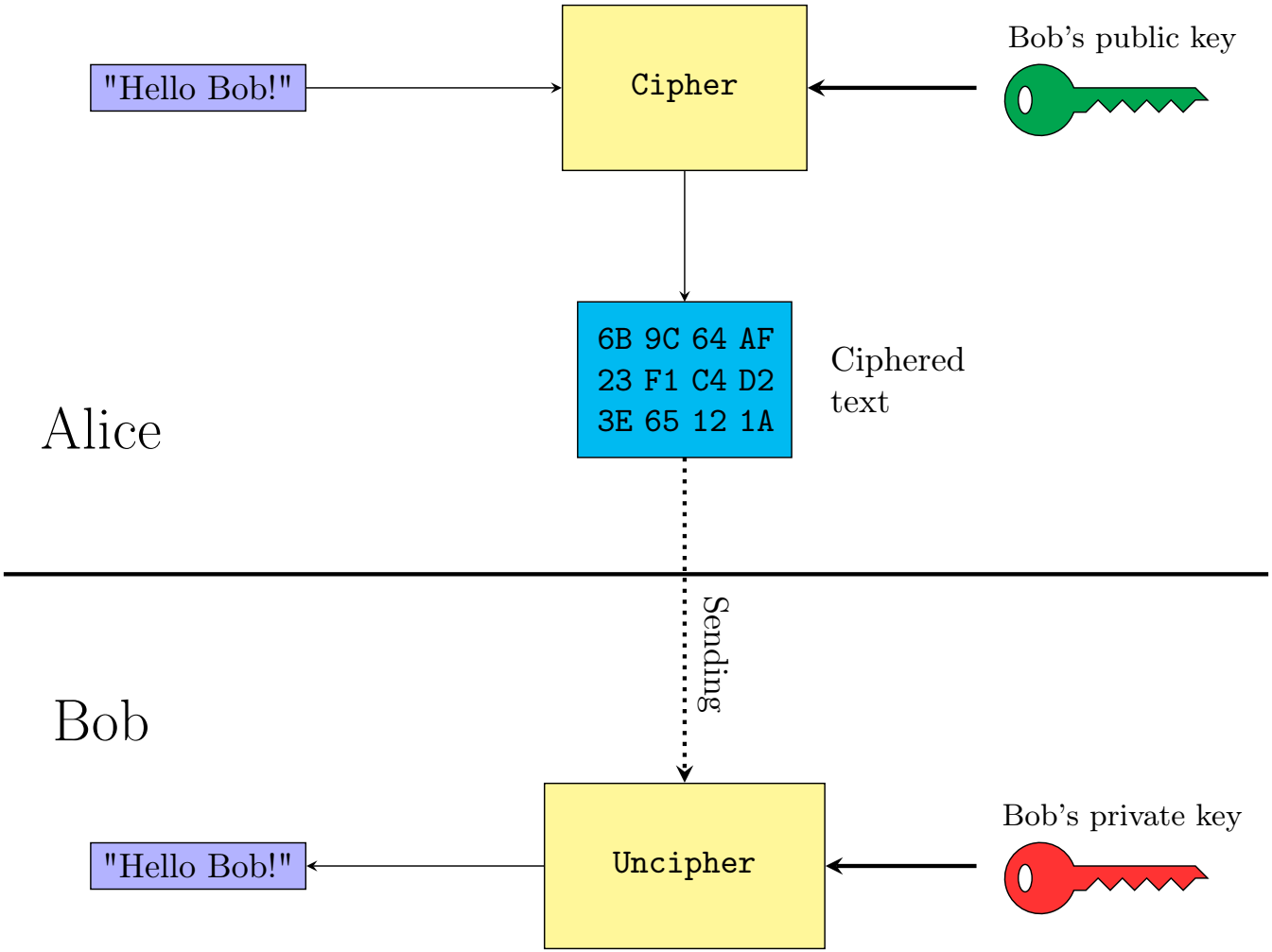
Solution

- Public key cryptography cleverly solves the problem
- *Key wrapping* technique (fast)

Asymmetric ciphers

- A pair of keys per actor : one to cipher, one to decipher
  - In the context of encoding data
- Alice ciphers her text with Bob's *public* key and sends it
- Bob deciphers the message with his own *private* key
- Since Bob is the only one having that private key, only him can read what has been sent
- Problem : public key first exchange
- Problem : time

Illustration



Key wrapping

- Alice wants to talk to Bob
- She generates a random symmetric key (session key)
- She ciphers the session key with Bob's public key, and sends it
- Bob receives the session key that he deciphers with his private key
- Alice and Bob can now cipher their messages with the session key

Avantages

- Fast, since asymmetric cipher is used on the secret key only
- Secure, *if* Alice already has Bob's public key

Illustration : wrapping

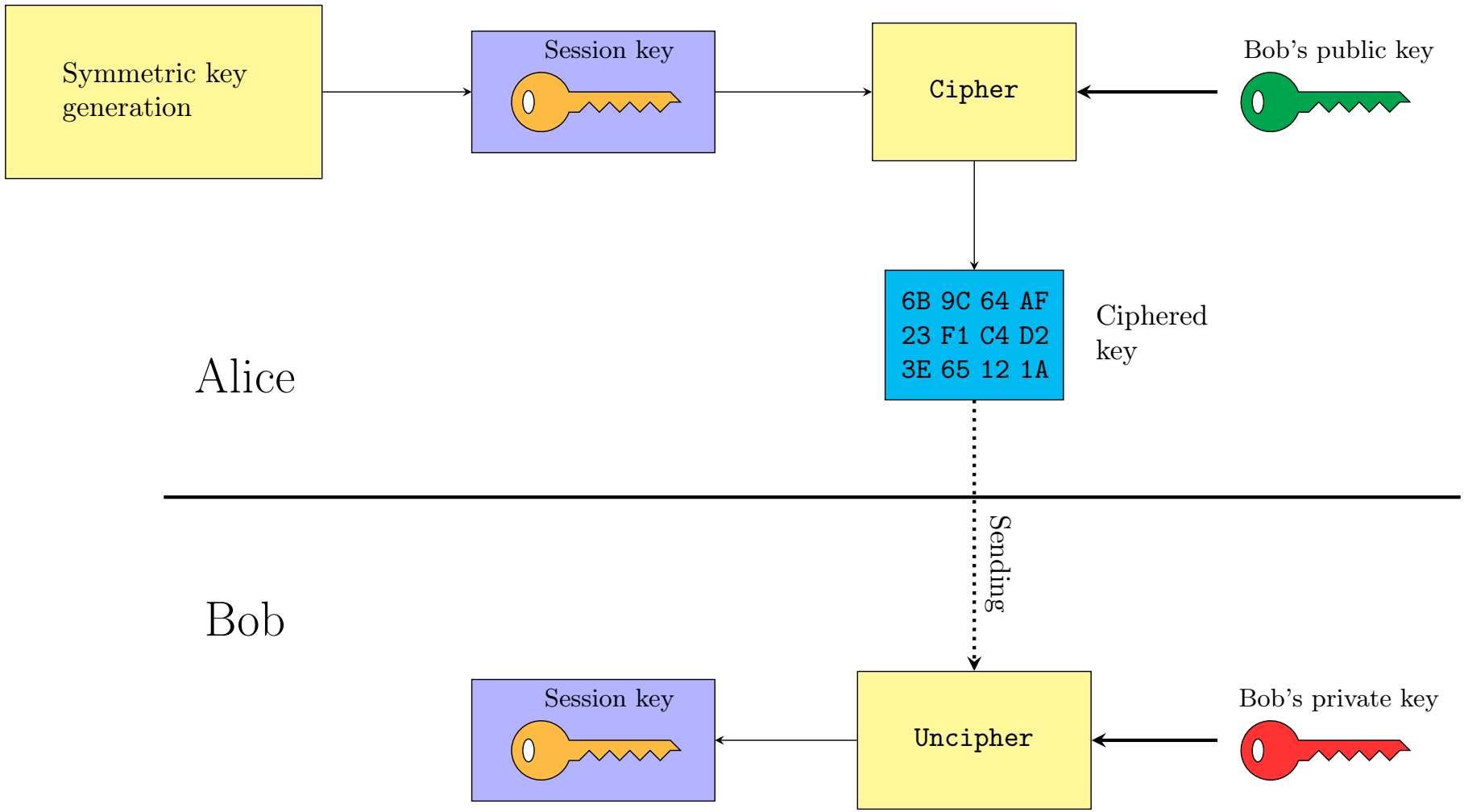
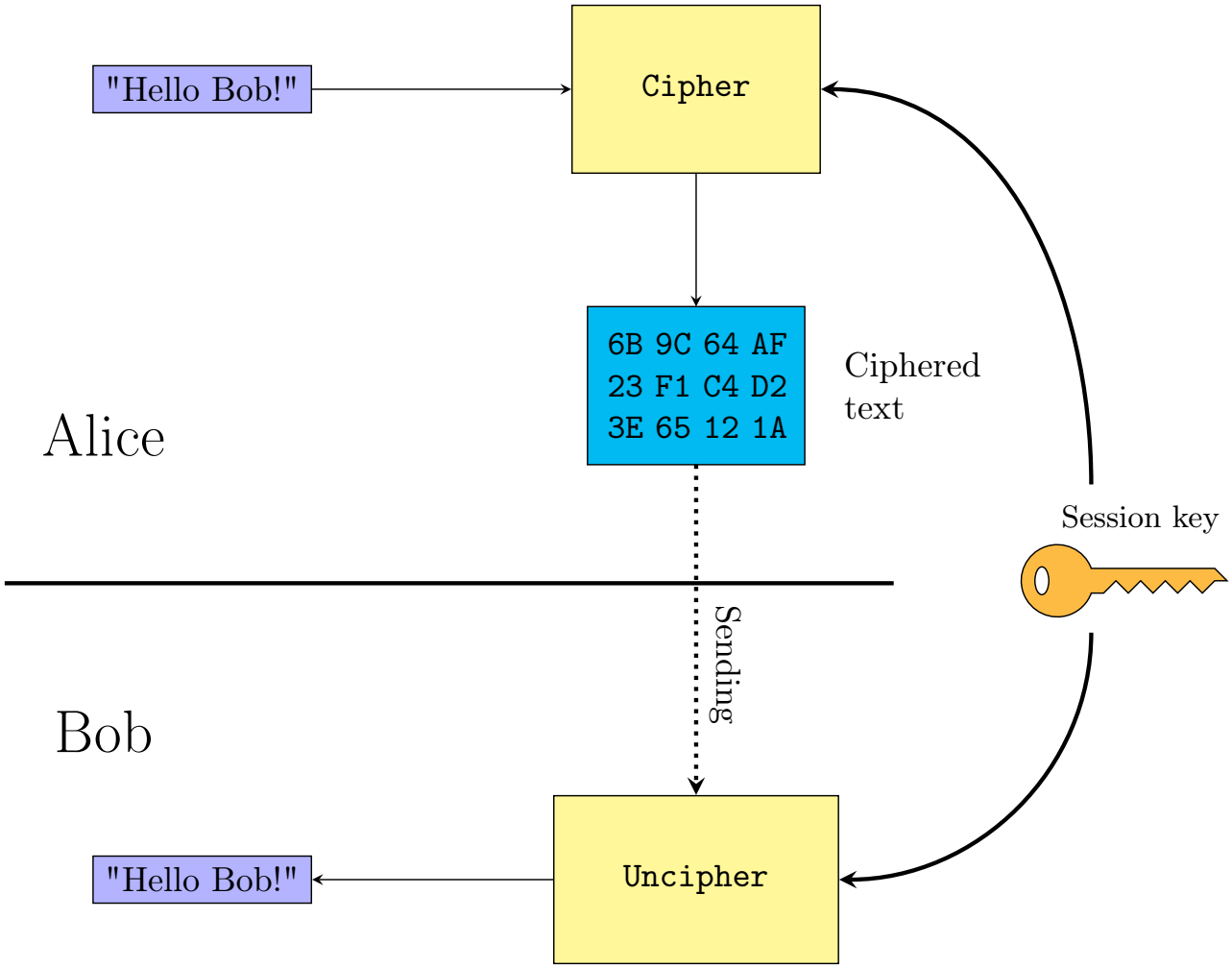


Illustration : cipher



Constraint

Quality

- It must be hard to deduce part of the plain text from the ciphered text
- It must be hard to deduce the private key from the public one

- Obvious applications to confidentiality and privacy

Numerical signature and certification authority

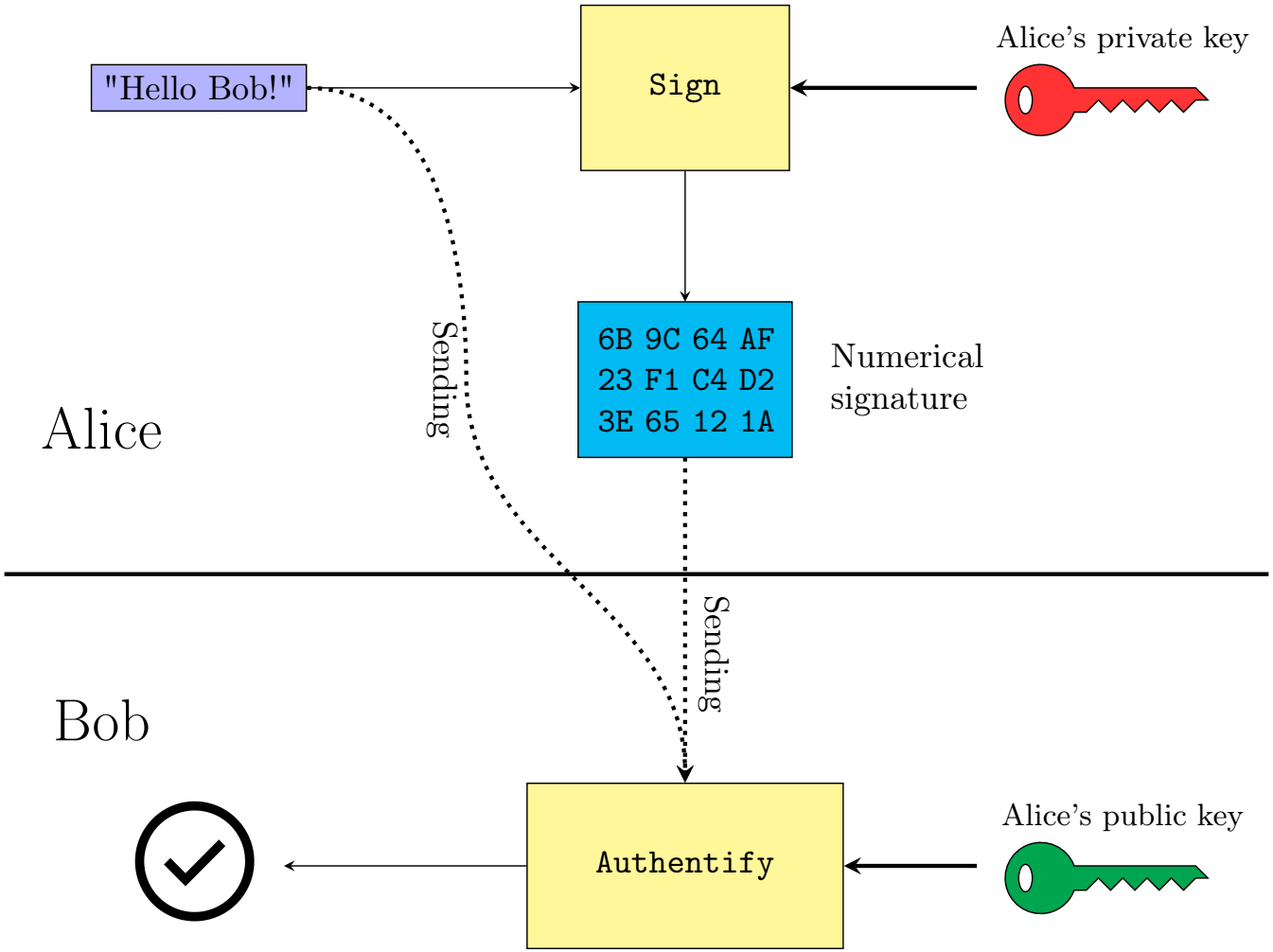
Numerical signature principle

- It is difficult to create a secure channel
- Consequently, we use *numerical signatures*.
- Allows to "authenticate" messages
- When a message is authenticated, we know that
  1. the content was left unaltered
  2. it came from the expected expediter
- Implemented with public key cryptography

Authenticated protocol example

1. Alice want to send a message to Bob
2. Alice signs her message with her private key, and send the "signed message" to Bob
3. Bob uses Alice's public key and use it to verify the signature
4. If successful, Bow knows that
   - the message was left unaltered
   - the message was signed with the private key corresponding to the public key he used

Illustration



Signature : problems

Problem

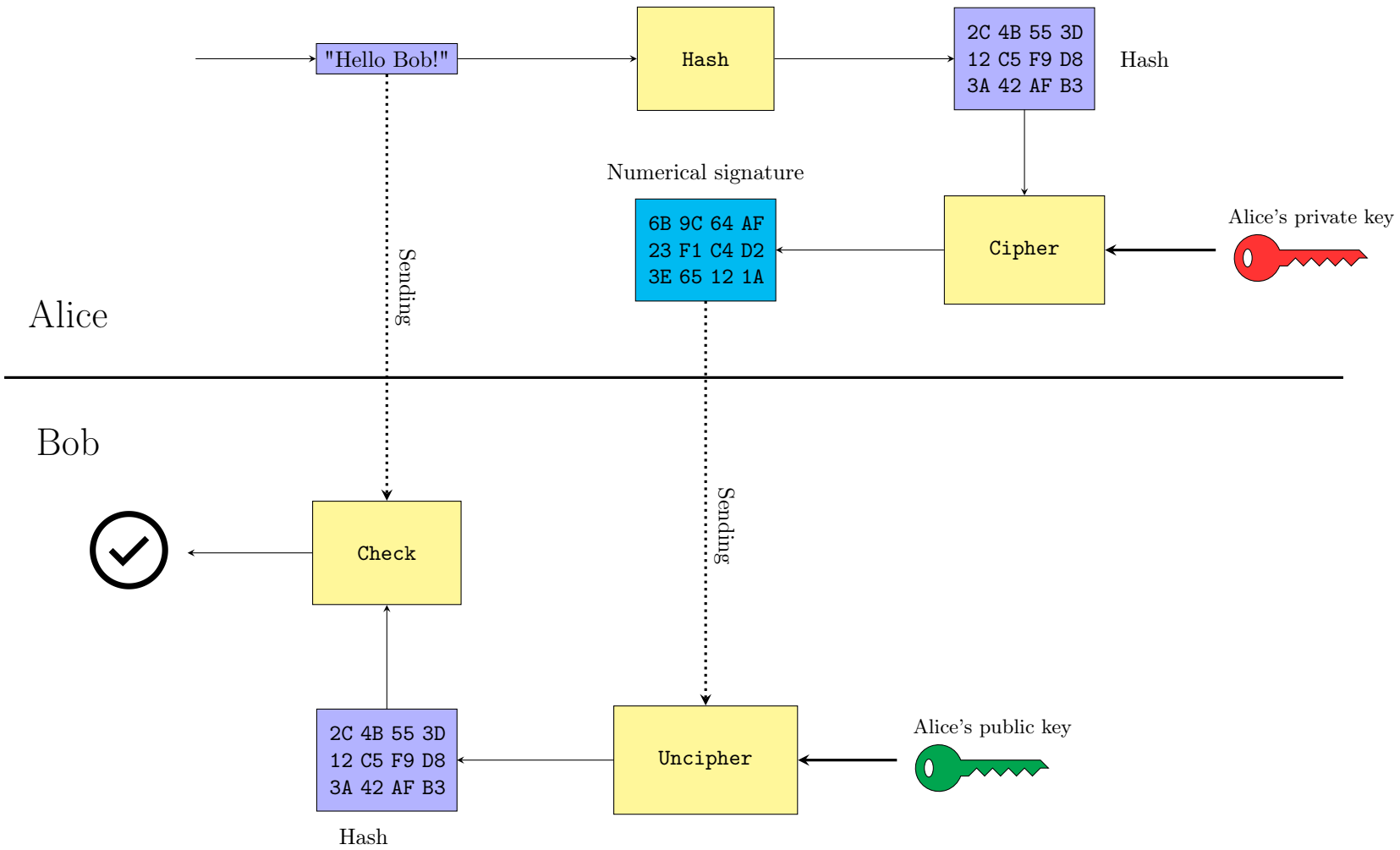
- We want to sign arbitrary long messages
- We want a signature of reasonable size
- We want a signature that is hard to falsify

Solution

- Use cryptographic hash functions

- We get a hash, and we cipher it with our own *private* key

Detailed illustration



Consequences of hash functions

- Second pre-image attack
  - From a signed message, find another one with the same signature
  - The signature is valid
- Collision attack
  - From a collision c of a message m, and the signature of m, we have a signature for c.
- We want it to be hard to be able to sign without the private key of someone
- We want want it to be hard to recover the private key
  - from the public key,
  - from intercepted signatures.

The need of a third party trusted intermediate

- Assume Alice receives a message from Oscar, a *stranger*, pretending to work for *iDiots*, a famous company.
- Oscar even gives a copy of its public key so Alice can authenticate the message
  - Alice can check that the message was signed with the corresponding private key
  - Alice can check that the message was left unaltered

Problem

- We still don't know who wrote the message
- Anyone could have generated a key pair and signed the message

- Finding out the identity of a key owner is called the *identification problem*.

Principles

- Alice and Oscar have a common relation that both of them trust
- Oscar meets this intermediate and gives him his public key
- This trusted intermediates recognizes Oscar and guarantees his identity
- This trusted intermediates uses his own private key to sign Oscar's public key, and sends it to Alice
  - Alice can check the key was left unaltered
  - Alice can check the key was sent by he trusted intermediate
- Alice can know fully authenticate the message of Oscar
- In practice : trusted intermediate = *certification authority*

Examples of certification authorities

Nom	Use	Market share
Comodo	6.1%	37.2%
Symantec	5%	30.2%
GoDaddy	2.2%	13.3%
GlobalSign	1.7%	10.4%
DigiCert	0.5%	3.1%
StartCom	0.1%	2.2%
Entrust	0.1%	0.8%
Verizon	0.1%	0.7%
Trustwave	0.1%	0.6%
Secom	0.1%	0.6%

- Source : *Usage of SSL certificate authorities for websites*, http://w3techs.com/technologies/overview/ssl_certificate/all - 22/11/2015.

Certificates and PKI

Terminology

- *(Public key) certificate* : electronic document used to prove the ownership of a public key
- *Public key infrastructure (PKI)* : set of hardware, software, people, policies and protocols needed to create, manage, distribute, store and revoke certificates

- Goal of a PKI
  1. make information transfer easier in network services (e-trade, etc.),
  2. link public keys to their owner with certification authorities

X.509 Certificates

- PKI standard
- Specifies, among other things, the format of public key certificates, their revocation, validation schemes, etc.
- Structure (non exhaustive) :
  - Version number
  - Serial number
  - Client identity (name, address, company, etc.)
  - Expiration date
  - Information about the client's public key
    - Algorithm and specifications
    - Public key
  - Signature algorithm
  - Certificate signature

Threshold cryptography

Threshold cryptography

- Idea : we want to share a secret
  - Cryptographic key for ciphering or signing
  - Access codes
- Motivation : allow service if at least x people agree to cooperate-
  - Bank safe, missile codes, etc.

Overview

Basic idea

- We want to share a secret (a key) between several people
- We want to be able to recover the secret if *enough* of these people cooperate
- If an insufficient number of people is gathered, the secret is safe

- How to proceed ?!
  - Several (mathematical) techniques
  - Here : polynomial interpolation
- Motivation
  - Fight against corruption of actors
  - Force several people to cooperate to make important decisions

(k, n) threshold cryptography

Definition

- A cryptographic system is called a *threshold cryptographic system* if several people (more than the threshold) need to cooperate in the protocol used to (de)cipher a message.
- Such a system is a (k, n)-threshold if at least k people among n must cooperate to (de)cipher the message

- Idea : the private key is shared among actors
- Independently discovered by Shamir (polynomials) and Blakley (hyperplanes) in 1979

Key sharing (1/2)

Safe sharing

- *Safe sharing* if someone with less than k pieces as no more information that someone with no pieces.

- Example : secret `Password` to share
  1. People 1 : `Pa......`
  2. People 2 : `..ss....`
  3. People 3 : `....wo..`
  4. People 4 : `......rd`
- Someone with no pieces must guess among $26^8 \simeq 208$ billions possibilities
- Someone with one piece must guess among $26^6 \simeq 308$ millions possibilities
- Unsafe sharing

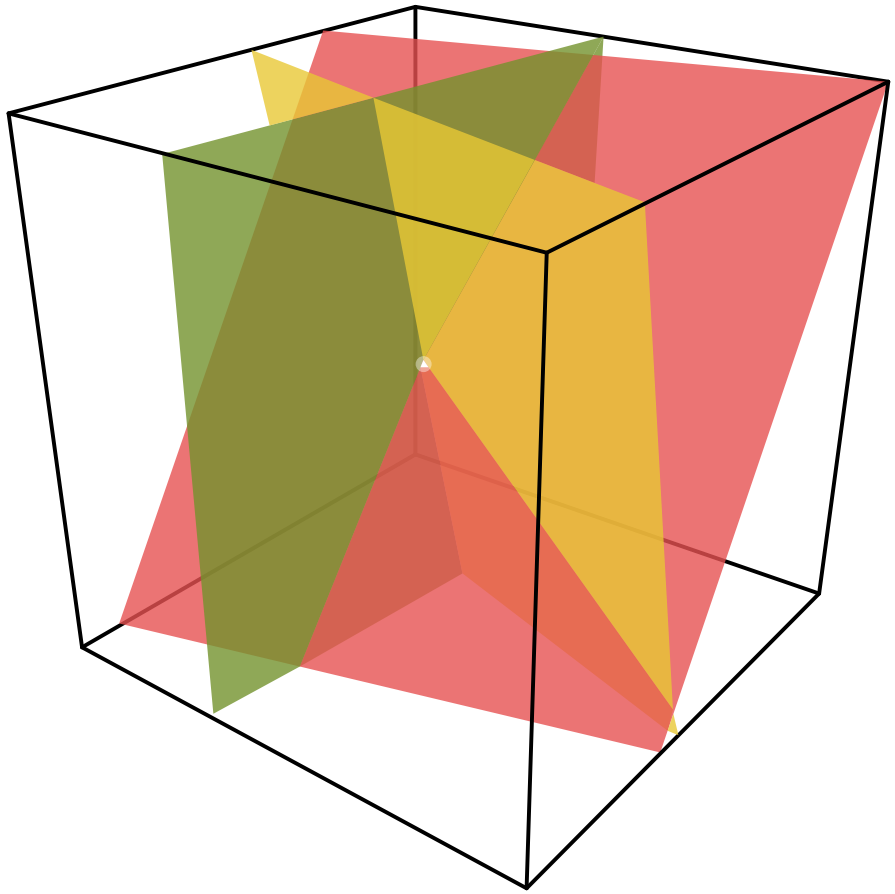
Key sharing (2/2)

- A secret is iteratively ciphered with k public keys
- The k corresponding private keys are distributed among $n = k$ actors
  - Someone can remove the first level of cipher
  - If two people cooperate, they can remove two levels
  - Unless k persons cooperate, the secret is still completely confidential
- Bypassing the system without k keys is reduced to decipher datas without a key
  - Not very practical, but safe

Problem

- How to proceed so that $k < n$ people are able to decipher ?

Example with few people (Blakley)



Polynomial interpolation

- In the case of Shamir secret sharing, we rely on polynomial interpolation
- To share a secret among k people,
  1. we work with a polynomial of degree $(k-1)$
  2. The key is one of the coefficient (arbitrary)
  3. Other coefficients are generated at random
  4. We generate n distinct points of the polynomial and give one of them to each actor
- If k actors cooperate, we have k points of a polynomial of degree $k-1$
  1. We build it with polynomial interpolation
  2. We recover the key

In practice

- We have $n+1$ points $p_i = (x_i, y_i)$
- We want to compute the coefficients of a polynomial

$$\begin{aligned} P(x) &= \sum_{i=0}^{n} a_i x^i \\ &= a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 \end{aligned}$$

- Substituting x in this equation and $P(x)$ by their values, we get a system with $n+1$ equations and $n+1$ unknown (the coefficients)

$$\begin{bmatrix} x_0^n & x_0^{n-1} & x_0^{n-2} & \dots & x_0 & 1 \\ x_1^n & x_1^{n-1} & x_1^{n-2} & \dots & x_1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n^n & x_n^{n-1} & x_n^{n-2} & \dots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

System solving in practice

- We must solve this system in the variables a_i
- Mimic the "manual" method
  - Make the matrix triangular
- Gaussian method
  - Running time : $\mathcal{O}(n^3)$
  - High conditioning
    - Subject to numerical errors : numerical instability
    - How a controlled error on inputs alters the output ?
- Other methods
  - Rewrite the polynomial
    - Lagrangian interpolation
    - Newtonian interpolation
  - Divided differences

Other concepts

Avalanche effect (1/2)

- Desired effect of cipher algorithms and cryptographic hash functions
- Idea : if we slightly change the input, the output changes greatly
  - Flipping a bit in the plain text will strongly change the ciphered text
  - Flipping a bit in a message strongly changes its hash
- If this effect is not strong enough, the randomness of the output is low, and cryptanalyst can deduce clues
  - related to the plain text or the private key, from the ciphered text,
  - related a message from its hash
- Butterfly effect

Avalanche effect (2/2)

- Two avalanche criterions

Strict avalanche criterion

- If a single bit is flipped in the input, each bit of the output has 50% chances to change

Bits independence criterion

- Two bits i and j of the output independently change when a bit k of the input is flipped, for each i, j, k.

Padding

- No cryptographic algorithm processes all its input in one time
- The input is arbitrary large
- Use of a *padding scheme*
- Idea : cut the input into blocks of equal size, and process each block separately
  - Hash : iteratively input blocks to a one-way compression function (Merkle-Damgård construction)
  - Cipher : each block is ciphered iteratively using a chaining method (e.g., CBC)
  - Signature : each block is hashed and ciphered
- The size of a block is algorithm dependent
  - Java : `doFinal` errors