




# 9<sup>th</sup> International Cybersecurity Forum

24<sup>TH</sup> & 25<sup>TH</sup>  
OF JANUARY 2017

LILLE  
GRAND PALAIS

**TLS : analyse des vulnérabilités récentes et  
introduction à TLS 1.3 /**  
**TLS : analysis of recent vulnerabilities and  
introduction to TLS 1.3**



**Olivier Levillain @ ANSSI**

Smarter security for **future technologies**

# INTERVENANTS/ SPEAKERS

Olivier Levillain, Responsable du centre de formation, ANSSI



**24 & 25** | **LILLE**  
JANVIER 2017 | GRAND PALAIS

# Qui suis-je ?

Olivier Levillain (@pictyeye)

- ▶ stage de DEA en cryptographie sur une fonction de hachage
- ▶ membre du laboratoire « système » de l'ANSSI (2007-2012)
- ▶ responsable du laboratoire « réseau » de l'ANSSI (2012-2015)
- ▶ responsable du CFSSI, centre de formation de l'ANSSI (2015-)

Recherche

- ▶ participation aux travaux sur les mécanismes bas-niveau x86
- ▶ travaux sur SSL/TLS (thèse entre 2011 et 2016)
- ▶ études sur les langages depuis 2007
- ▶ travaux sur les *parsers*

# Master class sur TLS @ FIC 2017

Description rapide de SSL/TLS

Plus de vingt ans de vulnérabilités

- Handshake Protocol

- Record Protocol

- Erreurs d'implémentation

TLS 1.3 : un nouvel espoir ?

# Master class sur TLS @ FIC 2017

Description rapide de SSL/TLS

Plus de vingt ans de vulnérabilités

- Handshake Protocol

- Record Protocol

- Erreurs d'implémentation

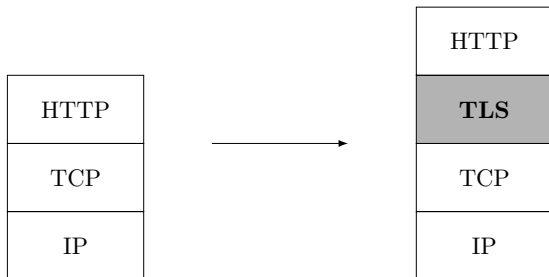
TLS 1.3 : un nouvel espoir ?

## Objectifs du protocole

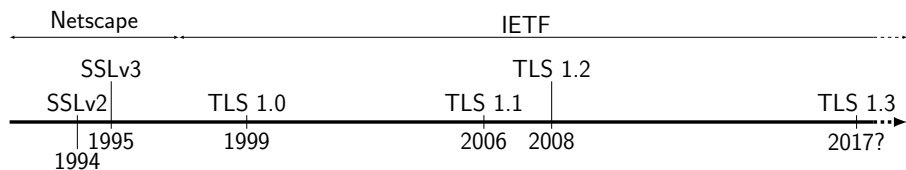
SSL/TLS, un protocole de sécurité garantissant

- ▶ l'authentification du serveur (et optionnellement du client)
- ▶ la protection en confidentialité et en intégrité des données

SSL/TLS, une couche transport

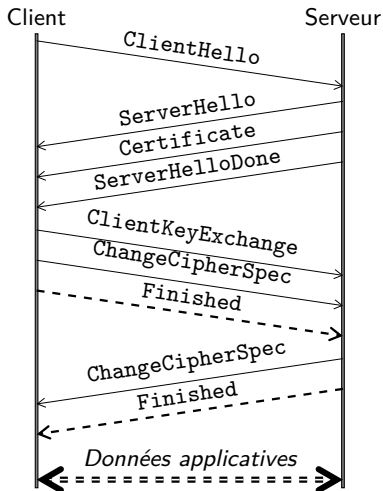


# SSL ou TLS



Un protocole âgé de plus de 20 ans

- ▶ conception originale par Netscape pour créer HTTPS (SSL)
- ▶ maintenance par l'IETF depuis 1999 (TLS)
- ▶ utilisation très large aujourd'hui
  - ▶ protection de nombreux protocoles Internet
  - ▶ mise en place de tunnels VPN
  - ▶ authentification dans un échange EAP



## Deux phases

- ▶ Handshake Protocol
  - ▶ négociation des algorithmes
  - ▶ authentification du serveur
  - ▶ échange de clé
- ▶ Record Protocol
  - ▶ échanges de données

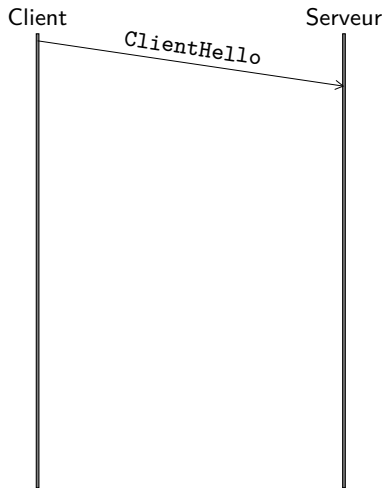


Client



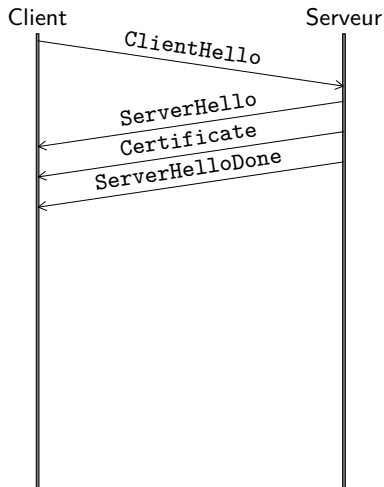
Serveur





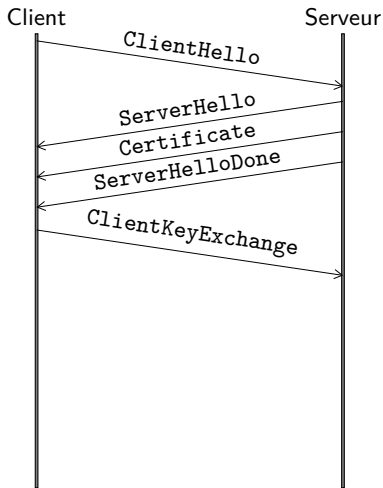
### Propositions du client

- ▶ version max. du protocole
- ▶ suites cryptographiques
  - ▶ AU=Authentification
  - ▶ KX=Échange de clé
  - ▶ ENC=Chiffrement
  - ▶ MAC=Intégrité
- ▶ extensions



### Réponse du serveur

- ▶ choix des paramètres
  - ▶ *on suppose ici  $KX=RSA$*
- ▶ présentation de la chaîne de certificats

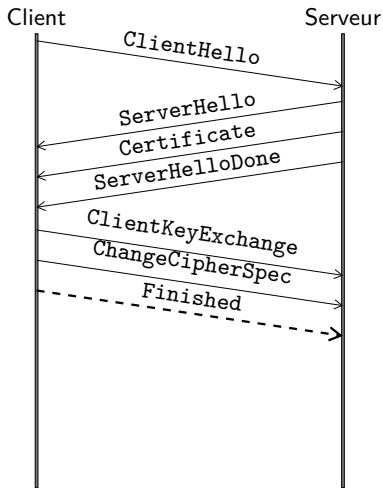


## Échange de clé par chiffrement RSA

- ▶ le client choisit un aléa
- ▶ et le chiffre avec la clé publique du serveur

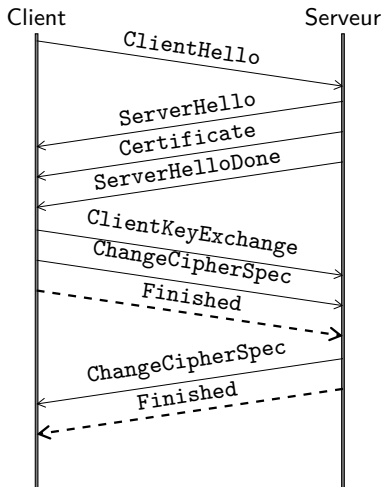
## Après ce message

- ▶ les paramètres sont choisis
- ▶ client et serveur partagent un secret (l'aléa choisi par le client)



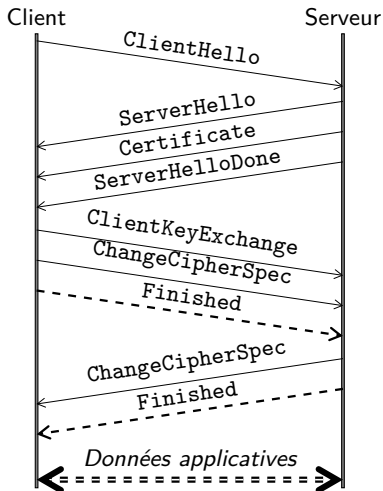
Activation de la protection

- ▶ du client vers le serveur



### Activation de la protection

- ▶ du client vers le serveur
- ▶ du serveur vers le client



Le tunnel est prêt

- ▶ les données sont chiffrées et protégées en intégrité

# SSL/TLS en chiffres

## 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
- ▶ combien d'implémentations maison ?



## Les implémentations maison

Que répond un serveur si vous lui proposez les suites crypto **AES128-SHA** et **ECDH-ECDSA-AES128-SHA** ?

## Les implémentations maison

Que répond un serveur si vous lui proposez les suites crypto **AES128-SHA** et **ECDH-ECDSA-AES128-SHA** ?

A **AES128-SHA**

## Les implémentations maison

Que répond un serveur si vous lui proposez les suites crypto **AES128-SHA** et **ECDH-ECDSA-AES128-SHA** ?

A **AES128-SHA**

B **ECDH-ECDSA-AES128-SHA**

## Les implémentations maison

Que répond un serveur si vous lui proposez les suites crypto **AES128-SHA** et **ECDH-ECDSA-AES128-SHA** ?

- A **AES128-SHA**
- B **ECDH-ECDSA-AES128-SHA**
- C une alerte

## Les implémentations maison

Que répond un serveur si vous lui proposez les suites crypto **AES128-SHA** et **ECDH-ECDSA-AES128-SHA** ?

- A **AES128-SHA**
- B **ECDH-ECDSA-AES128-SHA**
- C une alerte
- D la réponse D (**RC4\_MD5**)

## Les implémentations maison

Que répond un serveur si vous lui proposez les suites crypto `AES128-SHA` et `ECDH-ECDSA-AES128-SHA` ?

- A `AES128-SHA` (0x002f)
- B `ECDH-ECDSA-AES128-SHA` (0xc005)
- C une alerte
- D la réponse D (`RC4_MD5`) (0x0005)

Le pire, c'est qu'on peut l'expliquer :

- ▶ une suite cryptographique est un entier sur 16 bits
- ▶ pendant longtemps, les seules valeurs utilisées étaient 00 XX
- ▶ du coup, pourquoi considérer l'octet de poids fort ?

# Master class sur TLS @ FIC 2017

Description rapide de SSL/TLS

**Plus de vingt ans de vulnérabilités**

- Handshake Protocol

- Record Protocol

- Erreurs d'implémentation

TLS 1.3 : un nouvel espoir ?

## Classification des vulnérabilités

Dès 1995, beaucoup de failles et d'attaques ont été publiées

Analysons les failles selon les catégories suivantes :

- ▶ vulnérabilités affectant le Handshake Protocol
- ▶ attaques contre le Record Protocol
- ▶ erreurs d'implémentation
- ▶ *problèmes liés aux certificats*



# Master class sur TLS @ FIC 2017

Description rapide de SSL/TLS

Plus de vingt ans de vulnérabilités

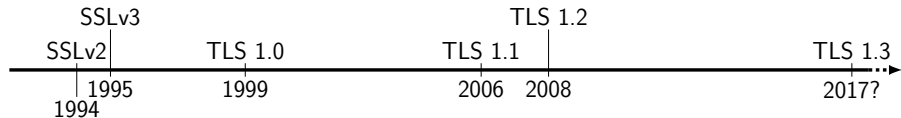
- Handshake Protocol

- Record Protocol

- Erreurs d'implémentation

TLS 1.3 : un nouvel espoir ?

# Vulnérabilités affectant le Handshake Protocol



## Paramètres crypto faibles



## Failles dans la spécification



## Attaques cross-protocoles

Confusion  
RSA/DHE  
[WS96]

Confusion  
DHE/ECDHE  
[MVVP12]

FREAK  
[BBD+15]

# Bleichenbacher

## RSA PKCS#1 v1.5

- ▶ le chiffrement RSA repose sur un schéma de bourrage
- ▶ comment traiter un *padding* invalide lors du déchiffrement ?

# Bleichenbacher

## RSA PKCS#1 v1.5

- ▶ le chiffrement RSA repose sur un schéma de bourrage
- ▶ comment traiter un *padding* invalide lors du déchiffrement ?

## L'attaque de Bleichenbacher (1998)

- ▶ le principe : envoyer un texte chiffré altéré
- ▶ si l'attaquant peut distinguer un *padding* valide d'un *padding* invalide, il obtient de l'information sur le clair
- ▶ application à TLS : l'attaque *Million Message Attack*

# Master class sur TLS @ FIC 2017

Description rapide de SSL/TLS

Plus de vingt ans de vulnérabilités

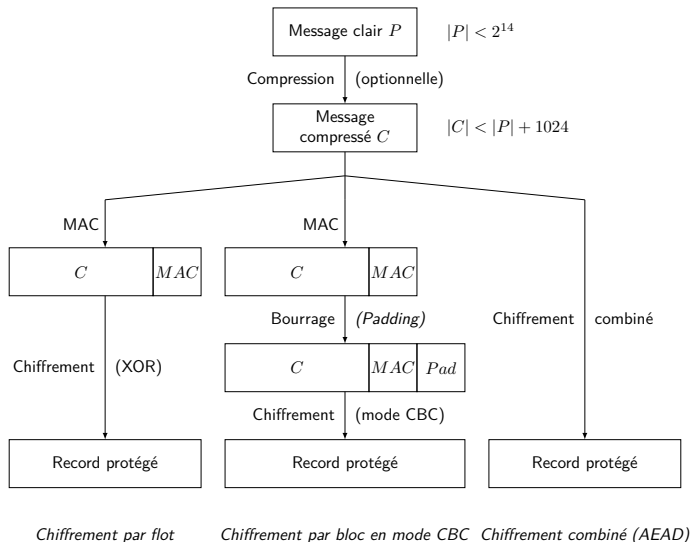
- Handshake Protocol

- Record Protocol

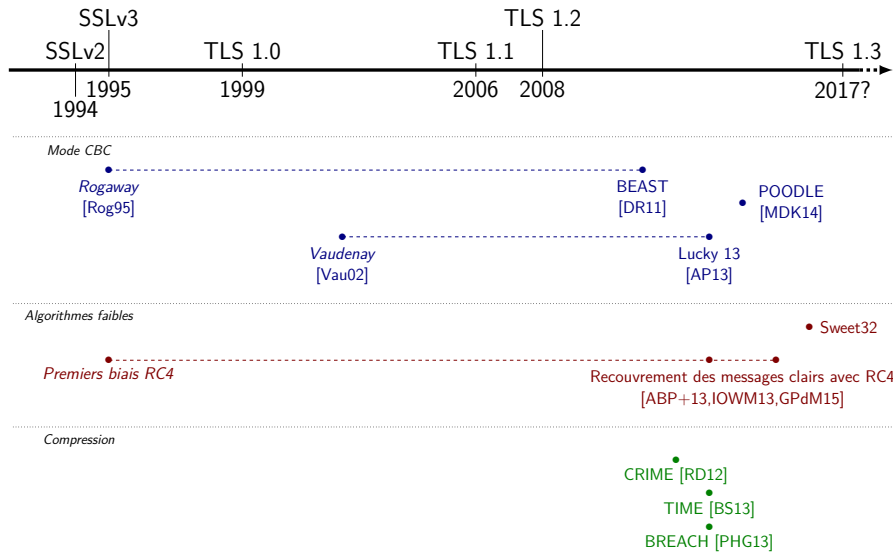
- Erreurs d'implémentation

TLS 1.3 : un nouvel espoir ?

# Description du Record Protocol



# Attaques contre le Record Protocol



# Réflexions autour du mode CBC

`https://www.openssl.org/~bodo/tls-cbc.txt`

Bodo Möller

Le document indique que



# Réflexions autour du mode CBC

<https://www.openssl.org/~bodo/tls-cbc.txt>

Bodo Möller

Le document indique que

- ▶ le mode CBC dans TLS favorise les oracles de padding
- ▶ l'utilisation d'un IV implicite peut mener à des attaques
- ▶ avec SSLv3, les octets de padding sont mal spécifiés, ce qui amplifie les oracles

# Réflexions autour du mode CBC

`https://www.openssl.org/~bodo/tls-cbc.txt`

Bodo Möller (2004-05-20)

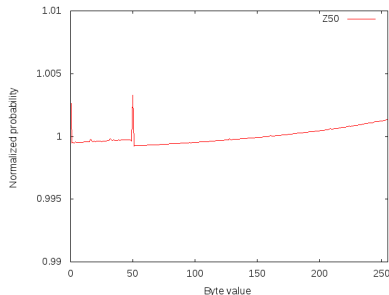
Le document indique que

- ▶ le mode CBC dans TLS favorise les oracles de padding
- ▶ l'utilisation d'un IV implicite peut mener à des attaques
- ▶ avec SSLv3, les octets de padding sont mal spécifiés, ce qui amplifie les oracles
- ▶ ... il décrivait Lucky 13, BEAST et POODLE

# Biais statistiques RC4

Hypothèses :

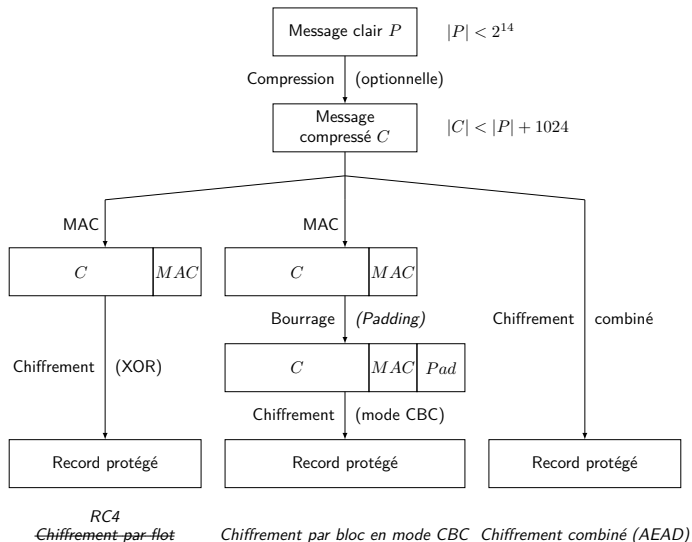
- ▶ la connexion utilise RC4
- ▶ l'attaquant peut observer les paquets chiffrés
- ▶ l'attaquant peut générer des connexions multiples contenant le secret



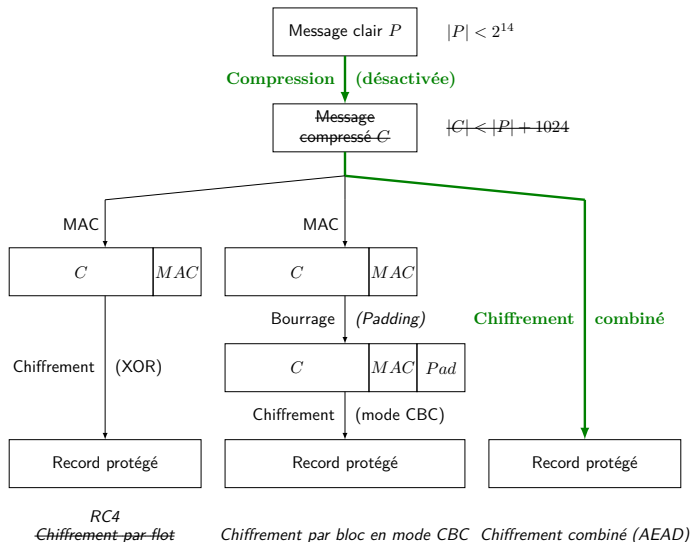
Contremesures proposées

- ▶ utiliser les suites AEAD (et donc TLS 1.2)
- ▶ utiliser le mode CBC
- ▶ utiliser un autre algorithme de chiffrement par flot
- ▶ randomiser la position du secret

# Record Protocol : la solution long-terme



# Record Protocol : la solution long-terme



# Master class sur TLS @ FIC 2017

Description rapide de SSL/TLS

Plus de vingt ans de vulnérabilités

Handshake Protocol

Record Protocol

**Erreurs d'implémentation**

TLS 1.3 : un nouvel espoir ?

## 2014 : une année dure pour TLS

Toutes les piles TLS majeures touchées par une vulnérabilité critique

- ▶ février : `goto fail` (Apple)
- ▶ février : `goto fail` (GnuTLS)
- ▶ avril : *Heartbleed* (OpenSSL)
- ▶ juin : *Early CCS* (OpenSSL)
- ▶ septembre : Universal signature forgery (Berserk ?) (Mozilla NSS)
- ▶ septembre : Universal signature forgery (Berserk ?) (CyaSSL)
- ▶ septembre : Universal signature forgery (Berserk ?) (mbedTLS)
- ▶ novembre : exécution de code arbitraire (MS SChannel)

## 2014 : une année dure pour TLS

Toutes les piles TLS majeures touchées par une vulnérabilité critique

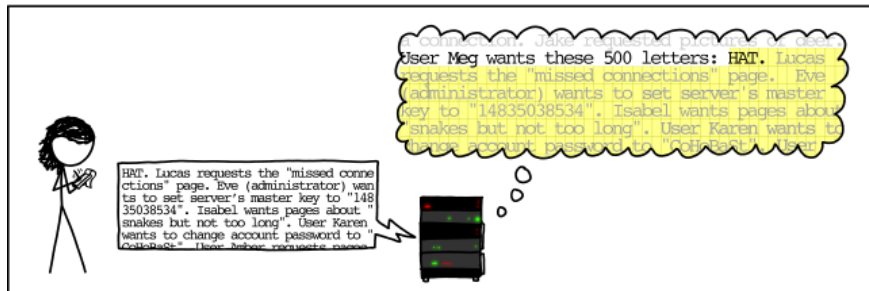
- ▶ février : `goto fail` (Apple)
- ▶ février : `goto fail` (GnuTLS)
- ▶ avril : *Heartbleed* (OpenSSL)
- ▶ juin : *Early CCS* (OpenSSL)
- ▶ septembre : Universal signature forgery (Berserk ?) (Mozilla NSS)
- ▶ septembre : Universal signature forgery (Berserk ?) (CyaSSL)
- ▶ septembre : Universal signature forgery (Berserk ?) (mbedTLS)
- ▶ novembre : exécution de code arbitraire (MS SChannel)

### Quizz

- ▶ Que s'est-il aussi passé début avril 2014 (en plus de Heartbleed) ?
- ▶ Même question le 24 septembre 2014 (publication de Berserk) ?



## Erreurs classiques (1/3) : Heartbleed



Source : <http://xkcd.com/1354>

```
+      /* Read type and payload length first */
+      if (1 + 2 + 16 > s->s3->rrec.length)
+          return 0; /* silently discard */
```

## Erreurs classiques (2/3) : WinShock

- ▶ L'authentification client utilisant des certificats avec des courbes elliptiques repose sur deux messages
- ▶ Le message `Certificate`, qui contient la chaîne de certification
  - ▶ il contient la courbe utilisée
  - ▶ en particulier, il indique la taille  $L$  du corps sous-jacent
- ▶ `CertificateVerify`, qui contient une signature couvrant les messages précédents
  - ▶ la signature contient les coordonnées d'un point, d'une taille  $l$
  - ▶ supposons que `SChannel` copie  $l$  octets dans une zone allouée pour  $L$  octets, sans se poser de question...

## Erreurs classiques (2/3) : WinShock

- ▶ L'authentification client utilisant des certificats avec des courbes elliptiques repose sur deux messages
- ▶ Le message `Certificate`, qui contient la chaîne de certification
  - ▶ il contient la courbe utilisée
  - ▶ en particulier, il indique la taille  $L$  du corps sous-jacent
- ▶ `CertificateVerify`, qui contient une signature couvrant les messages précédents
  - ▶ la signature contient les coordonnées d'un point, d'une taille  $I$
  - ▶ supposons que `SChannel` copie  $I$  octets dans une zone allouée pour  $L$  octets, sans se poser de question...

Sauf que l'authentification client par certificat est rarement utilisée

## Erreurs classiques (2/3) : WinShock

- ▶ L'authentification client utilisant des certificats avec des courbes elliptiques repose sur deux messages
- ▶ Le message `Certificate`, qui contient la chaîne de certification
  - ▶ il contient la courbe utilisée
  - ▶ en particulier, il indique la taille  $L$  du corps sous-jacent
- ▶ `CertificateVerify`, qui contient une signature couvrant les messages précédents
  - ▶ la signature contient les coordonnées d'un point, d'une taille  $I$
  - ▶ supposons que `SChannel` copie  $I$  octets dans une zone allouée pour  $L$  octets, sans se poser de question...

Sauf que l'authentification client par certificat est rarement utilisée

*Teaser* : tous les serveurs vulnérables étaient pourtant exploitables

## Erreurs classiques (3/3) : goto fail Apple

```
/* Extract from Apple's sslKeyExchange.c */
if ((err=SSLHashSHA1.update(&hashCtx,&serverRandom))!=0)
    goto fail;
if ((err=SSLHashSHA1.update(&hashCtx,&signedParams))!=0)
    goto fail;
if ((err=SSLHashSHA1.final(&hashCtx,&hashOut))!=0)
    goto fail;
```

La syntaxe n'aide pas, mais on aimerait que le compilateur nous signale ce code qui est clairement mort

## Erreurs de logique (1/2) : True, False, FILE\_NOT\_FOUND

Analysons la faille CVE-2014-0092 de GnuTLS (mars 2014) :

*But this bug is arguably much worse than APPLE's, as it has allowed crafted certificates to evade validation check for all versions of GNUTLS ever released since that project got started in late 2000.[...]*

*The `check_if_ca` function is supposed to return true (any non-zero value in C) or false (zero) depending on whether the issuer of the certificate is a certificate authority (CA). A true return should mean that the certificate passed muster and can be used further, but the bug meant that **error returns were misinterpreted as certificate validations.***

## Erreurs de logique (1/2) : True, False, FILE\_NOT\_FOUND

Analysons la faille CVE-2014-0092 de GnuTLS (mars 2014) :

*But this bug is arguably much worse than APPLE's, as it has allowed crafted certificates to evade validation check for all versions of GNUTLS ever released since that project got started in late 2000.[...]*

*The `check_if_ca` function is supposed to return true (any non-zero value in C) or false (zero) depending on whether the issuer of the certificate is a certificate authority (CA). A true return should mean that the certificate passed muster and can be used further, but the bug meant that **error returns were misinterpreted as certificate validations**.*

Au passage, une faille similaire avait été trouvée dans OpenSSL... en 2008 (CVE-2008-5077).

## Erreurs de logique (2/2) : L'oubli de l'extension Basic constraints

Un *bug* de Microsoft Internet Explorer découvert par Marlinspike en 2002 :

- ▶ la pile X.509 ne vérifiait pas l'extension Basic Constraints
- ▶ tout certificat final pouvait servir de certificat d'autorité



## Erreurs de logique (2/2) : L'oubli de l'extension Basic constraints

Un *bug* de Microsoft Internet Explorer découvert par Marlinspike en 2002 :

- ▶ la pile X.509 ne vérifiait pas l'extension Basic Constraints
- ▶ tout certificat final pouvait servir de certificat d'autorité

Le même *bug*, en 2010, dans Apple iOS...

## Erreurs de logique (2/2) : L'oubli de l'extension Basic constraints

Un *bug* de Microsoft Internet Explorer découvert par Marlinspike en 2002 :

- ▶ la pile X.509 ne vérifiait pas l'extension Basic Constraints
- ▶ tout certificat final pouvait servir de certificat d'autorité

Le même *bug*, en 2010, dans Apple iOS...

Il est peut-être temps de considérer que les développeurs ne sont pas seuls responsables... et d'améliorer nos langages/outils/spécifications

## La cryptographie obsolète (1/2) : Bleichenbacher

L'attaque de Bleichenbacher est-elle une erreur d'implémentation ?

L'attaque a resurgi en 2014

- ▶ en Java, une erreur de *padding* provoque une exception
- ▶ ainsi, pour éviter une *timing attack*, il faut redévelopper l'algorithme
- ▶ un développeur TLS doit choisir entre la modularité et la sécurité

## La cryptographie obsolète (1/2) : Bleichenbacher

L'attaque de Bleichenbacher est-elle une erreur d'implémentation ?

L'attaque a resurgi en 2014

- ▶ en Java, une erreur de *padding* provoque une exception
- ▶ ainsi, pour éviter une *timing attack*, il faut redévelopper l'algorithme
- ▶ un développeur TLS doit choisir entre la modularité et la sécurité

Et en 2016...

- ▶ DROWN (*Decrypting RSA with Obsolete and Weakened eNcryption*)
- ▶ attaque sur SSLv2 pour retrouver des *pre-master secret* TLS
- ▶ des *bugs* dans OpenSSL rendent l'attaque très efficace

## La cryptographie obsolète (1/2) : Bleichenbacher

L'attaque de Bleichenbacher est-elle une erreur d'implémentation ?

L'attaque a resurgi en 2014

- ▶ en Java, une erreur de *padding* provoque une exception
- ▶ ainsi, pour éviter une *timing attack*, il faut redévelopper l'algorithme
- ▶ un développeur TLS doit choisir entre la modularité et la sécurité

Et en 2016...

- ▶ DROWN (*Decrypting RSA with Obsolete and Weakened eNcryption*)
- ▶ attaque sur SSLv2 pour retrouver des *pre-master secret* TLS
- ▶ des *bugs* dans OpenSSL rendent l'attaque très efficace
- ▶ heureusement que SSLv2 est obsolète et inutilisé en pratique...

## La cryptographie obsolète (2/2) : MAC-then-Encrypt

Les oracles de *padding* existent aussi en symétrique

- ▶ *MAC-then-CBC* est naturellement vulnérable
- ▶ 2002 : Vaudenay présente le principe de l'attaque
- ▶ 2011 : *XML Encryption is broken*
- ▶ 2013 : Lucky 13 (l'attaque est applicable à TLS)

## La cryptographie obsolète (2/2) : MAC-then-Encrypt

Les oracles de *padding* existent aussi en symétrique

- ▶ *MAC-then-CBC* est naturellement vulnérable
- ▶ 2002 : Vaudenay présente le principe de l'attaque
- ▶ 2011 : *XML Encryption is broken*
- ▶ 2013 : Lucky 13 (l'attaque est applicable à TLS)

Le correctif ?

- ▶ du sparadrap (une note d'implémentation dans le standard)
- ▶ un *patch* sordide pour garantir un déchiffrement en temps constant
- ▶ utiliser *Encrypt-then-MAC* (RFC 7366) ou du chiffrement authentifié (AEAD)

Il faut choisir entre modularité et sécurité (et interopérabilité)

# TLS dans tous ses états : SMACK et FREAK

En 2015, des attaques concernant presque toutes les piles TLS

- ▶ en Java, l'envoi d'un message `Finished` précoce permettait à un attaquant de sauter toute la négociation (dont l'authentification)



# TLS dans tous ses états : SMACK et FREAK

En 2015, des attaques concernant presque toutes les piles TLS

- ▶ en Java, l'envoi d'un message `Finished` précoce permettait à un attaquant de sauter toute la négociation (dont l'authentification)
- ▶ OpenSSL acceptait l'échange de clé RSA-EXPORT lorsque le chiffrement RSA a été négocié (FREAK)

# TLS dans tous ses états : SMACK et FREAK

En 2015, des attaques concernant presque toutes les piles TLS

- ▶ en Java, l'envoi d'un message `Finished` précoce permettait à un attaquant de sauter toute la négociation (dont l'authentification)
- ▶ OpenSSL acceptait l'échange de clé RSA-EXPORT lorsque le chiffrement RSA a été négocié (FREAK)

En général, la machine à états réagit aux messages reçus, au lieu de maintenir une liste restreinte des messages licites

## Autres problèmes dans les machines à états

- ▶ *Early CCS*, trouvé à l'aide de méthodes formelles
- ▶ Pour WinShock, les messages `Certificate` et `CertificateVerify` étaient toujours interprétés, même s'ils n'étaient pas sollicités

Presque toutes les piles TLS étaient vulnérables à des attaques similaires :

- ▶ les spécifications sont peut-être trop complexes ?
- ▶ nous avons sans doute besoin de plus de tests

# Réflexions sur les failles d'implémentation

Attention à ne pas blâmer trop vite le développeur

- ▶ implémenter certaines primitives crypto proprement relève du jonglage
- ▶ au vu des erreurs répétées, les spécifications sont trop complexes ?

Éléments de solution

- ▶ des spécifications claires
- ▶ le retrait des primitives obsolètes
  - ▶ de la spécification
  - ▶ des déploiements
  - ▶ et *in fine* du code !
- ▶ de meilleures méthodologies de développements
  - ▶ utilisation de langages sûrs ou d'outils d'analyse de code
  - ▶ plus de tests

# Master class sur TLS @ FIC 2017

Description rapide de SSL/TLS

Plus de vingt ans de vulnérabilités

- Handshake Protocol

- Record Protocol

- Erreurs d'implémentation

TLS 1.3 : un nouvel espoir ?

# TLS 1.3 : un travail initié il y a plus de 3 ans

## Avertissement préalable

Cette présentation repose sur le *draft* 18 de la spécification TLS 1.3 en cours de définition

## Rapide historique de TLS 1.3

- ▶ novembre 2013 : premiers éléments sur la liste IETF
- ▶ avril 2014 : TLS 1.3 *draft* 00
- ▶ novembre 2016 : *draft* 18
- ▶ janvier 2017 : *Working Group Last Call*

## Cahier des charges

- ▶ dépoussiérer le protocole
- ▶ accélérer l'établissement de session
- ▶ prendre en compte les vulnérabilités publiées

# TLS 1.3 : un protocole avec de la cryptographie moderne

## Handshake Protocol

- ▶ retrait de l'échange de clé par chiffrement RSA
- ▶ définition de groupes nommés pour (EC)DHE
- ▶ retrait de la renégociation
- ▶ retrait de la compression
- ▶ signature de l'ensemble du transcript par le serveur
- ▶ mécanisme *anti-rollback*
- ▶ décorrélation du secret de reprise de session

# TLS 1.3 : un protocole avec de la cryptographie moderne

## Handshake Protocol

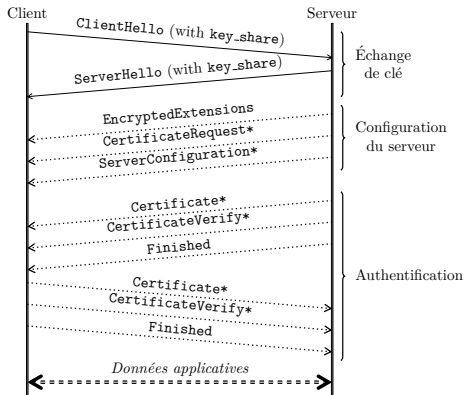
- ▶ retrait de l'échange de clé par chiffrement RSA
- ▶ définition de groupes nommés pour (EC)DHE
- ▶ retrait de la renégociation
- ▶ retrait de la compression
- ▶ signature de l'ensemble du transcript par le serveur
- ▶ mécanisme *anti-rollback*
- ▶ décorrélation du secret de reprise de session

## Record Protocol

- ▶ retrait du mode CBC
- ▶ retrait des algorithmes faibles (RC4, DES, 3DES)
- ▶ seul le mode AEAD est conservé (AES-GCM, ChaCha20+Poly1035)



# TLS 1.3 : un protocole plus rapide



- ▶ 1 RTT si le client propose un groupe (EC)DHE
- ▶ 2 RTT sinon
- ▶ 0 RTT possible en cas de reprise de session

## TLS 1.3 : un protocole plus simple et plus sûr

Beaucoup moins de fonctionnalités et d'options

- ▶ deux mode d'échange de clé : (EC)DHE et PSK (*Pre-Shared Key*)
- ▶ la reprise de session se fait par PSK
- ▶ un seul mode de protection des *records* : AEAD
- ▶ machine à état simplifiée

## TLS 1.3 : un protocole plus simple et plus sûr

Beaucoup moins de fonctionnalités et d'options

- ▶ deux mode d'échange de clé : (EC)DHE et PSK (*Pre-Shared Key*)
- ▶ la reprise de session se fait par PSK
- ▶ un seul mode de protection des *records* : AEAD
- ▶ machine à état simplifiée

Depuis plus d'un an, de nombreux de travaux de vérification formelle ont été publiés

- ▶ janvier 2016 : atelier TRON en marge de NDSS
- ▶ miTLS (Inria Prosecco)
- ▶ modélisation utilisant Tamarin
- ▶ attaques cross-protocoles (TLS 1.3 et TLS 1.2/QUIC)

## Le côté obscur de TLS 1.3

### Authentification tardive du client

- ▶ le serveur peut demander un certificat au client
- ▶ n'importe quand
- ▶ il peut envoyer plusieurs requêtes en parallèle
- ▶ le client doit maintenir un état non négligeable
- ▶ *a priori, cette fonctionnalité devrait être corrigée*

## Le côté obscur de TLS 1.3

### Authentification tardive du client

- ▶ le serveur peut demander un certificat au client
- ▶ n'importe quand
- ▶ il peut envoyer plusieurs requêtes en parallèle
- ▶ le client doit maintenir un état non négligeable
- ▶ *a priori, cette fonctionnalité devrait être corrigée*

### Mode 0 RTT

- ▶ utilisation de la PSK pour chiffrer des données dès le ClientHello
- ▶ rejeu de messages possibles
- ▶ complexification de l'automate
- ▶ *cette fonctionnalité est optionnelle*

# TLS 1.3 : une transition longue et difficile à prévoir

## État des lieux

- ▶ TLS 1.3 devrait être publié en 2017
- ▶ quelques piles serveurs (Google/Cloudflare)
- ▶ quelques piles client (NSS/Chrome)
- ▶ le support TLS 1.3 est prévu dans OpenSSL pour avril 2017

## Transition

- ▶ il faudra cohabiter avec les versions précédentes de TLS encore au moins une décennie
- ▶ SSLv3 vient juste de disparaître du monde HTTP
- ▶ mais SSLv2 est encore d'actualité dans l'écosystème SMTP...

## Conclusion

SSL/TLS est un protocole (trop) riche à l'histoire semée de vulnérabilités

- ▶ beaucoup de spécifications et de nombreuses fonctionnalités
- ▶ un écosystème varié, à l'évolution lente
- ▶ les implémentations doivent faire face à des défis intéressants

## Conclusion

SSL/TLS est un protocole (trop) riche à l'histoire semée de vulnérabilités

- ▶ beaucoup de spécifications et de nombreuses fonctionnalités
- ▶ un écosystème varié, à l'évolution lente
- ▶ les implémentations doivent faire face à des défis intéressants

TLS 1.3 : un nouvel espoir ?

- ▶ la plupart des primitives obsolètes ont été retirées
- ▶ sans le mode 0 RTT, la spécification est plus simple que TLS 1.2
- ▶ mais ce mode représente un risque en termes de sécurité



## Conclusion

SSL/TLS est un protocole (trop) riche à l'histoire semée de vulnérabilités

- ▶ beaucoup de spécifications et de nombreuses fonctionnalités
- ▶ un écosystème varié, à l'évolution lente
- ▶ les implémentations doivent faire face à des défis intéressants

TLS 1.3 : un nouvel espoir ?

- ▶ la plupart des primitives obsolètes ont été retirées
- ▶ sans le mode 0 RTT, la spécification est plus simple que TLS 1.2
- ▶ mais ce mode représente un risque en termes de sécurité

Sécuriser les communications est un problème global

- ▶ au-delà de la spécification, il faut vérifier les implémentations
- ▶ quid de la cohabitation avec les versions précédentes du protocole ?
- ▶ quid de la gestion de la confiance ?
- ▶ quid de la remédiation de la perte des secrets du serveur ?

Questions ?

Merci pour votre attention

# Compléments

## Pourquoi *{False, Snap} Start* a échoué ?

Un feuilleton qui a duré plusieurs années

- ▶ en 2010, Google propose des extensions, *False Start* et *Snap Start*
- ▶ constat : une intolérance trop importante dans la nature
- ▶ abandon en 2012 de ces propositions

## Pourquoi *{False, Snap} Start* a échoué ?

Un feuilleton qui a duré plusieurs années

- ▶ en 2010, Google propose des extensions, *False Start* et *Snap Start*
- ▶ constat : une intolérance trop importante dans la nature
- ▶ abandon en 2012 de ces propositions
  
- ▶ un an plus tard, le problème resurgit dans un autre contexte
- ▶ on apprend sur la liste `tls@ietf.org` qu'en fait, le problème vient d'un `ClientHello` trop gros...

## Ce paquet est une chimère

Analysons les premiers octets d'un ClientHello de 258 octets

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

## Ce paquet est une chimère

Analysons les premiers octets d'un ClientHello de 258 octets

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

Type	Version	Longueur	<b>TLS</b>
<i>HS</i>	<i>TLS 1.0</i>	<i>258</i>	

## Ce paquet est une chimère

Analysons les premiers octets d'un ClientHello de 258 octets

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

Type	Version	Longueur
<i>HS</i>	<i>TLS 1.0</i>	<i>258</i>

**TLS**

Longueur	<i>Pad.</i>	Type
<i>5635</i>	<i>...</i>	<i>CH</i>

**SSLv2**

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



## Ce paquet est une chimère

Analysons les premiers octets d'un ClientHello de 258 octets

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

Type	Version	Longueur
<i>HS</i>	<i>TLS 1.0</i>	<i>258</i>

**TLS**

Longueur	<i>Pad.</i>	Type
<i>5635</i>	<i>...</i>	<i>CH</i>

**SSLv2**

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

Google a proposé une extension pour ajouter du bourrage...