Regards critiques sur SSL/TLS

Olivier Levillain

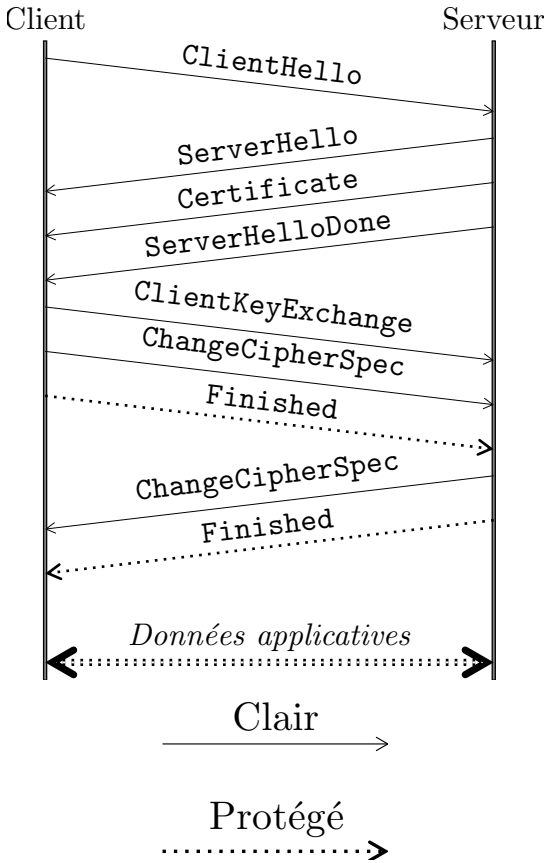
ANSSI/SDE/ST/LRP

Conférence ESSI du 17 septembre 2014

Table des matières

Table des matières

TLS en un schéma (la planche développement durable)



Deux phases

- établissement du canal sécurisé
  - négociation des algorithmes
  - authentification du serveur
  - établissement d'un secret partagé
- échange de données grâce à ce canal

Historique

- 1994 : publication de SSLv2 par Netscape, et naissance de `https://`
- 1995 : correction de failles majeures et publication de SSLv3
- 1999 : reprise du standard par l'IETF sous le nom TLS

- 2006 : TLSv1.1

- 2008 : TLSv1.2

- 2014 (?) : TLSv1.3

Historique

- 1994 : publication de SSLv2 par Netscape, et naissance de `https://`
- 1995 : correction de failles majeures et publication de SSLv3
- 1999 : reprise du standard par l'IETF sous le nom TLS

- 2004 : support de la compression dans TLS

- 2006 : TLSv1.1

- 2006 : support des *session tickets*

- 2008 : TLSv1.2

- 2012 : extension *Heartbeat*
- 2014 (?) : TLSv1.3

Historique

- 1994 : publication de SSLv2 par Netscape, et naissance de `https://`
- 1995 : correction de failles majeures et publication de SSLv3
- 1999 : reprise du standard par l'IETF sous le nom TLS
- 2003 : suites Kerberos
- 2004 : support de la compression dans TLS
- 2005 : suites SEED et PSK
- 2006 : TLSv1.1
- 2006 : courbes elliptiques
- 2006 : support des *session tickets*
- 2007 : suites SRP
- 2008 : TLSv1.2
- 2007 : suites GCM
- 2011 : suites Camellia et ARIA
- 2012 : extension *Heartbeat*
- 2014 (?) : TLSv1.3

SSL/TLS en chiffres

- 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`

B `ECDH-ECDSA-AES128-SHA`

C une alerte

D la réponse D (`RC4_MD5`)

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 ?

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 lite `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

TLS

Type	Version	Longueur
HS	*TLS 1.0*	*258*

SSLv2

Longueur	*Pad.*	Type
5635	...	*CH*

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

TLS

Type	Version	Longueur
HS	*TLS 1.0*	*258*

SSLv2

Longueur	*Pad.*	Type
5635	...	*CH*

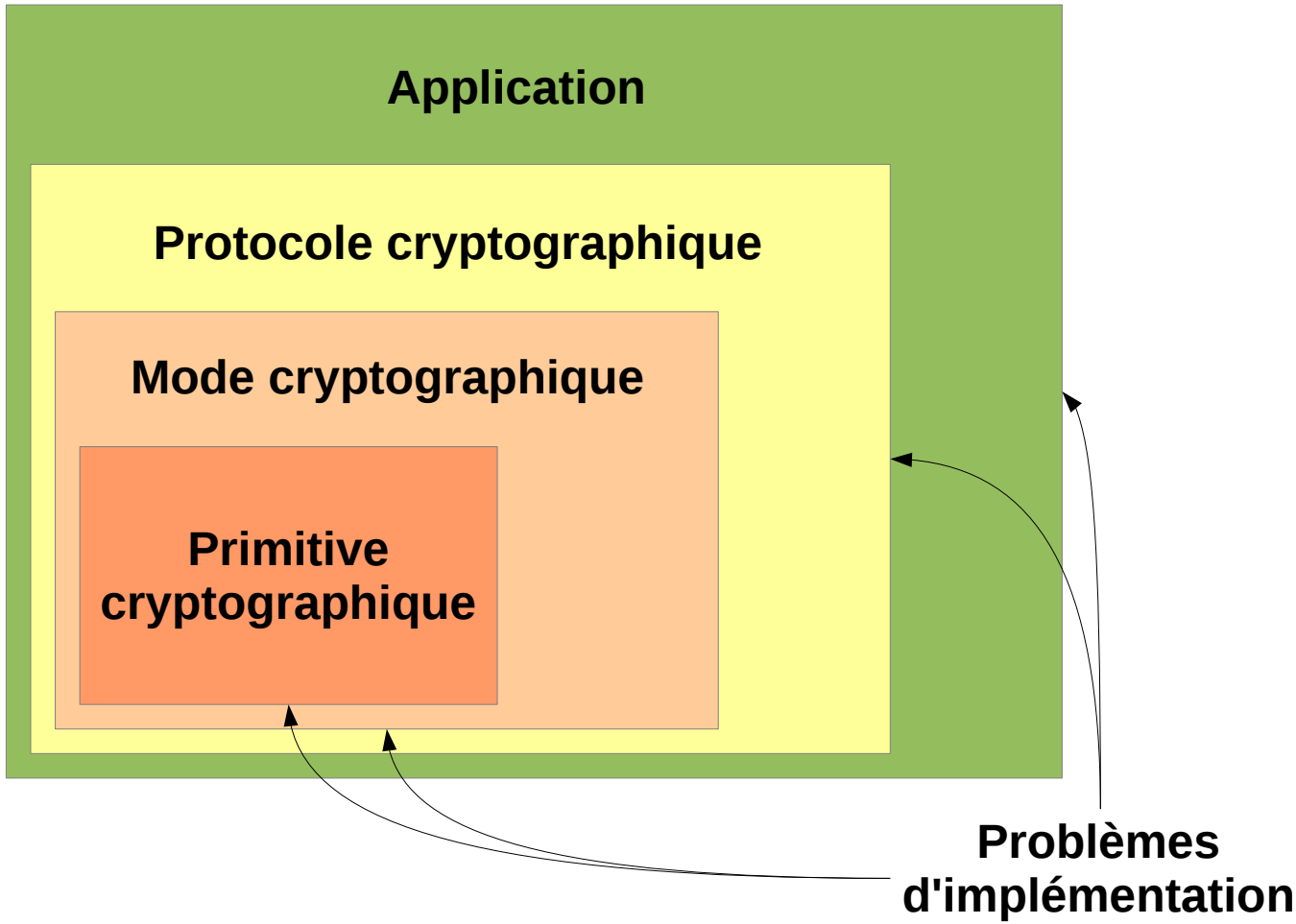
Un `ClientHello` TLS dont la longueur comprise entre 256 et 511 peut être confondu avec un `ClientHello` SSLv2 !

Là encore, la chute est amusante :

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

Table des matières

Établir une communication sécurisée : un problème complexe



Quelques vulnérabilités de SSL/TLS

Quelques vulnérabilités de SSL/TLS

- 1995 : négociation à la baisse dans SSLv2
- 2009 : attaque par renégociation
- 2011 : CRIME (Utilisation de la compression comme canal auxiliaire)

Quelques vulnérabilités de SSL/TLS

- 1995 : négociation à la baisse dans SSLv2
- 2009 : attaque par renégociation
- 2011 : CRIME (Utilisation de la compression comme canal auxiliaire)
- 2014 : *Triple Handshake* (attaque par renégociation/reprise de session)

Quelques vulnérabilités de SSL/TLS

- 1995 : négociation à la baisse dans SSLv2
- 1998 : attaque de Bleichenbacher

- 2009 : Collision MD5 sur des certificats réels
- 2009 : attaque par renégociation

- 2011 : BEAST (IV implicite dans le mode CBC)

- 2011 : CRIME (Utilisation de la compression comme canal auxiliaire)

- 2013 : Lucky 13 (CBC padding oracle) + biais statistique de RC4 dans TLS

- 2014 : *Triple Handshake* (attaque par renégociation/reprise de session)

Quelques vulnérabilités de SSL/TLS

- 1995 : négociation à la baisse dans SSLv2
- 1998 : attaque de Bleichenbacher
- 2002 : mauvaise interprétation de l'extension X.509 *Basic Constraints* (IE)

- 2009 : Collision MD5 sur des certificats réels
- 2009 : attaque par renégociation
- 2009 : mauvaise gestion des caractères nuls dans les certificats
- 2011 : BEAST (IV implicite dans le mode CBC)

- 2011 : CRIME (Utilisation de la compression comme canal auxiliaire)

- 2013 : Lucky 13 (CBC padding oracle) + biais statistique de RC4 dans TLS
- 2014 : `goto fail` Apple
- 2014 : contournement de la validation de certificats dans GNUTls
- 2014 : *Triple Handshake* (attaque par renégociation/reprise de session)
- 2014 : Heartbleed

Quelques vulnérabilités de SSL/TLS

- 1995 : négociation à la baisse dans SSLv2
- 1998 : attaque de Bleichenbacher
- 2002 : mauvaise interprétation de l'extension X.509 *Basic Constraints* (IE)
- 2008 : contournement de la validation de certificats dans OpenSSL
- 2009 : Collision MD5 sur des certificats réels
- 2009 : attaque par renégociation
- 2009 : mauvaise gestion des caractères nuls dans les certificats
- 2011 : BEAST (IV implicite dans le mode CBC)

- 2011 : CRIME (Utilisation de la compression comme canal auxiliaire)

- 2013 : Lucky 13 (CBC padding oracle) + biais statistique de RC4 dans TLS
- 2014 : `goto fail` Apple
- 2014 : contournement de la validation de certificats dans GNUTls
- 2014 : *Triple Handshake* (attaque par renégociation/reprise de session)
- 2014 : Heartbleed

Quelques vulnérabilités de SSL/TLS

- 1995 : négociation à la baisse dans SSLv2
- 1998 : attaque de Bleichenbacher
- 2002 : mauvaise interprétation de l'extension X.509 *Basic Constraints* (IE)
- 2008 : contournement de la validation de certificats dans OpenSSL
- 2009 : Collision MD5 sur des certificats réels
- 2009 : attaque par renégociation
- 2009 : mauvaise gestion des caractères nuls dans les certificats
- 2011 : BEAST (IV implicite dans le mode CBC)
- 2011 : mauvaise interprétation de l'extension X.509 *Basic Constraints* (iOS)
- 2011 : CRIME (Utilisation de la compression comme canal auxiliaire)

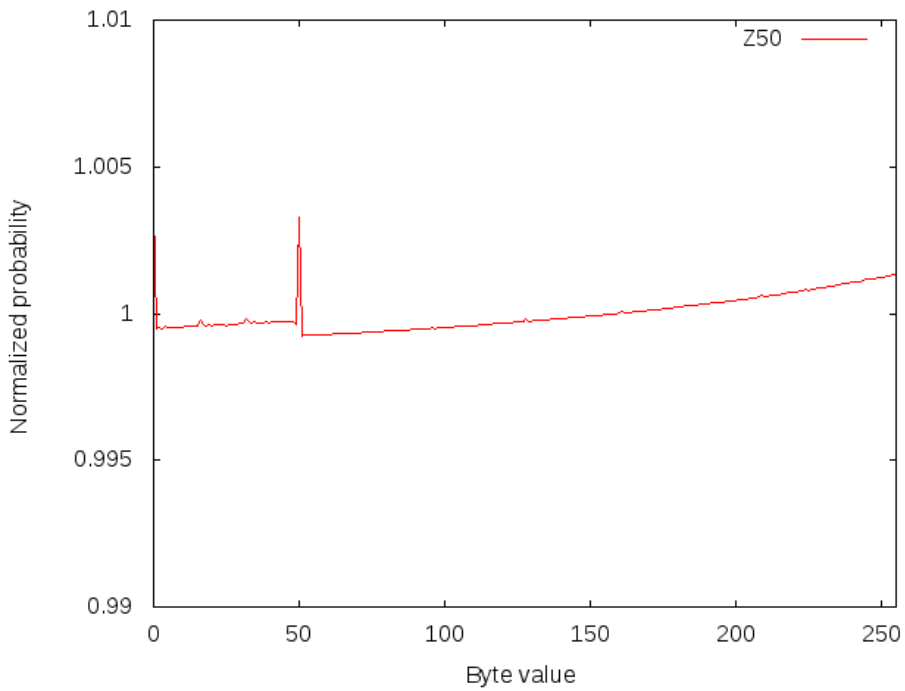
- 2013 : Lucky 13 (CBC padding oracle) + biais statistique de RC4 dans TLS
- 2014 : `goto fail` Apple
- 2014 : contournement de la validation de certificats dans GNUTls
- 2014 : *Triple Handshake* (attaque par renégociation/reprise de session)
- 2014 : Heartbleed

Quelques vulnérabilités de SSL/TLS

- 1995 : négociation à la baisse dans SSLv2
- 1998 : attaque de Bleichenbacher
- 2002 : mauvaise interprétation de l'extension X.509 *Basic Constraints* (IE)
- 2008 : contournement de la validation de certificats dans OpenSSL
- 2009 : Collision MD5 sur des certificats réels
- 2009 : attaque par renégociation
- 2009 : mauvaise gestion des caractères nuls dans les certificats
- 2011 : BEAST (IV implicite dans le mode CBC)
- 2011 : mauvaise interprétation de l'extension X.509 *Basic Constraints* (iOS)
- 2011 : CRIME (Utilisation de la compression comme canal auxiliaire)
- 2012 : *Mind your Ps and Qs* (problème d'aléa dans la génération de clés RSA)
- 2013 : Lucky 13 (CBC padding oracle) + biais statistique de RC4 dans TLS
- 2014 : `goto fail` Apple
- 2014 : contournement de la validation de certificats dans GNUTls
- 2014 : *Triple Handshake* (attaque par renégociation/reprise de session)
- 2014 : Heartbleed

Exemple sur une primitive crypto : RC4

- 1987 : algorithme de chiffrement par flot propriétaire conçu par Rivest
- 1994 : divulgation d'ARCFOUR
- 1995-2000 : découverte des premiers biais sur les octets de *keystream*
- 2001 : attaque de WEP (clés corrélées) par Flührer, Mantin et Shamir
- 2013 : découverte de biais exploitable dans le cadre de TLS



Exemple sur l'utilisation de la crypto : Bleichenbacher

RSA PKCS#1 v1.5

- le chiffrement repose sur une mise en forme préalable (*padding*)
- au déchiffrement, comment traiter un format invalide ?

Attaque de Bleichenbacher (1998)

- principe : demander le déchiffrement d'un chiffré modifié
- si on sait distinguer un `padding` est valide, une attaque est possible
- application à TLS : *Million Message Attack*

Réapparition de l'attaque en 2014

- en Java, une erreur de *padding* mène à une exception
- pour éviter une *timing attack*, il faut éviter les bibliothèques
- problème de modularité : pour implémenter TLS, il faut recoder la crypto !

Exemple sur l'utilisation de la crypto : *MAC-then-Encrypt*

Les problèmes de *padding* existent aussi en symétrique

- protéger le flux avec *MAC-then-CBC* est vulnérable
- 2002 : Vaudenay présente le principe de l'attaque
- 2011 : *XML Encryption is broken*
- 2013 : Lucky 13 démontre l'applicabilité à TLS

Contre-mesure

- du sparadrap (note d'implémentation dans TLS 1.0)
- un *patch* sordide pour déchiffrer en temps constant
- passer (enfin) à `Encrypt-then-MAC` (RFC 7366 !)

Là encore, il fallait choisir entre modularité et sécurité...

Exemple de faille dans le protocole : la renégo / THS

Deux attaques récentes sur le *protocole*

- 2009 : problème de la renégociation sécurisée
- 2014 : *Triple Handshake*

Analyse

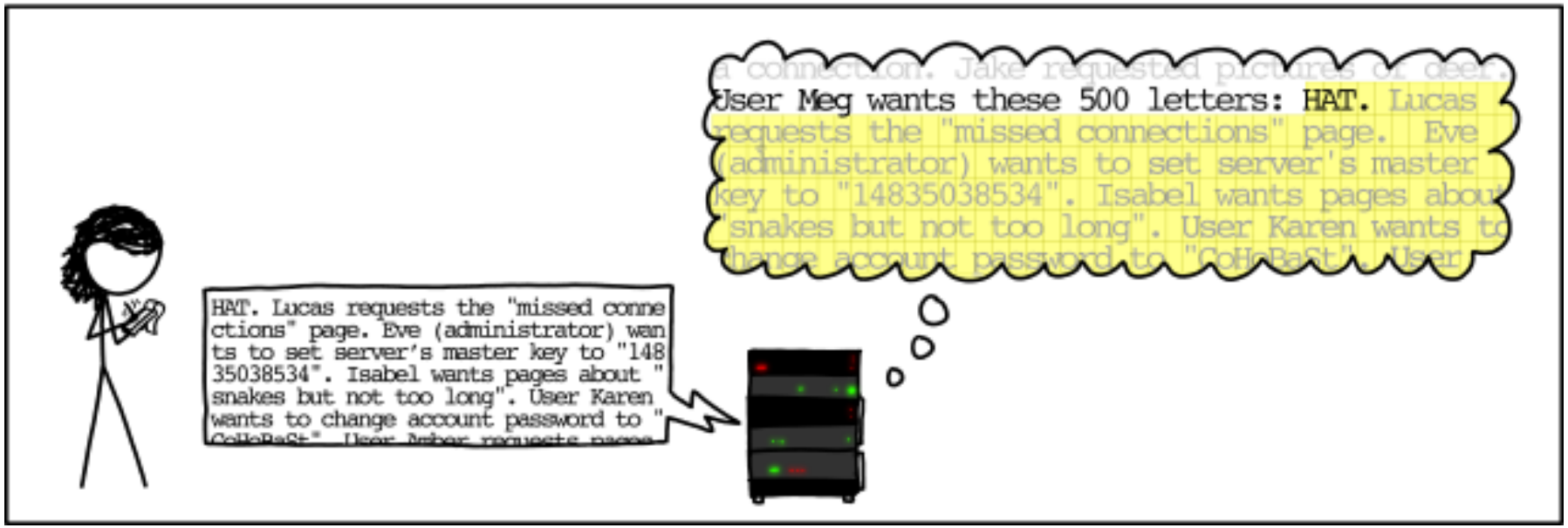
- attaques difficiles à décrire...
- elles sortent de la plupart des modèles envisagés au préalable
- la véritable question est : quelle est la propriété de sécurité attendue de TLS ?

Exemple de faille dans l'application : la vérification de la chaîne de certification

Depuis plusieurs années, de nombreuses failles liées à la vérification des certificats

- de nombreuses bibliothèques TLS ne vérifient *pas* le *hostname*
  - python, ruby
  - API sordides (0, 1 ou 2 ? !)
- des erreurs bêtes dans le code
  - 2008 : contournement de la validation de certificats dans OpenSSL (à cause d'une valeur de retour étrange : -1, 0 ou 1...)
  - 2014 : `goto fail` Apple
  - 2014 : contournement de la validation de certificats dans GNUTls (similaire à OpenSSL)

Exemple de *bug* d'implémentation : *Heartbleed*



Source : `http://xkcd.com/1354`

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

Table des matières

Comparaison avec d'autres protocoles

IKEv2/IPsec

+ séparation claire entre la négociation et la protection du flux
+ choix de *Encrypt-then-Mac*
+ de manière générale, conception pensée pour permettre la preuve
+ une gestion des clés par le noyau
− *un besoin d'interagir avec l'OS*

SSH

− *un mélange entre négociation et protection du flux*
+ ... mais avec un ordonnancement structuré
− *choix initial de* Encrypt-and-Mac
+ ... mais acceptable avec un mode compteur, généralement négocié
+ ... et *Encrypt-then-Mac* a été implémenté finalement
− *une négociation* très *bavarde*
− *SSH est une vraie couche transport (gestion de la congestion)*

Comment éviter ces failles

Primitives crypto

- fournir l'agilité cryptographiques
- et s'en servir au besoin : proposer des primitives éprouvées

Utilisation de la crypto

- utiliser des mécanismes éprouvés et aidant à la preuve
- objectif : pouvoir abstraire la couche crypto et obtenir un schéma modulaire

Spécification du protocole

- proposer un schéma simple
- lever toutes les ambiguïtés

Lien avec les applications

- améliorer les interfaces (*secure defaults*)

Implémentations

- besoin d'experts dans tous les domaines (crypto, système, réseau)
- code analysable (et analysé)

TLS 1.3 ?

Depuis plusieurs mois, TLS 1.3 en cours de discussion, avec du bon

- plus (+) de *forward secrecy*
- retrait des vieilles suites crypto (RC4, *Mac-then-Encrypt*)
- retrait de la compression
- amélioration de la négociation des groupes Diffie-Hellman

et du moins bon

- recherche de modes d'établissement rapide (1RTT, voire 0RTT)
- un machine à état encore plus complexe
- ... mais toujours pas formalisée proprement
- et de nombreuses discussions stériles pour réconcilier ces objectifs contradictoires

Affaire à suivre...

Questions ?

Merci de votre attention.

Références :

- Blog de Matthew Green (`http://blog.cryptographyengineering.com`)
- Crypto Fails (`http://www.cryptofails.com`)
- Cryptography Coding Standard `https://cryptocoding.net`