

# Cours de cryptologie appliquée de l'EPITA

## TLS - partie 1

Manuel Pégourié-Gonnard

ARM France - IoT - mbed TLS

19 novembre 2015

<https://github.com/mpg/cours-tls>

CC-BY-SA 4.0

## Introduction

Généralités sur la couche record

Chiffrement par flot avec RC4

Chiffrement par bloc avec CBC

Chiffrement authentifié

Faisons le point

# TLS ça fait quoi ?

Ça assure la sécurité des *communications*:

**Confidentialité** Un attaquant qui peut tout écouter ne peut rien apprendre sur les données échangées, à part peut-être leur longueur.

**Intégrité** Un attaquant qui modifie les données en transit sera détecté.

**Authentification** optionnelle d'une ou des deux parties :  
l'attaquant ne peut pas se faire passer pour quelqu'un d'autre.

## Quelles menaces ?

RFC 3553 section 3: the Internet threat model (2003)

### Attaquant réseau passif

- Ne fait qu'écouter
- Attaque la confidentialité

### Attaquant réseau actif

- Peut modifier les messages
- Attaque l'authenticité, l'intégrité, la confidentialité
- Man-in-the-middle (MitM)

## Quelles menaces ?

RFC 3553 section 3: the Internet threat model (2003)

### Attaquant réseau passif

- Ne fait qu'écouter
- Attaque la confidentialité
- Échelle (perpass)

### Attaquant réseau actif

- Peut modifier les messages
- Attaque l'authenticité, l'intégrité, la confidentialité
- Man-in-the-middle (MitM)
- Plus coûteux (ciblé ?)

## Quelles menaces ?

RFC 3553 section 3: the Internet threat model (2003)

### Attaquant réseau passif

- Ne fait qu'écouter
- Attaque la confidentialité
- Échelle (perpass)
- Attaques « du futur »

### Attaquant réseau actif

- Peut modifier les messages
- Attaque l'authenticité, l'intégrité, la confidentialité
- Man-in-the-middle (MitM)
- Plus coûteux (ciblé ?)

## Quelles menaces ?

RFC 3553 section 3: the Internet threat model (2003)

### Attaquant réseau passif

- Ne fait qu'écouter
- Attaque la confidentialité
- Échelle (perpass)
- Attaques « du futur »

### Attaquant réseau actif

- Peut modifier les messages
- Attaque l'authenticité, l'intégrité, la confidentialité
- Man-in-the-middle (MitM)
- Plus coûteux (ciblé ?)

### Tout le reste

Man-in-the-Browser, bugs, sécurité du poste local, utilisateurs, etc.

### Canaux auxiliaires

# Ça s'utilise où ?

## Couche

**Application** HTTP, IMAP, SMTP, XMPP,

**Session** TLS,

**Transport** TCP,

**Internet** IPv4, IPv6

**Lien** tout ce que vous voulez

## Modes

- avec port dédié : HTTP/80 → HTTPS/443
- sur le même port : STARTTLS (IMAP, SMTP, XMPP)



# Ça s'utilise où ?

## Couche

**Application** HTTP, IMAP, SMTP, XMPP, RTSP

**Session** TLS, DTLS

**Transport** TCP, UDP

**Internet** IPv4, IPv6

**Lien** tout ce que vous voulez

## Modes

- avec port dédié : HTTP/80 → HTTPS/443
- sur le même port : STARTTLS (IMAP, SMTP, XMPP)

# Versions

Nom	année	RFC	die-die-die	« vraie » version
SSL 1.0	–	–	mort-né	1.0
SSL 2.0	1995	–	6176 (2011)	2.0
SSL 3.0	1996	6101	7568 (2015)	3.0
TLS 1.0	1999	2246	–	3.1
TLS 1.1	2006	4346	–	3.2
DTLS 1.0	2006	4347	–	3.2
TLS 1.2	2008	5246	–	3.3
DTLS 1.2	2012	6347	–	3.3

# La réalité

En vrai, il faut lire un peu plus de RFC...

**Sécurité** 5746 secure renegotiation, 7627 session hash, 7366 encrypt-then-mac, 7507 downgrade SCSV,

**Algos** (300+ suites définies !): 4279 PSK, 4492 ECC, 5054 SRP, 5288 AES-GCM, 5289 ECC-AES-GCM, 5487 PSK-AES-GCM, 5489 ECDHE-PSK, 6655 AES-CCM, 7251 ECC-AES-CCM, ...

**Fonctionnalités** 6066 extensions diverses, 5878 autorisation, 6520 heartbeat, 5077 tickets de session, 7250 raw public key, 6091 OpenPGP, 7301 ALPN, ...

**Usage** 7457 attaques, 7525 bonnes pratiques générales, 7590 XMPP, d'autres à venir

Cf les registres sur [iana.org](http://iana.org).

## Plus de réalité

En 2015, sur les sites les plus populaires accessibles en HTTPS

- environ un tiers considérés sûrs ;
- 14 % acceptent des algos peu sûrs ;
- 31 % acceptent SSL 3.0 ;
- 69 % offrent TLS 1.2 ;
- 75 % offrent la *forward secrecy* ;
- 99,9 % ont un certificat avec un clé assez grande.

Source : <https://www.trustworthyinternet.org/ssl-pulse/>

Sur le million de sites les plus populaires, en 2014, seuls 45 % accessible en HTTPS...

## Introduction

## Généralités sur la couche record

Chiffrement par flot avec RC4

Chiffrement par bloc avec CBC

Chiffrement authentifié

Faisons le point

## Les couches de TLS

Handshake	ChangeCipherSpec	Alert	HTTP, SMTP, etc.
Record			
TCP, UDP			

- Handshake et ChangeCipherSpec (CCS) pour établir la connection
- Alert rarement : problèmes, fin de connection
- Record pour tout : encapsule, chiffre et authentifie les messages

## Structure globale

**Type** un octet : CCS = 20, Alert = 21, HS = 22, App = 23

**Version** deux octets : 0x03 0x03 pour TLS 1.2

**Length** deux octets : longueur du reste en gros-boutiste  
(limitée à  $2^{14}$  octets, soit 16 Ko).

T	VM	Vm	L1	L2	...
---	----	----	----	----	-----

Trois types de chiffrement-authentification :

1. Par flot (y compris NULL)
2. Par bloc avec CBC
3. AEAD = authenticated encryption with additional data  
(seulement TLS 1.2)

## Compression (1)

La couche *record* peut compresser avant de chiffrer. Ceci n'est *plus* recommandé.

### L'attaque CRIME

- Compression Ratio Info-leak Made Easy
- Idée de 2002, exploit (public) en 2012
- Application possible : fuite de cookies sécurisé
  1. On connaît `Cookie secret=`
  2. Pour chaque `x`, on ajoute `Cookie secret=x` et on mesure la longueur du chiffré
  3. La valeur qui donne un résultat plus court est la bonne
  4. On itère pour les caractères suivants
- Vole un cookie en 30 secondes

Dans TLS 1.3, la compression ne sera plus disponible.



## Compression (2)

### Conditions pour CRIME

1. L'attaquant peut injecter du texte clair (JS, actif)
2. L'attaquant peut observer le chiffré (réseau, passif)
3. La compresssion TLS doit être activée

### Variantes

**TIME** Timing Info-leak Make Easy : supprime la condition 2, mesure le temps à la place

**BREACH** Browser Reconnaissance & Exfiltration via Adaptive Compression of Hypertext : exploite la compression HTTP, plus répandue.

## Introduction

## Généralités sur la couche record

## Chiffrement par flot avec RC4

## Chiffrement par bloc avec CBC

## Chiffrement authentifié

## Faisons le point

## Chiffrement par flot (RC4)

### Chiffrement authentifié

- $MAC = HMAC(\text{type}, \text{version}, \text{longueur}, \text{message})$
- $\text{Chiffré} = RC4(\text{message}, MAC)$
- $\text{Envoyé} = \text{Chiffré}$
- L'état RC4 est conservé entre les messages (problème pour DTLS)

### Problème : RC4 n'est plus sûr

- 2013 : RC4 utilisé pour plus de 60 % des connections HTTPS
- 2013 : Royal Holloway, presque pratique ( $2^{24}$  chiffrés)
- début 2015 : RFC 7465 MUST NOT RC4
- mi 2015 : Bar-mitzvah, NOMORE, utilisable en pratique
- courant 2015 : RC4 retiré des navigateurs courants

# Apparté : comment vérifier un MAC

## Méthode naturelle

1. Calculer la bonne valeur
2. Comparer avec la valeur reçue avec `memcmp()`

## Attaque par timing

- La durée d'exécution de `memcmp()` est proportionnelle à la longueur du préfixe correct
- On brute-force octet par octet

## Une solution

```
unsigned char diff = 0;
for (size_t i = 0; i < len; i++)
    diff |= a[i] ^ b[i];
```

## Introduction

## Généralités sur la couche record

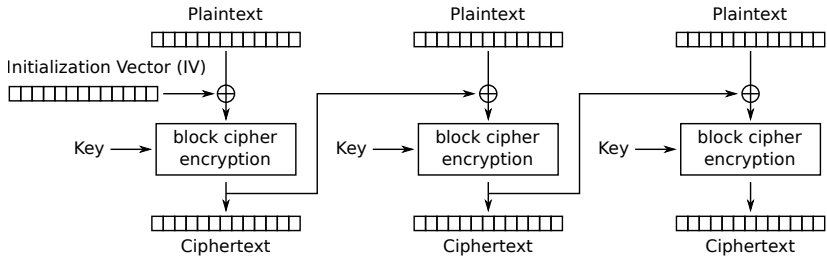
## Chiffrement par flot avec RC4

## Chiffrement par bloc avec CBC

## Chiffrement authentifié

## Faisons le point

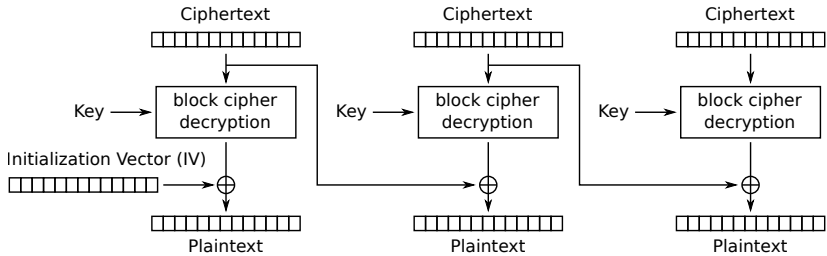
# Rappel : CBC



Cipher Block Chaining (CBC) mode encryption

(Crédit image : Wikipédia.)

# Rappel : CBC



Cipher Block Chaining (CBC) mode decryption

(Crédit image : Wikipédia.)

## Chiffrement authentifié avec CBC

### TLS: MAC then Encrypt (MtE)

- $MAC = HMAC(\text{métadonnées}, \text{message})$
- $\text{Sortie} = AES\text{-}CBC(\text{message}, MAC, \text{padding})$

### Encrypt and MAC

- $MAC = HMAC(\text{métadonnées}, \text{message})$
- $\text{Sortie} = AES\text{-}CBC(\text{message}, \text{padding}), MAC$

### Encrypt then MAC (EtM)

- $\text{Chiffré} = AES\text{-}CBC(\text{message}, \text{padding})$
- $MAC = HMAC(\text{métadonnées}, \text{chiffré})$
- $\text{Sortie} = \text{Chiffré}, MAC$



# Détails sur le padding

## Principe

- $L \in \{0, \dots, 255\}$  tel que  $L + 1 + l(m) = 0 \pmod{b}$ .
- En pratique,  $L \in \{0, \dots, b - 1\}$  et  $b = 16$  pour AES.
- Valeur avec SSL 3.0:  $L$  octets quelconques suivis d'un octet de valeur  $L$ .
- Valeur avec TLS 1.x:  $L + 1$  octets de valeur  $L$ .

## Exemple

- Message de longueur 4.
- Algos utilisés: AES et HMAC-SHA1.
- Longueur totale avant padding: 24 octets.
- Longueur typique du padding: 8 octets.
- Valeur du padding TLS: 07 07 07 07 07 07 07 07.

# Un oracle de padding : POODLE (1)

## Observation clé

- Supposons qu'il y a un block complet de padding, chiffré  $C_n$ .
- On remplace  $C_n$  par un  $C'$  quelconque.
- Alors le message est accepté ssi  $\text{AES}^{-1}(C') \oplus C_{n-1}$  se termine par un octet de valeur 15. (Exercice.)
- Si le message est accepté (une fois sur 256) on connaît le dernier octet de  $\text{AES}^{-1}(C')$ .
- La victime innocente s'est comportée comme un *oracle*.

## Contexte

- On cherche à voler un cookie
- On peut générer des requêtes contenant le cookie
- On peut modifier la requête chiffrée sur le chemin

## Un oracle de padding : POODLE (2)

POST /path Cookie: name=?????\r\n\r\n body MAC padding

1. On ajuste path et body pour que:
  - L'octet visé soit le dernier du bloc  $i$ .
  - Le padding soit de longueur un bloc exactement.
2. Le client innocent chiffre en  $C_1, \dots, C_i, \dots, C_n$ .
3. On envoie au serveur  $C_1, \dots, C_i, \dots, C_{n-1}, C_i$ .
4. Le serveur innocent nous donne une erreur, ou une fois sur 256, la valeur du dernier octet de  $\text{AES}^{-1}(C_i)$ .
5. On XORe avec le dernier octet de  $C_{i-1}$  pour trouver la valeur de l'octet visé.
6. On passe à un autre octet jusqu'à avoir tout le cookie.

## Un oracle de padding : POODLE (3)

### Analyse

- Le client innocent connaît le cookie secret et la clé de chiffrement.
- Le serveur innocent connaît la clé de chiffrement.
- Le serveur nous donne 8 bits du secret avec probabilité  $2^{-8}$ .

### Historique

- 1999 : problème résolu dans TLS 1.0
- 2003 : principe de l'oracle connu
- 2015 : exploit pratique publié, accélère la mort de SSL 3.0 (Padding Oracle On Downgraded Legacy Encryption)

# L'oracle de Vaudenay et Lucky 13

## Vaudenay

- Premier oracle de padding contre CBC publié (2002)
- POODLE en est une variante plus simple
- Fonctionne *presque* contre TLS 1.0+ (alertes chiffrées)
- Première variante par timing : 2003
- Contre-mesure 1 : toujours vérifier le MAC

## Lucky 13

- Autre variante utilisant le timing et un effet de seuil
- Contre-mesure 2 : toujours MACer la même longueur
- Contre-mesure délicate à implémenter (cf *Lucky 13 strikes back*: cache cross-VM)

# Encrypt-then-MAC

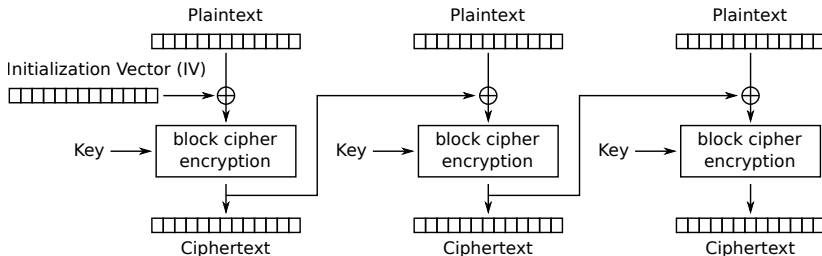
## Réparer CBC

- Cause commune à l'attaque de Vaudenay, POODLE et Lucky 13: le padding n'est pas authentifié.
- Solution: utiliser Encrypt-then-MAC (RFC 7366)

## Historique

- TLS 1.0 arrête POODLE
- Le RFC 5246 recommande la contre-mesure 1, et envisage la 2 mais pense qu'elle n'est pas nécessaire
- CBC avec EtM comme dans TLS 1.0 a une preuve de sécurité, qui ne prend pas en compte les canaux auxiliaires

## Vecteur d'initialisation



Cipher Block Chaining (CBC) mode encryption

**TLS 1.0-** Le tout premier est calculé au même moment que la clé, puis on réutilise le dernier bloc chiffré

**TLS 1.1+** Chaque IV est aléatoire et transmis explicitement

# Blockwise attack

## Principe

1. Rappel/notations  $C_i = E(C_{i-1} \oplus P_i)$ ; on suppose  $i < j$ .
2. En conséquence  $C_i = C_j \iff C_{i-1} \oplus P_i = C_{j-1} \oplus P_j$ .
3. Supposons que pour certains  $j$ , l'attaquant peut choisir  $P_j$  à un moment où  $C_k$  est connu pour tout  $k < j$ .
4. L'attaquant veut tester un candidat  $P'_i$  pour la valeur de  $P_i$ .
5. On choisit  $P_j = P'_i \oplus C_{i-1} \oplus C_{j-1}$  et on regarde si  $C_i = C_j$ .

## Analyse

- Publiée en 2004, corrigée en 2006 dans TLS 1.1 (point 3).
- Il faut deviner un bloc entier: la force brute ne marche pas
- Contourne la preuve de sécurité de CBC (CPA2  $\rightarrow$  BCPA2)



# BEAST

- 2011 : Browser Exploit Against SSL/TLS
- Principe : aligner les données pour n'avoir qu'un octet inconnu dans le bloc cible, puis le brute-forcer et itérer.
- Résolution correcte : passer à TLS 1.1 (70 % 4 ans après)
- Conseil à l'époque : utiliser RC4 à la place... oups !
- Contre-mesure pratique :  $1/n - 1$  record splitting
- Premier exploit médiatisé utilisant un MitB, a ouvert la voie

## Introduction

## Généralités sur la couche record

## Chiffrement par flot avec RC4

## Chiffrement par bloc avec CBC

## Chiffrement authentifié

## Faisons le point

# Authenticated Encryption with Additional Data

## Motivation

- Le choix de MtE au lieu de EtM est un problème de *composition* de primitive cryptographiques (chiffre et MAC)
- Problème de crypto, pas de chaque protocole
- Une API unifiée permet ceci et offre plus de liberté

**Avant** Plusieurs étapes définies par TLS :

MAC = HMAC(clé HMAC, métadonnées, message)

padding = cf plus haut

Sortie = AES-CBC(clé AES, IV, message, MAC, padding)

**Après** Sortie = AEAD(clé, nonce, message, métadonnées)

Disponible dans TLS depuis TLS 1.2.

# L'API AEAD (RFC 5116)

## Entrées

- Une clé secret unique (générée uniformément au hasard)
- Un nonce : ne doit *jamais* être réutilisé avec la même clé
- Le message à (dé)chiffrer
- Des données supplémentaires authentifiées mais pas chiffrées

## Sortie

- Soit une erreur (le chiffré n'est pas authentique)
- Soit le message (dé)chiffré

## En pratique

### AES-GCM

- Galois/Counter Mode (2005)
- Chiffrement avec AES en mode compteur
- Authentification utilisant la multiplication dans un corps fini
- Standard, preuve de sécurité, recommandé (Suite B, NIST, IETF).
- Potentiellement très rapide, encore plus avec AES-NI

### AES-CCM

Comparable, moins rapide mais implémentation plus compacte.

### Nonce

En pratique, utiliser le compteur de *records* TLS.

## Introduction

## Généralités sur la couche record

## Chiffrement par flot avec RC4

## Chiffrement par bloc avec CBC

## Chiffrement authentifié

## Faisons le point

# Historique

Année	Nom	Conditions	exploit
2002	Vaudenay	TLS 1.0- + accès aux alertes	non
2002	CBCATT	TLS 1.0- + CPA + chance	non
2003	CBCTIME	CPA	non
2011	BEAST	MitB + MitM	oui
2012	CRIME	compr TLS + MitB + écoute	oui
2013	TIME	compr TLS + MitB + temps	oui
2013	BREACH	compr HTTP + MitB + écoute	oui
2013	RC4 biases	session multiples	partiel
2015	POODLE	SSL 3.0, MitB + MitM	partiel
2015	RC4	sessions multiples	oui

# Références

## Méta-références

- <https://tools.ietf.org/html/rfc7457>
- <https://www.ietf.org/proceedings/89/slides/slides-89-irtfopen-1.pdf>
- <https://eprint.iacr.org/2013/049.pdf> section III

## Références spécifiques complémentaires

- <https://www.openssl.org/~bodo/ssl-poodle.pdf>
- <https://www.rc4nomore.com/>
- [http://www.imperva.com/docs/HII\\_Attacking\\_SSL\\_when\\_using\\_RC4.pdf](http://www.imperva.com/docs/HII_Attacking_SSL_when_using_RC4.pdf)