

# Soutenance projet annuel - Audit des implantations SSL/TLS

Claire Smets – William Boisseleau – Pascal Edouard –  
Mathieu Latimier – Julien Legras

Master 2 Sécurité des Systèmes Informatiques

28/02/2014



# Sujet et problématique

## Les clefs qui rôdent sur internet

- clefs SSH (Port 22), SSL/TLS (Port 443)
- taille des clefs + génération aléatoire

```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
              // guaranteed to be random.  
}
```

# Sommaire

## 1 Introduction

- Sujet et problématique

## 2 Audit des clefs RSA des certificats

### ■ Récupération

- Adresses
- Certificats

### ■ Factorisation

### ■ Résultats

## 3 Audit d'OpenSSL

### ■ Entropie

### ■ Entropie

### ■ Génération des clefs

### ■ Chiffrement et protocoles

### ■ Signature et authentification

### ■ Protocole SSL/TLS

### ■ Ouverture

## 4 Analyse dynamique du navigateur client

### ■ Faiblesses identifiées

### ■ Implémentation

### ■ Qualis

### ■ Démonstration

## 5 Conclusion

# Récupération des adresses 1

## ZMAP

- open source ;
- outil de scan réseau ;
- adresses IPv4 ;
- paquets SYN sur le port 443.

# Récupération des certificats 1

## Application Récupération de Certificats

- script perl ;
- certificats SSL ;
- stocker l'ensemble des empreintes dans un dossier :
  - certificats ;
  - clefs de session.

# Récupération des certificats 2

## Algorithme

**Entrées:** Fichier *f*, contenant les adresses ayant le port 443 ouvert

**Sorties:** Certificats et clef de session des adresses

**Données:** log, certificat, clef de session

**pour tous les adresses de *f* faire**

    Se connecter au serveur;

**si echec alors**

**si le log contient protocol alors**

            on incrémente le nombre d'échecs de protocoles;

**fin**

**si le log contient handshake alors**

            on incrémente le nombre d'échecs de poignées de mains;

**fin**

**sinon**

        On capture la session (dont le certificat serveur);

**si lechec de capture de session alors**

            on extrait le certificat et la clef de session;

**fin**

**fin**

**fin**

**retourner** certificats et clés de session

# Certificats récupérés

## Failles

- pas de Comon Name ;
- que le Serial Number.

# Gestion des doublons 1

## Gestion des doublons

- script perl ;
- si une empreinte de trouve déjà dans le dossier : on la stocke dans un autre dossier, celui des doublons ;
- liens symboliques.



# Gestion des doublons 2

## Algorithme gestion des doublons

**Entrées:** Pré-requis : Exécution de l'algorithme `ssl_collector`

Création d'un dossier `D` contenant les certificats à traiter

**Sorties:** Dossiers : `certs_doublons`, `certs_links`; Fichier : `moduli`

**Données:** Certificats `C`; Fingerprint `F`; Chaîne de caractères `S`; Modulo `M`

`certs_doublons` ← dossier\_vide;

`certs_links` ← dossier\_vide;

`moduli` ← fichier\_vide;

**pour tous les** `C`  $\in$  `D` **faire**

`F` ← *fingerprint*(`C`);

`S` ← *nom\_fichier*(`C`);

`M` ← *modulo*(`C`);

**si** *échec*(`F`) **alors**

        continue;

**fin**

**si** `F`  $\in$  `certs_links` **alors**

`certs_doublons/F` ← *concat*(`certs_doublons/F`, `S`);

**sinon**

`moduli` ← *concat*(`moduli`, `M`);

`certs_links` ← `F`;

**fin**

**fin**

# Factorisation

PGCD deux à deux

$N_1$

$N_2$

$N_3$

$N_4$

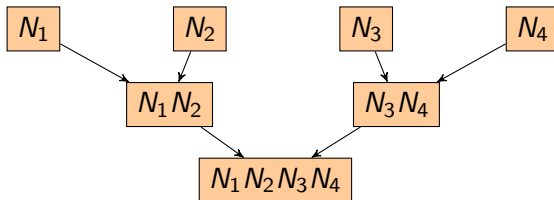
# Factorisation

## PGCD deux à deux



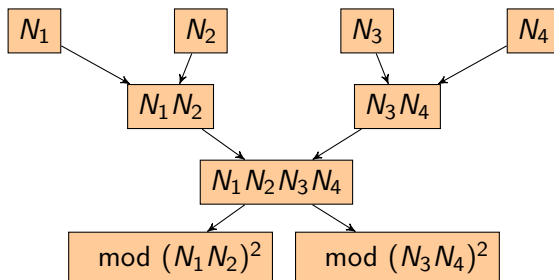
# Factorisation

## PGCD deux à deux



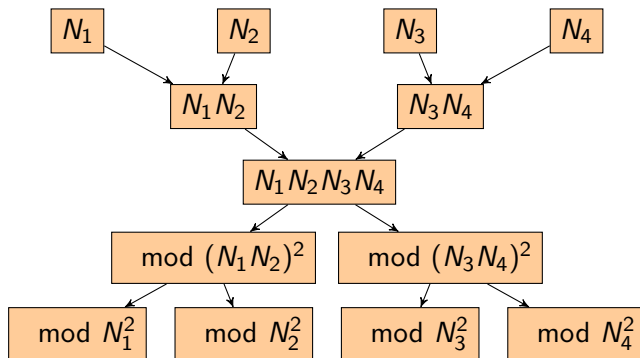
# Factorisation

## PGCD deux à deux



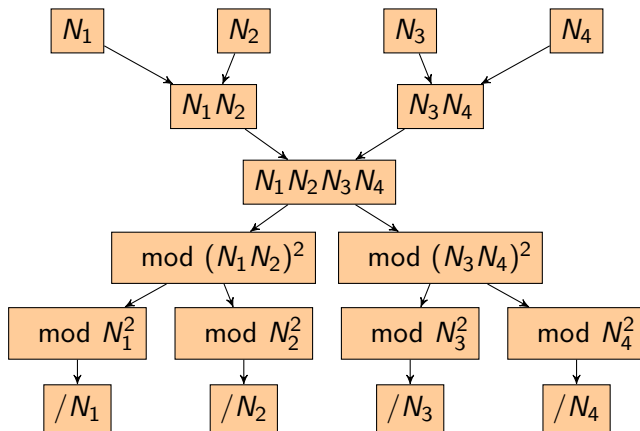
# Factorisation

## PGCD deux à deux



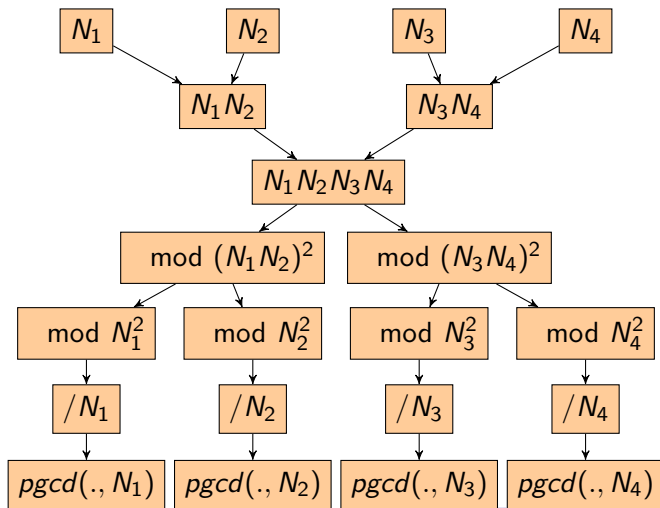
# Factorisation

## PGCD deux à deux



# Factorisation

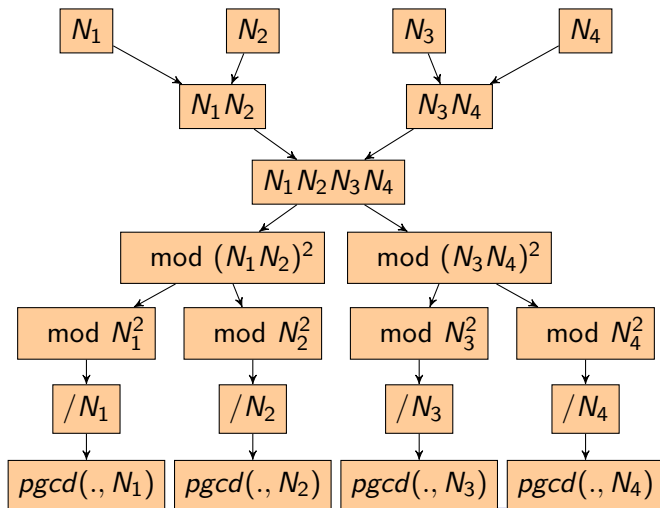
## PGCD deux à deux





# Factorisation

## PGCD deux à deux



# Factorisation

## Exemple

$$2 * 3$$

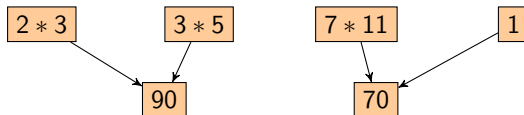
$$3 * 5$$

$$7 * 11$$

$$1$$

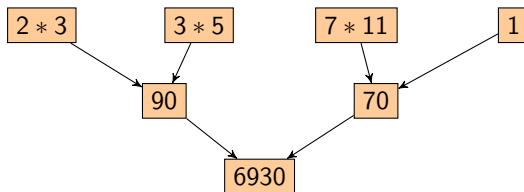
# Factorisation

## Exemple



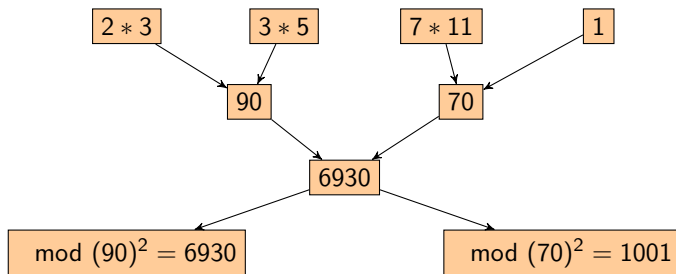
# Factorisation

## Exemple



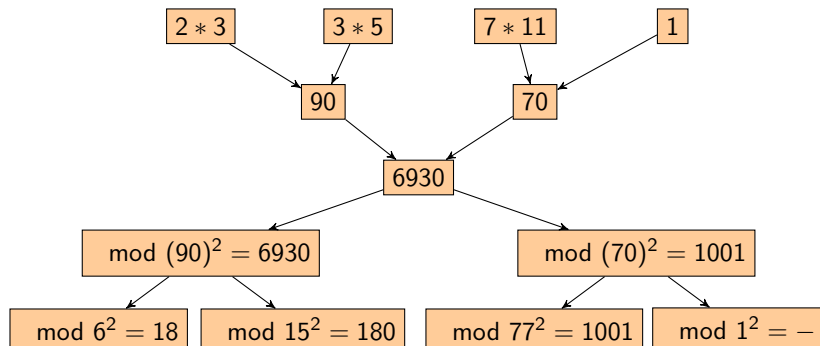
# Factorisation

## Exemple



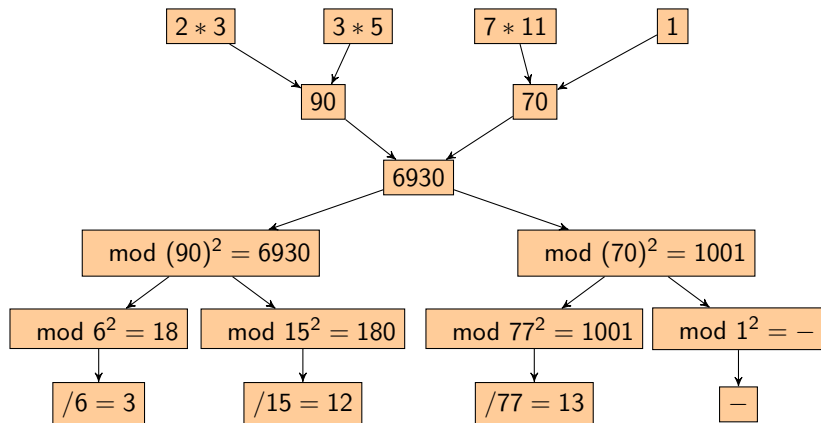
# Factorisation

## Exemple



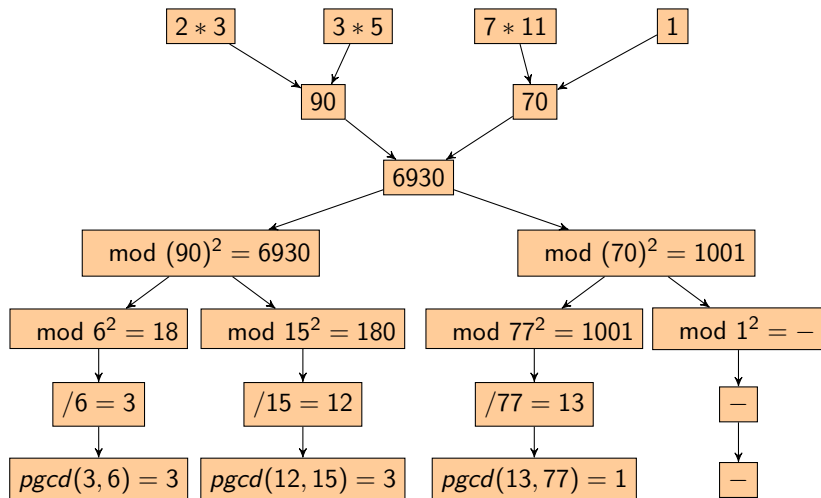
# Factorisation

## Exemple



# Factorisation

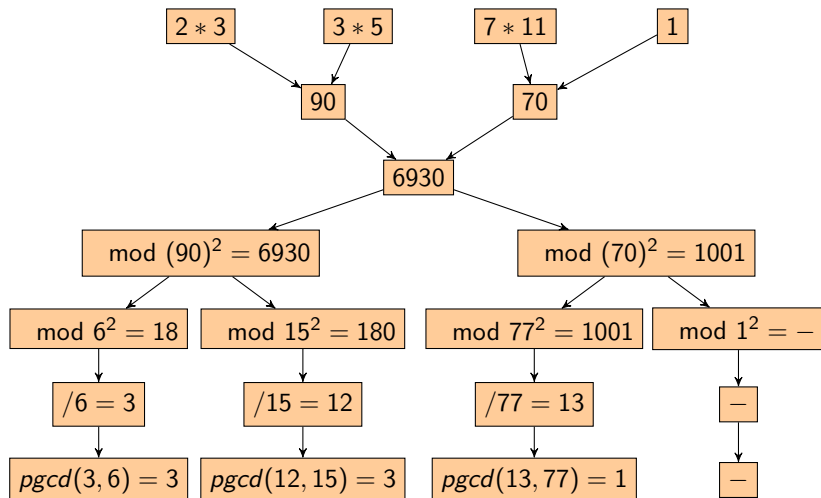
## Exemple





# Factorisation

## Exemple



# Factorisation

Algorithmes associés : Arbre des produits, Arbre des restes

**Entrées:** tableau des moduli :  $T$

**Sorties:** Hauteur arbre, produits des  
moduli

$v \leftarrow T;$

$level \leftarrow 0;$

**tant que**  $|v| > 1$  **faire**

$tmp \leftarrow \emptyset;$

**pour chaque**  $i \in \{0, \dots, |v|/2\}$  **faire**

$tmp[i] \leftarrow v[i \times 2] \times v[i \times 2 + 1];$

**fin**

$storeProductLevel(v, level);$

$v \leftarrow tmp;$

$level \leftarrow level + 1;$

**fin**

**retourner**  $level$

# Factorisation

Algorithmes associés : Arbre des produits, Arbre des restes

**Entrées:** tableau des moduli :  $T$

**Sorties:** Hauteur arbre, produits des moduli

```

 $v \leftarrow T;$ 
 $level \leftarrow 0;$ 
tant que  $|v| > 1$  faire
     $tmp \leftarrow \emptyset;$ 
    pour chaque  $i \in \{0, \dots, |v|/2\}$  faire
         $tmp[i] \leftarrow v[i \times 2] \times v[i \times 2 + 1];$ 
    fin
     $storeProductLevel(v, level);$ 
     $v \leftarrow tmp;$ 
     $level \leftarrow level + 1;$ 
fin
retourner  $level$ 
  
```

**Entrées:** Hauteur de l'arbre :  $level$

**Sorties:** PGCDs des moduli

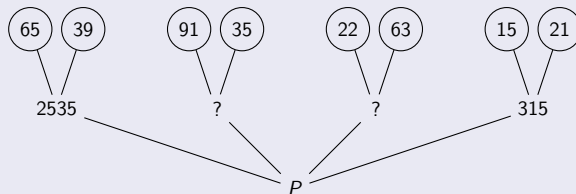
```

tant que  $level > 0$  faire
     $P \leftarrow getRemainderLevel(level);$ 
     $v \leftarrow getProductLevel(level - 1);$ 
    pour chaque  $i \in \{0, \dots, |v|\}$  faire
         $v[i] \leftarrow P[i/2] \pmod{v[i]^2};$ 
    fin
     $storeRemainderLevel(v, level);$ 
     $v \leftarrow tmp;$ 
     $level \leftarrow level - 1;$ 
fin
 $w \leftarrow \emptyset;$ 
pour chaque  $i \in \{0, \dots, |v|\}$  faire
     $w[i] \leftarrow P[i/2] \pmod{v[i]^2};$ 
     $w[i] \leftarrow w[i]/v[i];$ 
     $w[i] \leftarrow pgcd(w[i], v[i]);$ 
fin
retourner  $w$ 
  
```

# Optimisations I

## Parallélisation

En largeur avec 100 threads



## Parallélisation

En hauteur : ralentissement du traitement à cause de la dépendances entre les niveaux

# Optimisations II

## Langage et bibliothèque

C & *libGMP*

## Stockage

- programme initial : utilisation de fichiers pour stocker chaque niveau ;
- programme amélioré : stockage de l'arbre des produits en RAM (4-5 Go pour 500 000 clefs).  
⇒ jusqu'à 10 fois plus rapide même avec un SSD

## Calcul des carrés

Préférer la fonction d'exponentiation de GMP plutôt que la multiplication ⇒ jusqu'à 5 fois plus rapide

# Factorisation – Démonstration

# Résultats

## Base de donnée

- Récupération des facteurs communs - clefs publique
- Scripts perl + requêtes SQL

# Résultats

## Base de donnée

- Récupération des facteurs communs - clefs publique
- Scripts perl + requêtes SQL

## Statistiques

- Les émetteurs
- Outil de recherche
  - taille de clef : 99.6% (R) - 0.4% (V)
  - émetteurs, sujet : CISCO - 6280 (R) - 40 (V)



# Introduction

## Contexte

- récent scandale sur la NSA ;
- beaucoup d'outils utilisés.

Mais à qui peut-on faire confiance ?

## Audit d'OpenSSL

Cinq grands axes :

- l'entropie ;
- la génération des clefs ;
- le chiffrement et les protocoles ;
- les signatures et les authentifications ;
- les protocoles SSL et TLS.

# Failles

## Où ?

- site des CVE ;
- site de Vigil@nce ;
- RFC.

## Quand ?

- année passée ;
- failles plus anciennes.

## Corrections ?

Présence de corrections ? Si oui, lesquelles et par qui ?

# Entropie

## Définitions

### Générateur

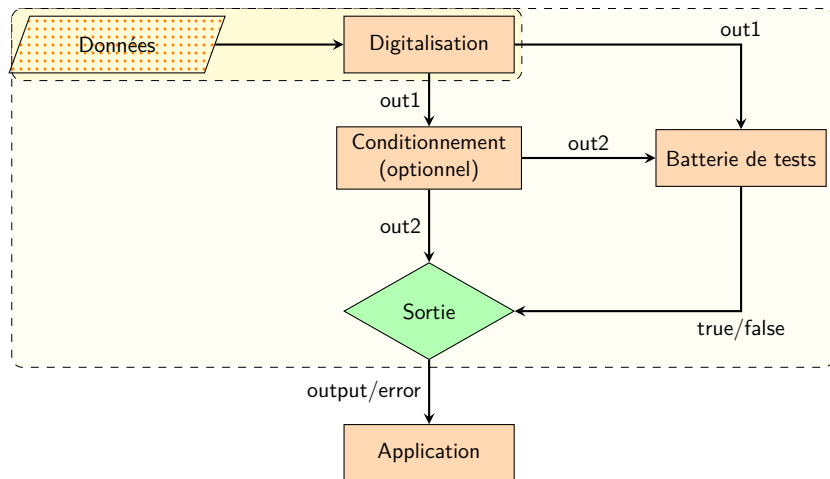
- problème aléatoire ;

- mesure de l'entropie

$$H(X) = - \sum_x P(X = x) * \log_2(P(X = x)) ;$$

- entropie et moduli.

# Entropie



# Entropie

## Tests

- Tests trop anciens, FIPS1 (1994), non mis à jour
  - test monobit
  - test poker
  - test runs
  - test long runs
- Autres tests proposés dans le rapport

# Entropie – Démonstration

## Faible Debian 4.0 sous OpenSSL 0.9.8

Après avoir récupéré l'ensemble des certificats sur l'Internet, on peut identifier rapidement ceux qui ont été générés durant la faille Debian/OpenSSL entre 2006 et 2008.

- **Durée de l'attaque** : quelques heures
- **Conséquences** : forger de faux certificats, de fausses signatures et déchiffrer des messages privées.
- **Qui ?** : grandes entreprises (e.g. IBM, CISCO), routeurs, universités, etc.
- **Fin de validité de certificats** : Entre 2020 et 2030.

# Génération des clefs

## Principes de Kerckhoff

- Le *secret* réside dans la clef ;
- Les algorithmes de génération de clés ne doivent donner aucune information sur la clé.

## Deux grands types de générateur de clés

- Générateurs de bits aléatoires (RNG) pour les clés privées de certains algorithmes (i.e. AES, DSA) ou pour le salage (i.e. *seed* de RSA)
- Générateurs de clefs asymétriques, qui utilisent des fonctions à sens uniques, largement diffusées et ne devant délivrer aucune information sur le secret.

# Génération des clefs

## Audit : Diffie-Hellman Ephémère en mode FIPS

- **Description** : Un attaquant écoutant une communication chiffré en SSL/TLS entre un client et un serveur peut déchiffrer tout les messages en forçant la génération d'un secret Diffie-Hellman prédictible.
- **Comment ?** : En modifiant le trafic réseau par exemple.
- **Pourquoi ?** : L'activation du mode FIPS ne rejette pas les paramètres P/Q faibles pour les algorithmes EDH/DHE.
- **Où ?** : Dans `crypto/dh/dh_key.c` une partie de code génère un faux positif dans certains cas (sur une condition de test).
- **Solution** : Logiciel *Nessus Vulnerability Scanner* pour tester la configuration des serveurs.



# Chiffrement et protocoles 1

## RSA-OAEP

laïus RSA-OAEP (fonctionnement, +, -)

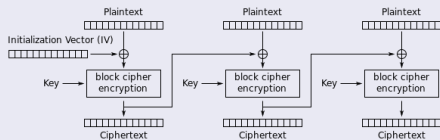
# Chiffrement et protocoles 2

## Manger's attack

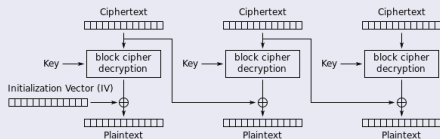
- OpenSSL 1.0.0 ;
- OAEP : défaillant ?
- contrôler la taille des paramètres à hacher ;
- sur serveur : variations de délais ;
- systèmes embarqués : plus problématique.

# Chiffrement et protocoles 3

## CBC



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

# Chiffrement et protocoles 4

## *Man in the Middle*

- attaque à clair choisi ;
- récupérer cookies de session.

## Problème

- SSL/TLS chiffre un canal de session ;
- problème d'IV.

## Recommandations

- ne pas utiliser CBC ;
- concaténer tous les objets.

# Signature et authentification

## Définition

- Juridiquement, une signature électronique à même valeur qu'une signature manuscrite.
- Elle **DOIT** assurer l'intégrité, l'authentification et la non-répudiation d'un message.

Des anciennes versions d'OpenSSL ont des vulnérabilités au niveau de la vérification de messages signés, ou des fuites d'informations sur la clé privée ayant servi à chiffrer.

# Signature et authentification

## Audit : Attaque par injection de fautes sur les certificats RSA.

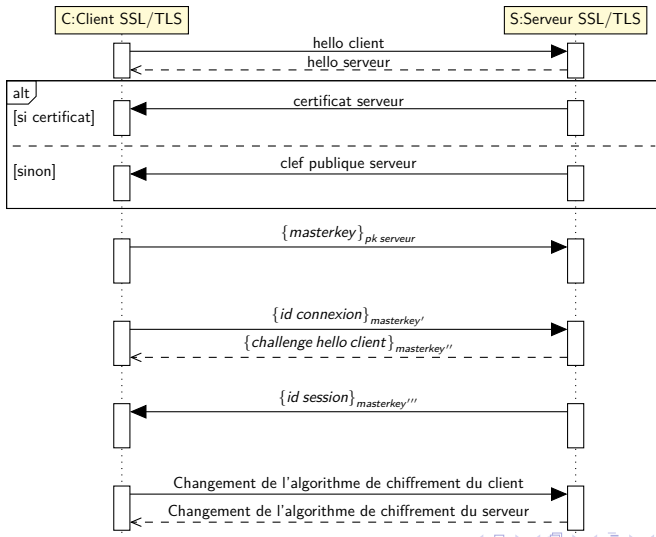
- **Description** : L'attaque se fait sur des morceaux de la signature récupéré afin de récupérer la clé privée bit à bit.
- **Comment ?** : Du bon matériel, surtout au niveau de la mémoire vive (i.e. Système Linux avec une architecture SPARC) et un oracle (e.g. système de prédictions) permettant de fabriquer la clé.
- **Temps de l'attaque** : une centaine d'heure.

# Signature et authentification

## Audit : Attaque par injection de fautes sur les certificats RSA.

- **Un problème dans le code OpenSSL ?** : La fonction `Fixed_Window_Exponentiation` utilise des milliers de multiplications, qui est opération la plus sensible en cas de dégradation du micro-processeur.
- **Solution** : Aucune ! On pourrait utiliser la technique du `square_and_multiply` mais elle a l'inconvénient d'être vulnérable à une attaque par *timing*.
- **Conséquences** : A moins que l'attaquant n'ai accès physiquement à votre machine les risques sont faibles. Cependant l'Université du Michigan cherche un moyen de faire des injections à distance à base d'impulsions lasers.

# Protocole SSL/TLS I





# Protocole SSL/TLS II

## Historique

- SSL 1.0 et 2.0 (1995) conçues par Netscape, 3.0 (1996) par l'IETF ;
- TLS 1.0 (1999) : légère amélioration de SSL 3, ajout d'extensions ;
- TLS 1.1 (2006) : protection contre les attaques contre CBC ;
- TLS 1.2 (2008) : remplacement MD5/SHA1 par SHA256, ajout des modes GCM et CCM.

# Faibles notables d'OpenSSL

SSL 3 et TLS 1.0 :

- 2002 - attaque sur le padding du mode CBC par Serge Vaudenay (reprise par Lucky Thirteen) : *timing attack* ;

TLS 1.1 :

- **2011** (CVE-2011-4576) - récupération d'informations sur un échange précédent : mémoire non réinitialisée ;
- **2013** (CVE-2013-0169)- Lucky Thirteen : *timing attack*,  $2^{23}$  sessions TLS pour retrouver un bloc en clair ;

TLS 1.2 :

- **2013** (CVE-2013-6449) - déni de service en envoyant une structure mal formée causant un *crash* dans la fonction `ssl_get_algorithm2`

# Ouverture

## Alternatives

- CyaSSL ;
- GnuTLS ;
- MatrixSSL ;
- Network Security Services (Firefox) ;
- PolarSSL (pas de support du DTLS) ;
- Java Secure Socket Extension (pas de support du DTLS).

## Attention – 25 février 2014

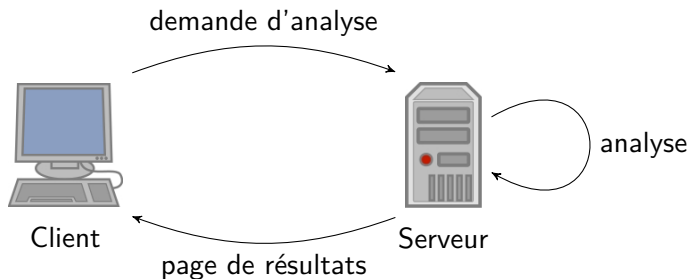
CVE-2014-1959 dans GnuTLS : validation de certificat (X509 version 1) intermédiaires comme un certificat d'AC par défaut.

# Faiblesses identifiées I

## Critères

- version du protocole
- *ciphersuites* proposées par le client
- courbes elliptiques supportées
- algorithmes de signature
- compression TLS
- activation ou non du ticket de session

## Faiblesses identifiées II

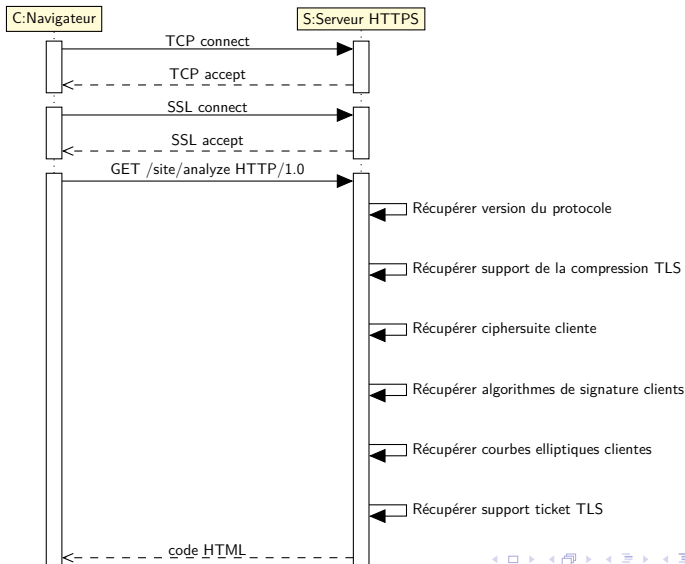


# Implémentation I

## Architectures possibles

- 1 journalisation de la connexion HTTPS avec une sonde IDS puis récupération des informations avec un langage tel que PHP → couche réseau ;
- 2 serveur HTTPS avec *libssl* puis analyse de la structure de la session → couche applicative.

# Implémentation II



# Qualis SSL Labs

## Description

Qualis est une plateforme multi-sécurité :

- Sécurité Réseau
- Sécurité Web
- etc

Le projet SSL Labs permet de :

- Tester l'implémentation SSL du navigateur.
- Tester la configuration et les certificats d'un serveur.
- Tester la configuration et les certificats d'un serveur.
- Lister les bonnes pratiques sur les déploiements SSL/TLS.



# Qualis SSL Labs

## Analyse des serveurs

L'analyse porte sur les points suivants :

- Contrôle du certificat
- Protocoles supportés
- Échange des clés
- Qualité de chiffrement
- Système de notation

# Qualis SSL Labs

## Tests sur un certificat vulnérable

SSL Report : spXXXX.XXXX.net (85.XXX.XXX.33)

- Notation : F
- Certificat non sûr → Non-confiance dans la chaîne de certification
- Sensible à une attaque de type CRIME
- Protocoles obsolètes →  
TLS\_RSA\_EXPORT\_WITH\_RC4\_40\_MD5
- Taille de clé insuffisante → RSA 1024 bits
- Re-négociation sécurisée non supportée.
- Compression TLS non-sécurisée
- Généré en Janvier 2014 - Expire en Janvier 2038

# Démonstration

## Analyse de la sécurité des navigateurs clients

- Sous le navigateur graphique Chrome : le plus utilisé dans le monde.
- Sous le navigateur console lynx : apprécié chez les développeurs.

## Modification manuelle

Nous pouvons modifier la *ciphersuite* du client avec la commande `s_client`.

# Conclusion

## Bilan

- Mise en évidence des clefs faibles
- Évolution du code OpenSSL