PJI Architecture TOR sur navigateur web

Franquenouille Kevin
Cornette Damien
Git Repository : `https://github.com/kfranquenouille/PJI-TOR.git`

Année universitaire : 2014-2015

Introduction

Tor est un projet pour le support de l'anonymat en dissimulant les communications entre un client web et un serveur. Le mécanisme s'appuie sur un ensemble de relais de confiance disposés sur des serveurs.

Le nombre limité de relais (moins de 5000) et la confiance dans leur non-compromission peut être considéré comme une limitation de l'approche de Tor. Un moyen d'outrepasser cette limitation serait de limiter le rôle du serveur a un simple relais et de laisser les clients gérer l'anonymat dans le système.

Afin de rendre le système le plus pratique et adoptable pour multiplier les chemins possibles, une idée serait de ne s'appuyer que sur des technologies webs (http, javascript) pour que chaque navigateur puisse devenir un client.

Le but de ce projet est d'étudier la faisabilité de l'approche à l'aide de technologie comme websocket. Le cas d'étude sera limité à un simple chat (saisie limitée à 140 caractères).

Table des matières

1 Description générale du projet

1.1 Problématique

Le but de ce projet était de mettre en place une "simulation" du réseau TOR via un navigateur web. Pour cela, nous avons pu voir avec Julien Iguchi-Cartigni les technologies que l'on pouvait utiliser. De ce fait, on a pu découpé le projet en 2 parties :

- Web Cryptography API
- WebSocket API

Pour réaliser cela, nous devions réaliser cela avec un seul relais client pour debuter puis ensuite le faire avec 3 afin de bien vérifier l'encryption des messages et que les messages sont bien redirigés vers le prochain relais.

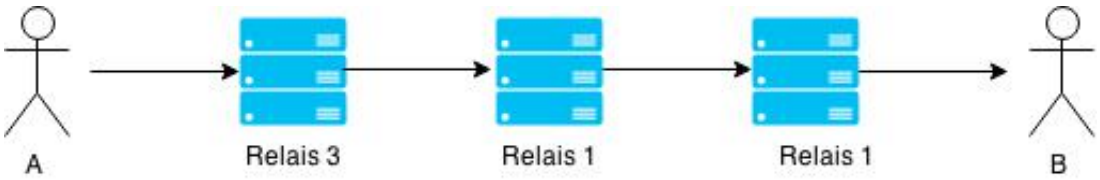

FIGURE 1 – Transfert d'un message de A vers B en passant par 3 relais

1.2 Web Cryptography API

Afin de bien crypter les messages envoyés et d'utiliser une technologies assez fiable et récente, nous avons privilégié cette API. En effet, elle est compatible sur presque tous les navigateurs (inclus de base). En revanche, il nous a fallu trouver un bon algorithme de chiffrement.
Dans un premier temps, nous avons utilisé le chiffrement AES-CBC afin de tester et découvrir Webcrypto API et réaliser un premier jet de ce que nous voulions faire. Une fois le messages encrypté 4 fois. Voici un exemple du cryptage :

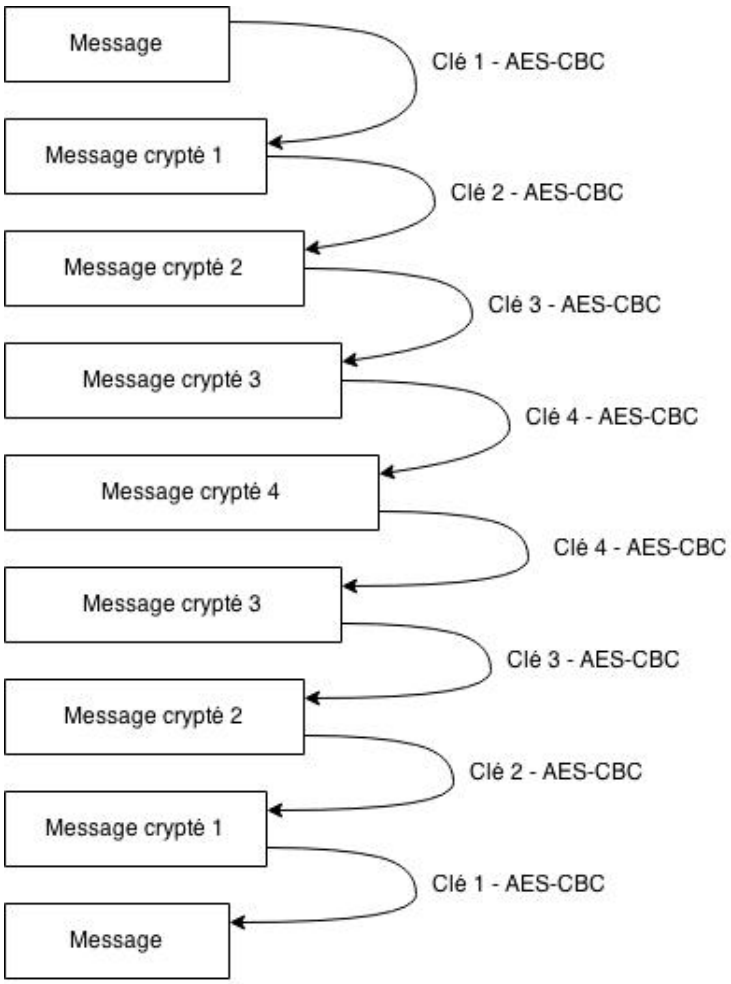

FIGURE 2 – Cryptage d'un message 4 fois de suite en AES-CBC

Pour la solution optimale, il est plus préférable d'utiliser un système de clés publiques et de clés privées. Avec l'algorithme RSA-OAEP, celà est possible. Chaque clé générée (l'objet KeyPromise) contient une clé publique et une clé privée. De ce fait, cette solution est plus attendu car elle ressemble très fortement à l'architecture du réseau TOR. Bien entendu, le client doit connaître le chemin afin qu'il puisse créer son message crypté avec les clés publiques des relais correspondants.

1.3 WebSocket API

1.4 Architecture de l'application

2 Mode d'emploi

2.1 Installer un relais

2.2 Configurer un relais

2.3 Configurer un client

2.4 Lancer l'application

3 Tests

3.1 Docker

3.2 Cryptographie

3.3 WebSocket

Conclusion