Face Key

Quentin GERARD, Louis L'HARIDON et Matthieu VILAIN
CMI Systèmes Intelligents et Communicants, Université de Cergy-Pontoise
mail@etu.u-cergy.fr

Résumé

abstract of the project

I. Réseau

A. Spécification

Pour le projet Face Key nous avons besoin de developper un serveur qui communique avec un client. Le serveur aura pour mission d'authentifier l'utilisateur et de fournir au client les données dont il a besoin au bon fonctionnement du logiciel, dans la limite de ce qui est accessible à l'utilisateur (un utilisateur ne peut avoir accès qu'à ses données). C'est le serveur qui assurera le lien avec la Base de données.

Le serveur devra :

- Authentifier l'utilisateur
- Creer un compte
- Fournir la combinaison Identifiant/Mot de passe pour un site donnée à l'utilisateur
- Recevoir les photos de l'utilisateur pour mise a jour du réseau de neurone
- Envoyer les poids du réseau de neurones à l'utilisateur

B. Fonctionnement global

Pour réaliser le Client/Server, nous avons choisi d'utilisé le protocole TCP, qui nous permet de nous assurer que chaque message à bien été transmis à l'utilisateur, et ne pouvant pas être utilisé en diffusion. Le serveur est également multi-client : un processus est créé, à partir du processus père, à chaque connexion au serveur. Voici le fonctionnement globale du Client/Serveur de Face Key :

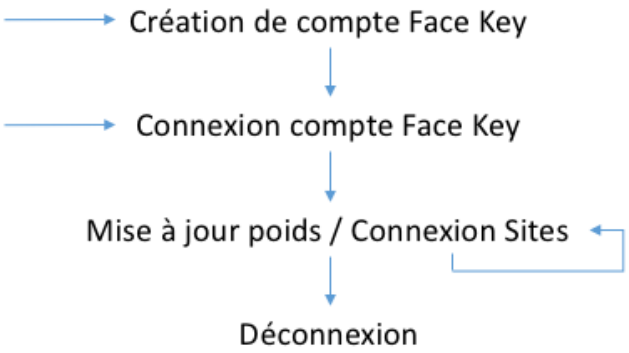

Figure 1. Fonctionnement globale de Face Key

C. Les requêtes

Une requête est constitué de 2 parties :

- Le code de la requête (voir tableau ci-dessous)
- Les informations

Dans la requête, les informations sont séparées du code de la requête par un ";". Ainsi la requête est composé de la manière suivante : | CODE | INFORMATIONS | Les informations sont quant à elles séparé par une virgule, nous

avons donc par exemple : | 100 | toto,facebook.com | Cela donne en pratique ”100 ;toto,facebook.com”.

Liste des codes :

Code Diagramme (000-099)	000	OK
	001	Code pour démarrer le diagramme de connexion à un site web
	002	Code pour démarrer le diagramme de création d'un compte
	003	Code pour démarrer le diagramme de mise a jour du réseau de neurones
Client -> Serveur (100-149)	100	Demande la liste des IDs disponible pour un utilisateur sur un site à l'instant t
	101	Demande le mot de passe pour un ID (accessible par l'utilisateur) sur un site
	102	Envoie d'une photo de l'utilisateur ainsi que ses coordonnées GPS
	103	Envoie de l'identifiant et du mot de passe Face Key pour authentification
	110	Envoie de l'email et du pseudo pour création du compte Face Key
	111	Envoie du mot de passe pour création du compte Face Key
	112	Envoie des informations de l'utilisateur pour création du compte Face Key (Genre, Nom, Prénom et Langue)
	113	Signal le serveur de l'envoie d'une photo
	114	Signal le serveur de l'envoie de la dernière photo
	136	Demande l'envoie du fichier des poids du réseau de neurones aux serveur
Serveur -> Client (200-249)	200	Envoie la liste des IDs disponible pour un utilisateur pour un site à l'instant t
	201	Envoie le mot de passe pour un ID (accessible par l'utilisateur) sur un site
Erreurs côté serveur (400-499)	400	Requête inconnue
	401	Il manque des informations dans la requête reçu
	402	Utilisateur introuvable lors de la phase d'authentification : Mauvais Identifiants
	403	Utilisateur introuvable lors de la phase d'authentification : Mauvais mot de passe
	405	Timeout Reached : l'utilisateur (ou le serveur) a mis trop de temps à répondre
	406	La requête reçu n'est pas pris en charge à ce point de l'application
	407	Aucun compte n'a été trouvé sur le domaine envoyé par l'utilisateur
	408	Le compte demandé pour le domaine n'est pas dans la liste des compte accessible par l'utilisateur
	409	Création du compte : l'email est déjà utilisé
	410	Création du compte : le pseudo est déjà utilisé

D. Diagrammes applicatifs

Dans cette partie nous allons présenter les différents diagrammes applicatifs du Client/Server de Face Key.

1) Création d'un compte: Pour démarrer ce diagramme il faut préalablement envoyer au serveur le code diagramme correspondant (code 002 pour la création de compte).

Ici les photo et les poids du réseau de neurone sont envoyer sous forme de fichier. Les fichiers sont ouvert par les deux programmes (client et serveur) et envoyer (et reçu) octet par octet sans code en entête. Le transfère d'un fichier est donc décomposé en plusieurs parties : tout d'abord les deux premiers message envoie le nom du fichier ainsi que le nombre d'octet qui va être envoyé (la taille du fichier), ensuite les octet sont envoyés un par un. Le receveur compte le nombre d'octet reçu et s'arrête une fois que l'intégralité des octet ont bien été reçu.

2) Phase d'authentification: Ici, le client envoie son identifiant et mot de passe Face Key, le serveur lui répondra par un code ”OK” si la combinaison est correct, ou par une erreur spécifiant quel partie de la combinaison est erroné.

3) Connexion à un site: Pour démarrer ce diagramme il faut préalablement envoyer le code diagramme (code 001 pour la connexion à un site) au serveur ***et*** *effectué une phase d'authentification (code 103).*

4) Mise à jour des poids du réseau de neurones: Pour démarrer ce diagramme il faut préalablement envoyer le code diagramme (code 003) au serveur ***et*** *effectué une phase d'authentification (code 103).*

Tout comme dans le diagramme de création d'un compte, les poids du réseau de neurones sont envoyés de la même manière : sous forme de fichier avec une transmission octet par octet.

II. Sécurité

A. Introduction au problème

Lors de l'utilisation de l'application, beaucoup de données transitent entre les différents blocs que composent l'architecture du logiciel, et/ou sont stockés dans le Base de Donnée. Parmis elles, se trouvent des informations

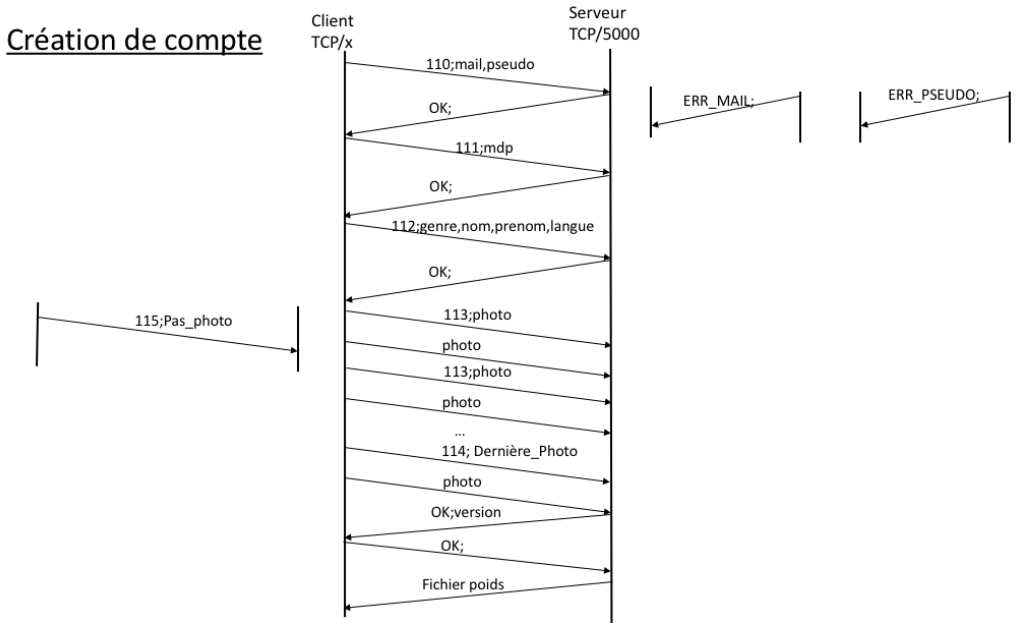

FIGURE 2. Création d'un compte

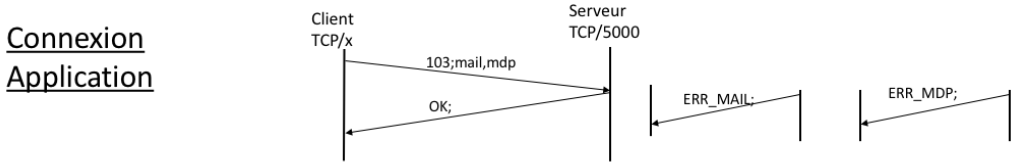

FIGURE 3. Authentification au compte Face Key

extrêmement sensibles concernant les utilisateurs. Pour un utilisateur, nous pouvons y trouver ces informations (liste non ordonnée et non exhaustive) :

- Adresses mails
- Mot de passe (de l'application Face Key)
- Combinaisons Identifiants/Mot de passe de plusieurs sites
- Coordonnées Bancaires
- Et d'autres

Si un utilisateur mal intentionné arriverait à intercepter les messages entre le client et le serveur, en se plaçant entre les deux grâce à un attaque basique de type Man in the Middle (MITM), alors il aurait accès à toutes ces données. De même si la base de donnée arrivait un jour à être compromise, et accessible par un utilisateur mal intentionné, alors il aurait également accès à toutes les données. Il nous faut donc trouver une solution qui nous permet à la fois de sécuriser les envoie de données Client/Server, et une solution qui nous permet de rendre illisible les informations dans la base de données à tous les utilisateurs autres que l'utilisateur concerné.

B. RSA

1) Introduction: Le RSA est un algorithme de cryptographie dit "asymétrique" (une clé privé et une clé publique, contrairement à la cryptographie symétrique qui utilise la même clé pour crypter et décrypter). Il a été rendu public par R.L. Rivest, A. Shamir, et L. Adleman dans un papier publié en 1977 appelé "*A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*". Il a pour but, à la fois de crypter des messages afin qu'ils puissent être transmis sans risquer d'être lu par une personne tierce, mais aussi de signer les messages. Un des principe du RSA est que la clé privé, permettant de décrypter les messages soit impossible à retrouver à partir de la clé publique, qui elle permet d'encrypter les messages. Nous avons donc décidé d'utiliser cette méthode afin de crypter les messages entre le Client et le Server.

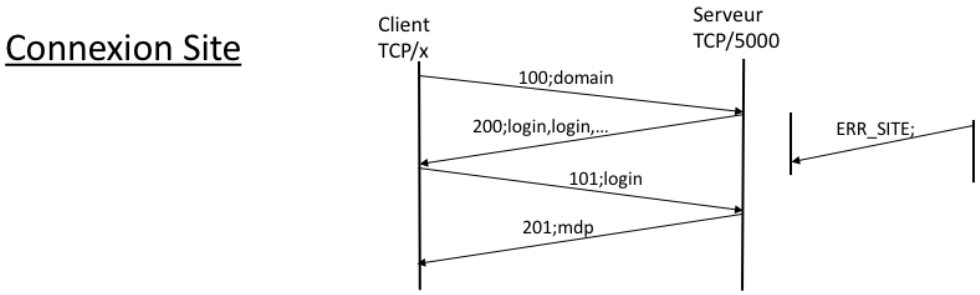

FIGURE 4. Connexion à un site web

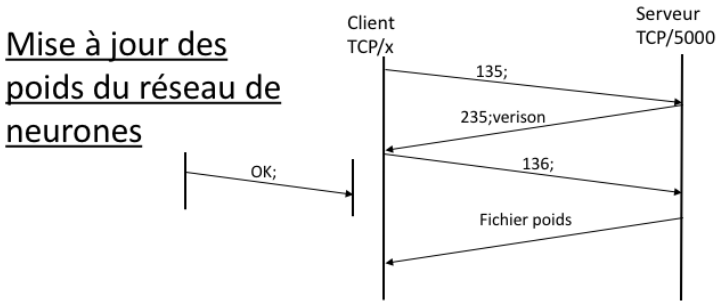

FIGURE 5. Mise à jour des poids du réseau de neurones

2) Génération des clés: La génération des clés appartient au receveur des messages, il doit ensuite communiquer la clé publique, qui n'est pas une information sensible, à l'envoyeur du message. Dans notre cas, le serveur et le client vont chacun générer de leur coté une pair de clés et transmettre à l'autre partie, la clé publique. Ainsi, même si les clés sont interceptés, n'importe qui pourra envoyer des messages au client et au serveur, mais personnes ne pourra lire les messages crypter envoyé, à l'exception du client et du serveur.

Pour créer les clés nous devons d'abord choisir deux nombres premiers p et q et ensuite calculer

$$n = p.q$$

Notons que n sera présent dans la clé publique il est donc important de trouver deux nombres premiers p et q assez grand pour qu'il soit difficile de décomposer n (Il est recommandé d'après les auteurs d'utiliser un nombre premiers composer de 100 chiffres), en effet chaque nombre n à une décomposition **unique** en facteur premiers. On cherche ensuite d tel que :

$$pgcd(d, (p-1).(q-1)) = 1$$

Nous venons donc de créer notre clé privé (d, n), pour trouver notre clé publique nous devons calculer e tel que :

$$e.d \equiv (mod(p-1).(q-1))$$

Notre clé publique est donc le couple (e, n).

3) Cryptage et Décryptage des messages: Pour crypter un message M nous avons besoin de notre clé publique (e, n), notre message crypter C se calcule de la manière suivante :

$$C \equiv M^e (mod\ n)$$

La fonction de décryptage est tout aussi simple, à partir de notre message crypter C , et de notre clé privé (d, n), nous calculons :

$$M \equiv C^d (mod\ n)$$

La méthode décrite içi est démontrée mathématiquement dans le papier d'origine de R.L. Rivest, A. Shamir, et L. Adleman.

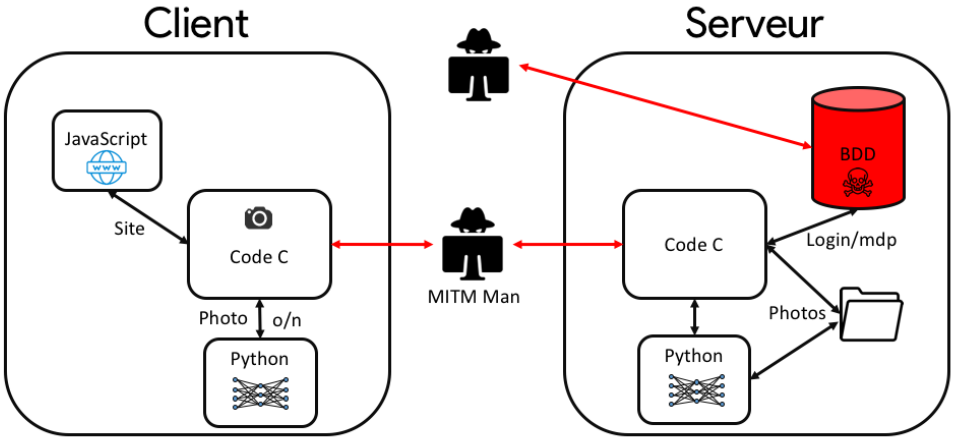

FIGURE 6. Problème de sécurité

4) Implémentation: Pour implémenter l'algorithme nous avons choisis d'utiliser la bibliothèque libre et open source OpenSSL, et plus particulièrement la bibliothèque *libcrypto* qui fourni les algorithmes de cryptographie. OpenSSL est une bibliothèque largement répandu et utilisé, il est donc plutôt aisé de trouver de la documentation dessus (suivant les fonctionnalités utilisées).

a) Génération des clés: Les clés publique et privés du serveur et du client sont généré préalablement, et stocké les les fichiers des deux programmes.

b) Échange des clés: Lors de chaque connexion du client au serveur, les clés publiques les deux parties sont échangé, envoyé sous forme de fichier "*.pem*" et stocké, avec les autres clés pour le client, et dans un dossier nommé par le PID du processus pour le serveur (le serveur étant multi-processus).

c) Cryptage et décryptage: Pour le cryptage et le décryptage, les fonctions de OpenSSL sont utilisées après avoir charger les clef dans la mémoire. Le cryptage et le décryptage est opérationnel pour peu que les deux clés issus de la même pair soit utilisées pour crypter et décrypter.

d) État de l'implémentation: Le système n'est que partiellement implémenté, en effet, malgré le bon fonctionnement de chacune des briques indépendamment des autres, nous pouvons observer un nombre non négligeable de raté lorsque nous essayons d'utiliser l'ensemble dans le contexte Client/Serveur. En effet, la transmission du message crypté ne se fait pas correctement (contrairement aux messages non crypté), il apparait effectivement que la somme de vérification du message crypté de l'une des partie, n'est pas la même que la somme de vérification du même message mais de l'autre côté. Nous pouvons en déduire que le message s'est mal transmis, cependant nous n'avons pas encore trouvé la raison de ce phénomène.

C. Fonctions de Hachages

1) Introduction: Une fonction de hachage cryptographique, est une fonction avec un message (ou fichier) en entré et une valeur en sortie appelé valeur de hachage, somme de vérification ou encore empreinte numérique. Le principe d'une fonction de hachage est de prendre le fichier en entré et de le transformé en une valeur (une chaine de caractère la plupart du temps) unique qui sera son "empreinte numérique". Une autre propriété importante de la fonction de hachage est qu'elle est très difficilement (voir presque impossible) inversible, c'est à dire qu'à partir de la valeur en sortie il est impossible (dans un temps raisonnable) de retrouver la valeur en entrée. De même une fonction de hachage idéal n'aurais aucune collision, c'est à dire que pour deux messages différents données ne pourrions pas avoir la même valeur en sortie.

2) Intégration au sein du projet: Au sein du projet Face Key, nous utiliserons une fonction de hachage (MD5) pour transformé les valeurs stocké dans la base donnée et régulièrement envoyé par l'utilisateur tel que : son mot de passe. Ainsi à chaque fois que l'utilisateur enverra son mot de passe, nous récupérerons la valeur hacher stocker dans la base, et nous transformerons le mot de passe envoyé et vérifierons les 2 sommes de vérifications. Si les 2 sommes sont les mêmes alors le mot de passe donnée est le même sinon le mot de passe est erroné. Cependant nous pouvons remarquer qu'une fonction de hachage n'est pas suffisante afin de garantir une sécurité satisfaisante. En effet, si la base venait à être compromise et lu par une personne tierce, il lui suffirait d'utiliser une liste des mot de passe les plus utiliser préalablement haché, et de seulement comparer les sommes de vérifications à celle dans la base donnée pour révéler plusieurs mot de passe. Sachant que 2 même mot de passe obtienne la même sortie cela diminue grandement le temps nécéssaire à retrouvé plusieurs mot de passe. De plus, il existe certaine méthode pour retrouver des mot de passe haché en clair, comme les attaques par dictionnaires ou Rainbow Table. Nous devons donc trouver une solution pour rendre la tache plus compliqué.

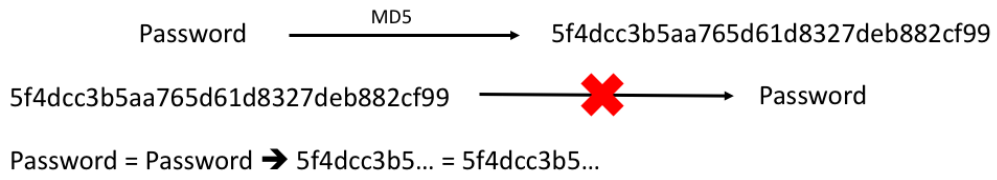

FIGURE 7. Principe du Hachage

3) Le Haching salé: Le principe du haching salé est simple : ajouter une chaine de caractère que l'on va concaténer au mot de passe avant de hacher la chaine final. Chaque utilisateur a un sel unique stocker sur la base de donnée. Ainsi 2 même mot de passe n'auront pas la même chaine haché. Cela rend les attaques par Brute Force, Dictionnaire et Rainbow Table, bien plus fastidieuse en augmentant le temps nécéssaire pour retrouver les mot de passes.

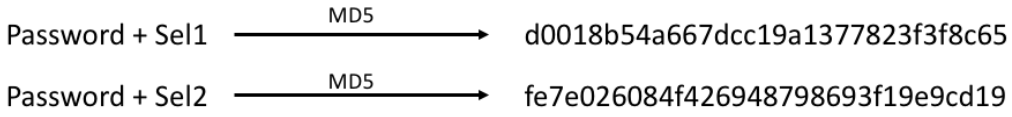

FIGURE 8. Hachage salé

D. Bilan de la sécurité

Pour commencer ce bilan, faisons un point sur ce qui est implémenté ou non. Nous avons vu que le RSA est partiellement implémenté, chaque bloc fonctionne de manière indépendante, mais la transmission du message crypté ne s'effectue pas bien. Pour ce qui est du hachage, seul le hachage classique est implémenté, le hachage salé n'est pas implémenté par manque de temps. En effet trouver d'où venait le problème du RSA nous à pris pas mal de temps, et nous ne pouvons à l'heure actuelle, estimer combien de temps il faudrais pour le résoudre, car nous ne connaissons pas la cause du problème. L'implémentation du Hachage Salé nécessiterait une modification de la base de donnée, pour stocké le "sel" de chaque utilisateur. De plus, il resterait d'autres problème de sécurité à régler, même si nous avions pu tout implémenté. En effet les combinaisons Identifiants/Mot de passe pour chaque site web d'un utilisateur ne sont pas protégé ! Il aurait aussi été interessant de crypter l'envoie et la sauvegarde des fichiers (images et réseau de neurones) sur le serveur, afin de préserver l'identité des utilisateurs.