Une “messagerie sécurisée” simplifiée

GS15 - A22 - Projet Informatique

Sujet présenté en cours le 08/11
Rapport et codes à rendre avant le 02/01
Soutenances entre le 03/01 et le 06/01

1 Description du projet à réaliser

L'objectif de ce projet informatique est de vous faire comprendre la conception d'un logiciel de messagerie sécurisée (inspirée du *protocole Signal*, utilisé dans *Signal*, mais aussi *WhatsApp*, *Messenger*, *Google Messages*, en option sur *Skype*, etc.)pour que vous puissiez l'implémenter entièrement. C'est une part importante de l'UE GS15 et plus généralement ce projet vous permettra de mieux comprendre certaines parties du cours.
Ce projet est conséquent, il nécessite beaucoup de travail, il est donc **fortement recommandé** d'y travailler le plus tôt possible ; si certains aspects n'ont pas encore été vus en cours, l'implémentation des algorithmes "de bases" (Inverse modulaire, exponentiation rapide, etc. ...) et les parties concernant le chiffrement symétrique sont possibles. Enfin, comme vous le verrez, la conception de ce logiciel a été modifiée légèrement afin d'inclure tous les aspects relatifs à la cryptographie vus en cours.
Je vous rappelle également que vous pouvez faire des recherches en lignes et vous aider de toutes les ressources à votre disposition (y compris des codes, des implémentations C, JS, Java du *protocoleSignal* sont disponibles sur github ...), mais vous devez les indiquer en références et indiquer explicitement si vous avez repris des portions de code ; le manquement à cette règle sera considéré comme une fraude à un examen.

2 Introduction

La conception d'un logiciel de messagerie sécurisée ou "cryptée" présente une particularité importante : elle doit pouvoir fonctionner de façon asynchrone. En effet, contrairement à d'autres cas d'usage, par exemple, la navigation sur Internet, le logiciel d'un messagerie doit prendre en compte le cas où les utilisateurs qui communiquent ne sont pas connectés en même temps ; Alice peut envoyer un message à Bob et ce dernier le recevra bien plus tard.
L'aspect asynchrone des logiciels de messagerie est en pratique géré par l'utilisation d'un serveur qui héberge les messages. Cependant, pour le cas d'un logiciel de messagerie sécurisée, ou souhaite s'assurer que même le serveur ne peut pas avoir connaissance du contenu des communications voire qu'il peut être compromis. Sur le plan fonctionnel, cela revient à considérer que le serveur n'est pas une entité de confiance.
Afin de pallier l'utilisation d'un serveur et de s'assurer que sa compromission ne peut compromettre les messages des utilisateurs, deux mécanismes essentiels sont mis en œuvre : le X3DH (eXtended Triple Diffie-Hellman) et le chiffrement assurant la sécurisation "dans le sens de la marche" (en bon Anglais le "forward secrecy"). Ces mécanismes sont respectivement décrits dans les sections 3 et 4.
La section 5 présente ensuite une vue globale de l'architecture de votre logiciel de messagerie sécurisé.

Pour approfondir, la section 6 vous présente certains aspects qui ne sont pas obligatoires, mais que vous pouvez regarder, comprendre en implémenter ; ces points sont utilisés dans les logiciels de messagerie sécurisée usuels.
Enfin, sur le plan de l'évaluation de vos projets, la Section 7 rappelle les consignes importantes à respecter.

3 Partie 1 : Initialisation des clés

Dans cette partie nous aborderons la partie d'initialisation du logiciel des comptes et des échanges. Cette partie repose essentiellement sur le X3DH. Pour sa présentation nous proposons de commencer par définir les éléments que vous devrez utiliser dans votre implémentation.

3.1 Notations et définitions

Dans votre projet, vous devrez utiliser une taille de clé de 2048 bits et une fonction de hashage de 512 bits. Nous rappellerons que l'échange de clé de Diffie Hellman procède de la façon suivante :

- Alice choisit un entier p (de 2048 bits dans notre cas) et un élément générateur g de $\mathbb{Z}_p$
- Alice choisit un entier a (clé privée) et calcule $A = g^a$ (clé publique) ; Elle envoie A à Bob
- Bob choisit un entier b (clé privée) et calcule $B = g^b$ (clé publique) ; Il envoie A à Bob
- Alice et Bob peuvent calculer leur clé partagée $S_K = A^b = B^a = g^{ab}$

Dans la suite, par souci de simplicité et de clarté, l'ensemble de ce protocole échange de clé sera noté $S_K = DH(a, B)$, la première clé étant toujours la partie privée, dont le résultat S_K correspond à la clé

partagée.

Chaque utilisateur possède un "lot" de couples de clés privées / publiques qui seront notées) :

- ID_A les clés d'identité de Alice, inchangeable et prévue pour une utilisation à long terme (i.e de talle au moins 2048 bits) ;
- $SigPK_A$ les "pré-clés" signées de Alice ; la de cette clé est notée $Sig(SigPK_A^{pub}; ID_A^{priv})$ ce qui signifie que la partie publique $SigPK_A^{pub}$ étant signées par la clé privée d'identité ID_A^{priv} ;
- $OtPK_A = (OtPK1_A, OtPK2_A, OtPK3_A, \ldots, OtPKN_A)$ les clés (couples de clés, privée / publique) de Alice ;
- Eph_A la clé (couple de clés, publique / privée) éphémère de Alice.

Dans les exemples ci-dessous nous avons arbitrairement choisit Alice, mais évidemment les rôles de Alice, Bob, Claire, Delphine et Eudes sont interchangeables ....
Dans ce projet, il vous ait demandé d'implémenter (au choix) la signature RSA ou la signature DSA.

3.2 Échange de clé asynchrone : X3DH (eXtended 3 Diffie-Hellman)

Alice et Bob publient chacun sur le serveur leur lot de clés constitué (pour Alice) de : $\{ID_A^{pub}; SigPK_A^{pub}; Sig(SigPK_A^{pub}; ID_A^{priv}); OtPK_A\}$;
Lorsque Bob contacte Alice, il reçoit ce "lot de clés" $\{ID_A^{pub}; SigPK_A^{pub}; Sig(SigPK_A^{pub}; ID_A^{priv}); OtPKn_A\}$.
Le serveur choisit aléatoirement une des clés à usage unique. Bien sûr, Bob commence par vérifier la signature de la "pré-clé" $Sig(SigPK_A^{pub}; ID_A^{priv})$ en utilisant la clé d'identité ID_A^{pub} (si cette vérification n'est pas concluante, le protocole s'arrête).
Le serveur de son côté supprime de la liste des clés à usage unique d'Alice celle envoyée à Bob ; bien sûr, Alice devra en fournir de nouvelles lorsque le serveur le lui demandera. Par ailleurs Alice doit (changer et) fournir au serveur une nouvelle "pré-clé" signée $SigPK_A$ avec la signature correspondante (de façon régulière, chaque jour, semaine, mois, ...).

Pour Bob, l'échange de clé par X3DH procède, après choix d'une clé éphémère privée (un grand entier choisit aléatoirement), en 4 étapes :

1. $DH1 = DH(ID_B^{priv}; SigPK_A^{pub})$;
2. $DH2 = DH(Eph_B^{priv}; ID_A^{pub})$;
3. $DH3 = DH(Eph_B^{priv}; SigPK_A^{pub})$;
4. $DH4 = DH(Eph_B^{priv}; OtPKn_A^{pub})$.

Après ces 4 étapes Bob peut calculer la clé partagée $SK = KDF(DH1||DH2||DH3||DH4)$. Il supprime alors sa clé privée éphémère et les résultats des échanges de clés $(DH1, \ldots, DH4)$ pour ne conserver que la clé partagée SK.

Ici, $KDF(\cdot)$ représente une "*Key-Derivation Function*", que vous êtes libre de choisir (voir précisions dans la section suivante 4).

Enfin, afin que Alice puisse calculer ensuite, lors de sa prochaine connexion la clé partagée SK, Bob lui envoie sa clé publique d'identité (que Alice peut vérifier en se connectant au serveur), la partie publique sa clé éphémère Eph_B^{pub} et l'identifiant / le numéro n de la clé à usage unique utilisée $OtPKn_A$.
Alice fera des opérations similaires (que je vous laisse déterminer, c'est facile) lors de sa prochaine connexion pour obtenir la même clé partagée SK.

4 Partie 2 : Communications sécurisées

Une fois que la phase d'initialisation des communications a été effectuée, Alice et Bob partagent une clé SK qui peut être utilisée pour échanger des messages.
Notez que dans ce projet on distinguera :

- les messages de type "texte" qui seront chiffrés avec une méthode de chiffrement de flux (que vous êtes libre de concevoir) ;
- les envois de "fichiers" qui seront chiffrés par bloc avec un algorithme reposant sur Feistel ou SP (à vous de décider) ;

La spécificité du *protocole Signal*, que vous devez implémenter, est que chaque message est chiffré avec une clé distincte. Cette façon de procédé, décrite ci-dessous, est appelée le "*double ratchet*" (le double "cliquetis") et permet d'assurer la "*forward secrecy*"[1].

4.1 Foward Secrecy : la confidentialité persistante, "dans le sens de la marche"

Le concept de "*forward secrecy*" est d'assurer la continuité de la sécurité même si une clé de communication est compromise. Pour cela, le principe mise en œuvre de clés qui "cliquètent" (en anglais "to ratchet") est assez simple et repose sur une chaîne de "Key-Derivation Functions" (KDF) :
La fonction de dérivation de la clé prend, en entrée, une "clé chaînée" (secrète et aléatoire) ainsi que des données ; les données retournées en sorties sont deux clés : (1) la clé de communication utilisée pour le chiffrement symétrique du message et (2) une "clé chaînée" qui sera utilisée comme entrée de l'itération suivante.
Bien sûr, il est impossible de retrouver la "clé chaînée" à partir de la "clé de message" et inversement. Aussi, si une clé de message est compromise, la suite des échanges de message de l'est pas.
L'illustration ci-dessous résume le concept de "Ratchet keys" dans le cas du chiffrement symétrique.

L'implémentation du "ratchet symétrique" est laissée à votre discrétion ; en pratique, dans le protocole Signal, la clé de message est calculée par en utilisant le H-MAC-SHA256 calculé avec la clé chaînée et la donnée $0x01$: $MessageKey = HMAC - SHA256(ChainKey, 0x01)$.
La clé chaînée est mise à jour en calculant le H-MAC-SHA256 avec la (même) clé chaînée et la donnée $0x02$: $ChainKey = HMAC - SHA256(ChainKey, 0x02)$.

[1] La "*forward secrecy*" peut se traduire littéralement par "confidentialité dans le sens de la marche" ; cette notion est généralement appelée "confidentialité persistante"

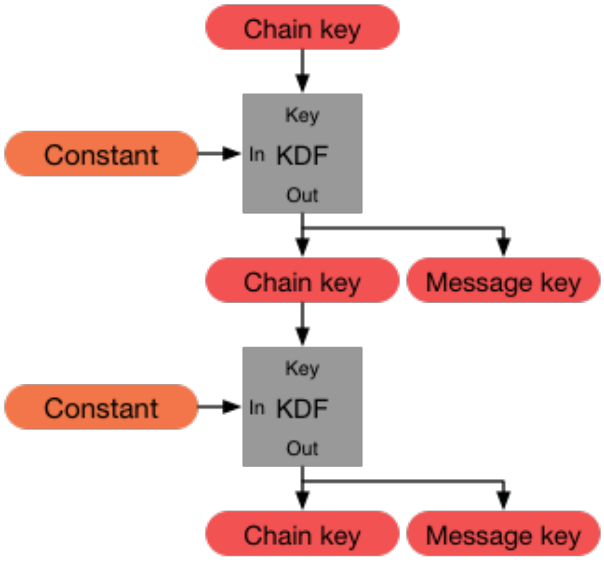

Figure 1: Illustration graphique de clés chaînées

La clé de message est en pratique constituée de trois éléments : (1) une clé de chiffrement symétrique (2) d'un vecteur d'initialisation du chiffrement symétrique, l'IV, et, (3) une clé pour signer le message à l'aide d'un HMAC.

4.2 Le Double Ratchet : symétrique et à clé publique

Le principe de "ratchet" décrit ci-dessus pour le chiffrement symétrique est également étendu avec du chiffrement à clé publique afin de modifier régulière les clés partagées entre Alice et Bob (on parle de sessions de communication).
Là encore le principe est assez simple : Alice et Bob procèdent par échange de clé en utilisation la méthode de Diffie-Hellman. Cependant à tout instant, Alice ou Bob pour modifier son couple de clés éphémères (publique /privée) en envoyant en entête d'un message la nouvelle clé publique utilisée.
La clé partagée par ce "Ratchet à clé publique" est utilisée dans une KDF afin de créer une nouvelle clé racine utilisée comme clé chaînée initiale dans un "ratchet symétrique".
Les figures 2 ci-dessous illustrent de façon simplifiée cette façon de modifier régulièrement les clés de session :

5 Le protocole de messagerie sécurisé (simplifié)

Le protocole de communication du programme que vous devez développer peut se résumer de la façon suivante, voir également la figure 3 ci-dessous:

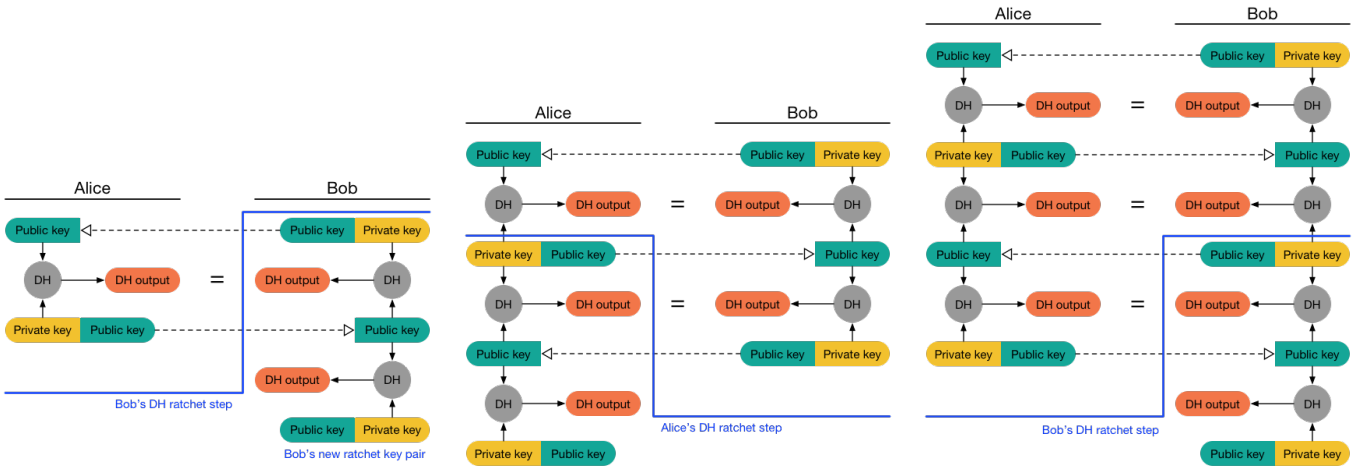

Figure 2: Illustration graphique des mises à jour des clés de sessions par “Ratchet à clé publique”.

1. Alice et Bob utilisent le X3DH pour initialiser la communication et partager une clé racine.
2. Alice et Bob utilisant des clés “éphémères” pour créer des sessions avec une clé racine utilisée comme première clé chaînée du “ratchet symétrique” ;
3. Alice et Bob peuvent s’envoyer des messages qui doivent être chiffrés (chiffrement symétrique), signés, numérotés et acquittés.

5.1 Les exigences auxquelles doit répondre votre projet

Il sera considéré que votre projet répond à toutes les exigences si les fonctionnalités suivantes sont toutes implémentées par vos soins:

- L’initialisation des clés et leurs échanges via X3DH ;
- La confidentialité persistante par “*double ratchet*” :
  Cette fonctionnalité doit, bien sûr, inclure une H-KDF de votre choix ;
- Le chiffrement de flux (de votre choix) pour les messages et le chiffrement par bloc pour les fichiers ;
- Il vous ait demandé d’implémenter une fonction éponge de hashage de 512 bits ;
- Vous devez également implémenter (au choix) la signature RSA ou la signature DSA.

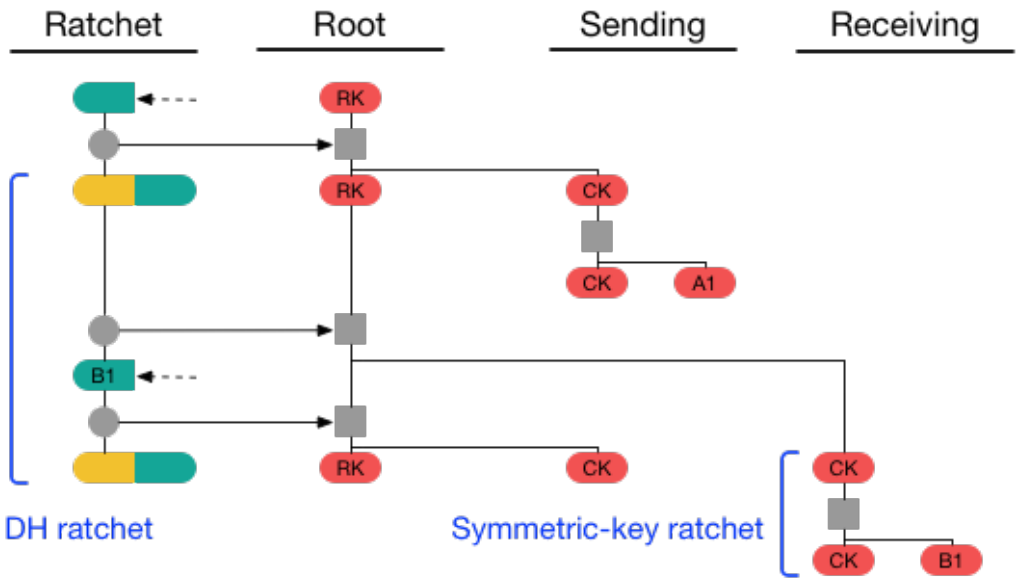

Figure 3: Illustration graphique des mises à jour des clés de sessions par “Ratchet à clé publique”.

6 Pour aller plus loin : quelques suggestions optionnelles

Le projet que je vous propose s’inspire du protocole Signal ; vous pouvez aller plus loin en consultant la documentation de whatsApp (qui implémente le protocole Signal) notamment :

- Le livre blanc de la sécurité dans l’application whatsApps
- La documentation technique du protocole Signal sur l’implémentation du “double ratchet”
- La documentation technique du protocole Signal sur l’implémentation du X3DH
- La documentation technique du protocole Signal sur l’algorithme générale de chiffrement par sessions

Les options que vous pouvez implémenter sont nombreuses, par exemple

- la gestion de plusieurs appareils pour chaque utilisateur ;
- La gestion des communications par groupe, avec plusieurs utilisateurs ;
- Vous pouvez implémenter un logiciel dans lequel deux instances python (deux terminaux) communiquent.

De façon générale, toutes amélioration est la bienvenue.

7 Exigences : questions pratiques et autres détails

Il est impératif que ce projet soit réalisé en binôme. Tout trinôme obtiendra une note divisée en conséquence (par 3/2, soit une note maximale de 13,5).
Encore une fois votre enseignant n'étant pas omniscient et afin de rendre le projet le plus facile pour vous, l'utilisation du langage python est imposée.
Par ailleurs, votre code devra être commenté (succinctement, de façon à comprendre les étapes de calculs, pas plus).
De même les soutenances se font dans mon bureau. Vous devez pouvoir exécuter votre code *python3* sur mon PC (*python2* possible mais désormais obsolète). Dans tous les cas (notamment si vous utilisez plusieurs bibliothèques, dont certaines non usuelles) amenez si possible votre machine afin d'assurer de pouvoir exécuter votre code durant la présentation.

Un rapport très court est demandé :

Par de formalisme excessif, il est simplement attendu que vous indiquiez les difficultés rencontrées, les solutions mises en œuvre, expliquer les fonctionnalités supplémentaire, ou au contraire celles manquantes, et justifier les choix que vous avez fait (par exemple : utilisation d'une bibliothèque très spécifique, quelle KDF, quelle fonction de hashage, quelles modifications ont été nécessaires, méthodes implémentée pour générer rapidement des clés publiques de grande taille, etc. ...). Faites un rapport très court (environ 2 pages) ce sera mieux pour moi, comme pour vous. Le rapport est à envoyer avec les codes sources.

La présentation est très informelle, c'est en fait plutôt une discussion autour des choix d'implémentation que vous avez faits avec démonstration du fonctionnement de votre programme.

Vous avez, bien sûr, le droit de chercher des solutions sur le Internet, dans des livres (en fait, où vous voulez), par contre, essayez autant que possible de comprendre les éléments techniques trouvés pour pouvoir les présenter en soutenance, par exemple comment trouver un entier premier sécurisé, comment trouver un générateur, etc. ...

L'utilisation de ressources qui ne seraient pas citées et indiquées explicitement sera considérée comme une fraude à un examen et sera transmise à la commission disciplinaire de l'UTT.

Enfin, vous pouvez vous amuser à faire plus que ce qui est présenté dans ce projet ... cela sera bienvenu, assurez-vous cependant de faire a minima ce qui demandé, ce sera déjà très bien.

Je réponds volontiers aux questions (surtout en cours / TD), mais je ne ferais pas le projet à votre place ... bon courage !