Un “mini” protocole de communication sécurisé et simplifié

GS15 - A23 - Projet Informatique

Rapport et codes à rendre avant le 07/01
Soutenances entre le 08/01 et le 12/01

1 Description du projet à réaliser

Le but de ce projet informatique est de vous faire créer une technologique similaire à une «mini PKI», *Public Key Infrastructure*, ou Infrastructure (de gestion) de clés publiques. L'idée étant de vous faire développer un outil permettant de mettre en œuvre des communications sécurisées entre deux utilisateurs.

La description du projet est volontairement «peu détaillée» afin **de laisser, dans ce projet, une très grande liberté quant à l'implémentation pratique des algorithmes, l'architecture du projet** et toutes les autres spécificités d'implémentation. Les contraintes et exigences que votre projet doit respecter sont explicitées dans la section 1.1.
La présence section introductive décrit le but et l'architecture générale du projet à réaliser.

La mise en place d'une communication sécurisée entre deux utilisateurs quelconques, sans aucune connaissance préalable, sur un réseau non sécurisé, nécessite les opérations suivantes :

1. Authentification de la personne : pour cela il faut recevoir un ***certificat*** qui, d'une part, doit être signé par une tierce personne et, d'autre part, doit pouvoir identifier la personne (dans la vraie, le nom de domaine, dans notre cas, le numéro de téléphone ou l'email ou autre ...).
2. Vérifier la validité du certificat en interrogeant un ***dépot*** de certificat et de clé publique.
3. Enfin, sécuriser la communication en utilisant (1) pour la confidentialité, des clés secrètes éphémères pour le chiffrement symétrique et (2) pour l'intégrité, une fonction de hachage (Hmac, Cmac ou simple Hash).

Il découle de la présente description des étapes que les protagonistes suivants seront impliqués :

1. Utilisateurs : leur rôle est simplement d'utiliser votre système ; attention pour cela ils doivent tout de même s'inscrire en générant un couple de clés publique / privée (1024 bits minimum) et en donnant un identifiant (email, téléphone,).
2. Autorité de confiance : c'est celle auprès de laquelle les utilisateurs s'enregistrent ; c'est également elle qui signe les certificats et enregistre ces données dans le ***dépot***.
3. Autorité de séquestre : c'est celle qui gère le ***dépot*** des certifications en y associant une date de validité. Dans notre cas simplifié, elle veille également à la révocation des certificats le cas échéant.

Pour aller plus loin, il vous ai également demandé d'implémenter les fonctions suivantes :

1. La communication sécurisée peut se faire en mode «asynchrone». C'est-à-dire que Alice peut échanger un message à Bob si même si ce dernier n'est pas connecté. Dans ce cas il faut pouvoir remédier à la phase de partage de clé secrète. Cette clé ne peut (évidemment !!) pas être directement envoyée avec le message que Bob consultera plus tard (sinon tout le monde notamment l'autorité de confiance) peut lire ce message. Il faut alors mettre en place un mécanisme de «partage de secret» de «partage asynchrone d'une clé secrète» entre Alice et Bob et faire en sorte que cette clé change.
2. Enregistrée de façon sécurisée des documents publics «non fongibles» dans une chaîne sécurisée. Cette chaîne doit être vérifiable par tous et tout le monde peut demander à y ajouter un document qui n'est pas nécessairement confidentiel.
3. De vérifier la clé publique d'un utilisateur sans compromission de la clé privée. Chaque utilisateur doit pouvoir demander à n'importe quel autre utilisateur de prouver qu'il a bien la clé privée qui correspond à la clé publique enregistrée par l'autorité de séquestre. Cela se fera soit par résolution d'un challenge (demande de chiffrement d'un message, protocole "Ali Baba" ou preuve de connaissance de Schnorr).

La très grande majorité des algorithmes sont décrits de façon très succincte, car ils ont été (ou seront) étudiés en cours. Les algorithmes un peu plus détaillés sont ceux demandés dans la section 1.1.
Encore une fois, vous devez répondre à certaines exigences et faire le minimum ou en faire beaucoup plus.

1.1 Exigences

Commençons par rappeler qu'un projet est un examen, toute fraude sera punie (en particulier la réutilisation / reproduction de sources, y compris entre groupes de GS15, sans les mentionner clairement). Toute utilisation de package python pour la cryptographie est prohibée ; les seules librairies autorisées sont celles de portée générale (*e.g.* os, bitarray, numpy, ...). **Toutes les bibliothèques de cryptographie sont interdites !**

Notons également qu'il vous ait demandé d'utiliser, pour le chiffrement symétrique, l'algorithme *SERPENT* qui ne sera pas vu en cours et, par ailleurs, qui est légèrement modifié par les soins de votre professeur pervers et sadique.

Il vous est demandé de faire un menu permettant de tester individuellement les algorithmes implémentés ; votre programme doit donc, typiquement, afficher lors de l'exécution le menu suivant :

```
Bonjour ô maître Rémi ! Que souhaitez-vous faire aujourd'hui ?
->1<- Chiffrer / déchiffrer des messages.
->2<- Créer un couple de clés publique / privée (générer un grand nombre premier).
->3<- Signer / générer un certificat.
```

```
->4<- Vérifier un certificat.
->5<- Enregistrer un document dans le coffre-fort.
->6<- Envoyer un message (asynchrone).
->7<- Demander une preuve de connaissance.
->0<- I WANT IT ALL !! I WANT IT NOW !! SecCom from scratch?.
```

Les algorithmes que vous devez développer, pour chacun des choix possibles, sont décrits ci-dessous (en ne commençant pas ceux que vous pouvez le plus implémenter le plus tôt dans le semestre) :
Choix `->1<-` : chiffrement et déchiffrement symétrique ?? pour la création d'un couple de clés publique / privée (Choix `->4<-`), ?? pour le hachage (Choix `->4<-` et `->5<-`) et la preuve de travail (Choix `->6<-`), ?? pour la génération et la vérification d'une signature (Choix `->7<-`).

Enfin, le principe de fonctionnement d'une blockchain est présenté dans la section ??. Cela vous sera utile autant pour votre "culture" personnelle que pour l'implémentation des opérations de création, incrémentation et vérification d'une blockchain (Choix `->8<-` et `->9<-`).

Il est conseillé de réutiliser les fonctions données / écrites pour les devoirs, notamment pour la lecture et l'écriture des fichiers, ainsi que les fonctions arithmétiques (tests de Rabin Miller, exponentiation rapide, etc. ...).

De nombreux points sont laissés à votre discrétion. En revanche, il y a également de nombreuses consignes à respecter. Ci-dessous sont rappelées les **principales consignes que vous devez obligatoirement respecter** :

1. réaliser votre projet en utilisant le **langage python** ;
2. **respecter les consignes** données dans la section 6 du présent document ;
3. **chaque section contient un paragraphe "exigences" que vous devez suivre** (par exemple utiliser des clés publiques/privées de 1024 bits au moins, écrire les transactions dans des fichiers, etc. ...) ainsi qu'une section "recommandations" dans laquelle des suggestions sont proposées pour aller plus loin.
4. **Vous devez rendre un rapport court**, **répondant uniquement (et pas plus)** aux exigences de la section 6 ; les soutenances auront lieu la soutenance précédent les finaux ; **vous devez réserver un créneau de soutenance**.

Enfin, je vous informe que la notation est faire afin que :
— un programme qui fonctionne et respecte l'ensemble des exigences se voit attribuer un 15/20 ;
— le respect, en sus, de l'ensemble des "recommandations" garantit un 20/20
— toutes les initiatives personnelles seront appréciées et valorisées (mais il est plus important de respecter les consignes)
— les projets de GS15 sont assez complets et chronophages ; **commencez en avance** et, si vous le faites bien, **utilisez les sur votre CV** pour montrer vos compétences dans le domaine de la crypto !

2 Chiffrement symétrique SERPENT

Nous décrirons ici une version légèrement simplifiée de l'algorithme de chiffrement symétrique par bloc SERPENT. Vous pouvez, naturellement, consulter les articles de références (notamment celui de wikipedia et l'article scientifique soumis pour l'appel à proposition du NIST pour la standardisation de AES.).

De façon globale, SERPENT est un chiffrement basé sur un réseau de substitution-permutation à 32 itérations ; lors de chaque itération (excepté la dernière), on applique successivement 3 opérations :

1. XOR avec une clé d'itération ;
2. une application de S-boxes (distinctes pour chaque itération) ;
3. une transformation linéaire.

En outre, une permutation initiale et une permutation finale (qui est l'inverse de la permutation initiale). Aussi, le chiffrement C d'un bloc clair M peut se résumer synthétiquement par les relations suivantes :

$$\begin{aligned} B_0 &= PI(M)\,, \\ B_{i+1} &= LT\Big[S_i(B_i \oplus K_i)\Big], \qquad \forall i \in \{0, \ldots, 30\} \\ B_{32} &= S_i(B_{31} \oplus K_{31}) \oplus K_{32}\,, \\ C &= PF(B_{31})\,. \end{aligned}$$

2.1 Permutations

Les permutations initiales et finales sont respectivement données par :
Permutation Initiale (PI) :

```
for i in 0 ... 127
    swap( bit(i) , bit((32 * i) % 127) )
```

Permutation Finale (PF) :

```
for i in 0 ... 127
    swap( bit(i) , bit((2 * i) % 127) )
```

Vous pourrez vérifier que $PF = PI^{-1}$ ...

2.2 S-boxes

Dans SERPENT, les S-boxes (rappel : chaque itération utilise une S-box S_i distincte) sont appliquées sur des blocs de 4bits ; une S-box est donc en réalité constitué de 32 sous-S-boxes et sera donc représentée comme un tableau de taille 32×16 indiquant la relation entrée-sortie pour chaque bloc de 4 bits.
Dans l'article scientifique original de SERPENT, soumis dans le concours de standardisation de l'AES

du NIST, les S-Box sont laissées à la discrétion de l'utilisation ; une méthode très générale étant proposée.
Nous proposons ci-dessous une description simplifiée d'une instanciation inspirée de l'article scientifique initial. Il s'agit de la première simplification.
Nous proposons d'utiliser pour la S-box S_0 les 32 tableaux constituant les S-boxes de DES.
Ensuite, il vous est proposé de permuter les S-boxes de façon itérative : la S-box S_i est dérivée de S_{i-1} de la façon suivante :

```
for index_box in 0 ... 31
    for index_bits in 0 ... 15
        i = index_bits + sbox[ index_box , index_bits ]
        j = sbox[ i , index_bits ]
        swap ( sbox[ index_box , index_bits ] , sbox[ index_box , j ] );
```

2.3 Transformation Linéaire *LT*

La transformation linéaire est utilisée sur 4 blocs de 32 bits notés (après ajout de la clé K_i et application de la S-box S_i) X_0, X_1, X_2, X_3

$$
\begin{aligned}
&(X_0, X_1, X_2, X_3) = S_i(B_i \oplus K_i)\ ;\\
&X_0 = X_0 <<< 13\ ,\\
&X_2 = X_2 <<< 3\ ,\\
&X_1 = X_1 \oplus X_0 \oplus X_2\ ,\\
&X_3 = X_3 \oplus X_2 \oplus (X_0 << 3)\ ,\\
&X_1 = X_1 <<< 1\ ,\\
&X_3 = X_3 <<< 7\ ,\\
&X_0 = X_0 \oplus X_1 \oplus X_3\ ,\\
&X_2 = X_2 \oplus X_3 \oplus (X_1 << 7)\ ,\\
&X_0 = X_0 <<< 5\ ,\\
&X_2 = X_2 <<< 22\ ;\\
&B_{i+1} = (X_0, X_1, X_2, X_3)\ .
\end{aligned}
$$

avec $<<< n$ l'opération (binaire) de permutation circulaire de n bits et $<< k$ l'opération (binaire) de décalage de k (avec donc perte des k premiers bits.

2.4 Génération des clés d'itération

C'est le second point de (légère) simplification qui est proposée.
On peut considérer que la clé est toujours de 256 bits partagés en 8 blocs de 32 bits, notés $w_0, \ldots, w_7$.

Chaque clé d'itération K_i est simplement constituée de 4 blocs consécutifs distincts $K_i = (w_{i*4}, w_{i*4+1}, w_{i*4+2}, w_{i*4+3})$. Puisque qu'il faut 33 clés d'itérations K_i, il faut donc 132 blocs w_i.
Les blocs additionnels de 32 nécessaires $w_8, \ldots w_{131}$ sont générées en utilisant les précédents par la relation de récurrence suivante :

$$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \Omega \oplus i) <<< 11$$

où Ω est une constante donnée par $0x9e3779b9$.

3 Communication Asynchrone

Une fois que la phase d'initialisation des communications (authentification à l'aide du ou des certificats, échange d'une clé secrète) Alice et Bob partage une "master key" *MK* qui peut être utilisée pour échanger des messages avec le chiffrement symétrique SERPENT.

La spécificité du *protocole de communication asynchrone de Signal*, que vous êtes invité à implémenter, est que chaque message est chiffré avec une clé distincte. Cette façon de procéder, décrite ci-dessous, est appelée le "*ratchet*" (le "cliquetis") et permet d'assurer la "*forward secrecy*" [1] qui assure la continuité de la confidentialité des communications même si une clé de chiffrement est compromise !

Pour cela, le principe de 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 $0x02$: $ChainKey = HMAC - SHA256(ChainKey, 0x02)$.
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.

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

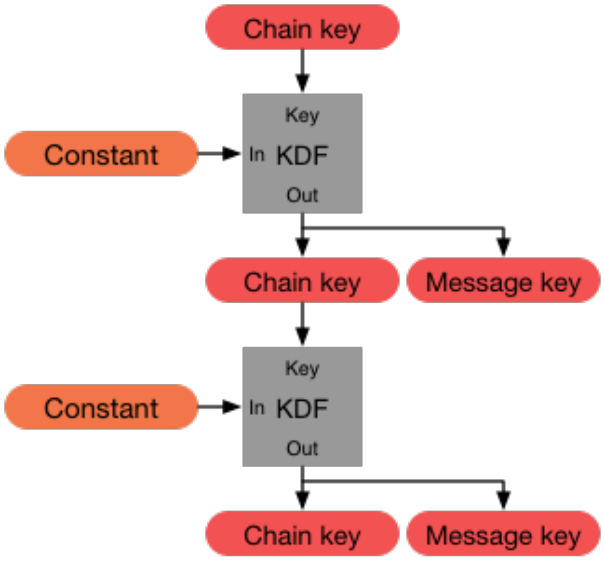

Figure 1 – Illustration graphique de clés chaînées

4 Échange de propriété de document

Depuis près d'une décennie, les NFT (*Non-Fungible Tokens*) permettent d'échanger la propriété que les utilisateurs peuvent avoir sur des données numériques. Contrairement à une monnaie, dans lesquels les fonds sont fongibles, c'est-à-dire identiques et interchangeables, les NFT ne le sont pas : chaque document est unique (un euro vaut un autre euro, au contraire, un NFT ne vaut pas un autre NFT).
Le principe d'enregistrement des transactions de NFT est celui de la *blockchain* ; très simplement, il s'agit d'un ensemble de transactions séquentielles en chaine de sorte que chaque transaction identifie la précédente formant ainsi une "chaîne de bloc" (dans notre NFT, un bloc correspond à une transaction). Une transaction, ou un bloc, est constitué de :

1. de l'identifiant (hash) du bloc précédent ;
2. de transactions (une dans notre cas) ;
3. des données sur la donnée, son auteur et son propriétaire ;
4. un sel, c'est-à-dire une donnée (pseudo-)aléatoire ;
5. le hash de l'ensemble des données du bloc (excepté le hash lui-même, bien sûr . . .)

La particularité d'une *blockchain* est que cette dernière est décentralisée et peut être publique : n'importe qui peut ajouter un bloc de transactions. L'ensemble des utilisateurs doivent donc travailler sur la même chaîne de blocs ; en cas de *fork*, si deux chaînes co-existent, l'une des deux va s'imposer comme étant celle majoritairement utilisée.

Si vous voulez changer une transaction, vous devrez tous les blocs suivants ... Afin de rendre cela impossible, chaque bloc est validé par une "preuve de travail" (PoW pour *Proof of Work*), le sel est choisi de sorte que les N premiers bits du hash soient tous égaux à 0. Si vous prenez par exemple $N = 16$ (en pratique on a plutôt $N = 32$) pour chaque sel choisit (pseudo-)aléatoirement, vous avez une probabilité de $2^{-16} = 1/65536$ que le hash soit valide . . .changer un bloc de transaction nécessite de recalculer tous les sels des blocs suivants ...
Notez qu'en pratique cette procédure est extrêmement couteuse en calcul aussi (1) celui qui valide un bloc est "payé" ce qui donne lieu à des mineurs de bitcoins pour lesquels il s'agit d'une activité rémunératrice et (2) une autre méthode de validation par "preuve d'enjeux" (PoS pour *Proof of Stake*) dans lequel c'est un ensemble d'utilisateurs dont la somme des soldes atteint un certain niveau qui est utilisé ... (dans notre cas, nous ne pourrons pas implémenter cette option, car nous n'aurons pas assez d'utilisateurs . . .)

5 Protocole de vérification de connaissance

Imaginons le cas suivant : vous recevez de l'UTT un certificat signé par une tierce personne (disons le Ministère de l'Enseignement Supérieur et de la Recherche, MESR). Vous exigez que le MESR prouve qu'il possède bien la clé privée avec lequel le certificat a été signé. Évidemment, le MESR accepte seulement s'il peut vous convaincre sans divulguer la moindre information sur sa clé privée.
C'est tout le but des protocoles ZPK (*Zero-Knowledge proof* ou "preuve à divulgation nulle de connaissance"). Les deux implémentations qui vous sont proposées sont valables pour RSA et El-Gamal/ DSA respectivement.

5.1 Protocole de Schnoor (logarithme discret)

Dans le cas où la clé publique est un entier p premier et a un générateur ; la clé publique est constituée de la définition du corps cyclique p et a ainsi que $pub = a^s$ et la clé secrète est s.
Pour s'authentifier auprès de Rémi, le MESR choisit $m \in \mathbb{Z}_p$, calcule $M = a^m$ et l'envoie à Rémi.
Ensuite, Rémi choisit $r \in \mathbb{Z}_p$ et le divulgue au MESR.
Enfin, le MESR calcule $Preuve = m - r \times s$ et le retourne à Rémi.
Si Rémi peut vérifier que $M = a^{Preuve} \times pub^r$ alors la vérification est acceptée.

5.2 Protocole de Guillou-Quisquater (RSA)

Dans le cas où le MESR possède une clé privée RSA (on rappellera que la clé publique est constituée du module $n = p \times q$ et de l'exposant d'encryption e ; la clé privée est l'exposant de déchiffrement $d = e^{-1} \mathrm{mod} \Phi(n)$), Rémi reçoit une signature $S = H^e$ pour lequel le MESR doit prouver qu'il peut calculer $H = S^d$ . . .
Pour cela le MESR choisit $m \in \mathbb{Z}_p$, calcule $M = a^m$ et l'envoie à Rémi.
Rémi choisit $r < e$ et le divulgue au MESR.

Enfin, le MESR calcule $Preuve = mH^{-r}$ et le retourne à Rémi.
Si Rémi peut vérifier que $S^r \times Preuve^e = M$ alors la vérification est acceptée.

---

6 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 ne connaissant pas tous les langages informatiques du monde ; aussi le langage de programmation est imposé : *python*. Par ailleurs, les bibliothèques / packages liées à la cryptographie sont interdites (les modules généraux, *bitarray*, *numpy*, *os*, etc . . .peuvent être utilisés).
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 (H109). Vous devriez pouvoir exécuter votre code *python* sur mon PC. 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.

Votre code doit être a minima capable de prendre en entrée un texte (pour le chiffrement, la signature, le hash, la blockchain, etc. ...) ; vous pouvez aussi vous amuser à assurer la prise en charge d'image pgm comme en TP, de fichiers binaires, etc. .... mais la prise en charge des textes est le minimum souhaité.

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 et, si des choix particuliers ont été faits (par exemple utilisation d'une bibliothèque très spécifique, quelle fonction de signature, quelle fonction de hachage, quelles modifications ont été nécessaires) les justifier brièvement. 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 net dans des livres (ou, 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. . . .

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

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