Un coffre-fort numérique simplifié

GS15 - A24 - Projet Informatique

Sujet présenté en cours le 29/10
Rapport et codes à rendre avant le 05/01
Soutenances entre le 06/01 et le 10/01

1 Description du projet à réaliser

Un coffre-fort numérique est un système sécurisé permettant de stocker et gérer des informations sensibles sous forme numérique. Il fonctionne comme un coffre-fort physique, mais au lieu de protéger des objets matériels, il protège des données, des documents, ou des fichiers numériques. Ce type de solution est souvent utilisé pour des informations confidentielles, telles que des mots de passe, des fichiers juridiques, des informations bancaires ou des contrats sensibles. Les principales caractéristiques d'un coffre-fort numérique sont les suivantes :

— **Sécurisation des accès** : Les utilisateurs accèdent au coffre-fort à l'aide de moyens d'authentification forts, comme des mots de passe robustes, un couple de clés publique/privée, l'authentification à deux facteurs (2FA), etc. . . .

— **Chiffrement des données** : Les données stockées dans un coffre-fort numérique sont généralement chiffrées, ce qui garantit que seules les personnes autorisées peuvent y accéder ou les lire (en plus d'une éventuelle gestion des droits d'accès comme sur tout système de fichiers . . .).

— **Gestion des utilisateurs** : Les utilisateurs peuvent configurer des niveaux d'accès pour différentes parties prenantes. Par exemple, certains utilisateurs peuvent avoir des droits de lecture, tandis que d'autres peuvent être autorisés à modifier ou supprimer des documents.
— **Traçabilité et audit** : Les actions effectuées dans le coffre-fort sont souvent journalisées, permettant un suivi des accès et des modifications pour garantir une transparence et détecter d'éventuelles tentatives d'intrusion. On précisera que (idéalement) le coffre-fort doit être en ligne et que les étudiants peuvent utiliser un stockage quelconque pour cela (git space ?), mais que l'utilisation d'un dur sur le disque local est suffisante.

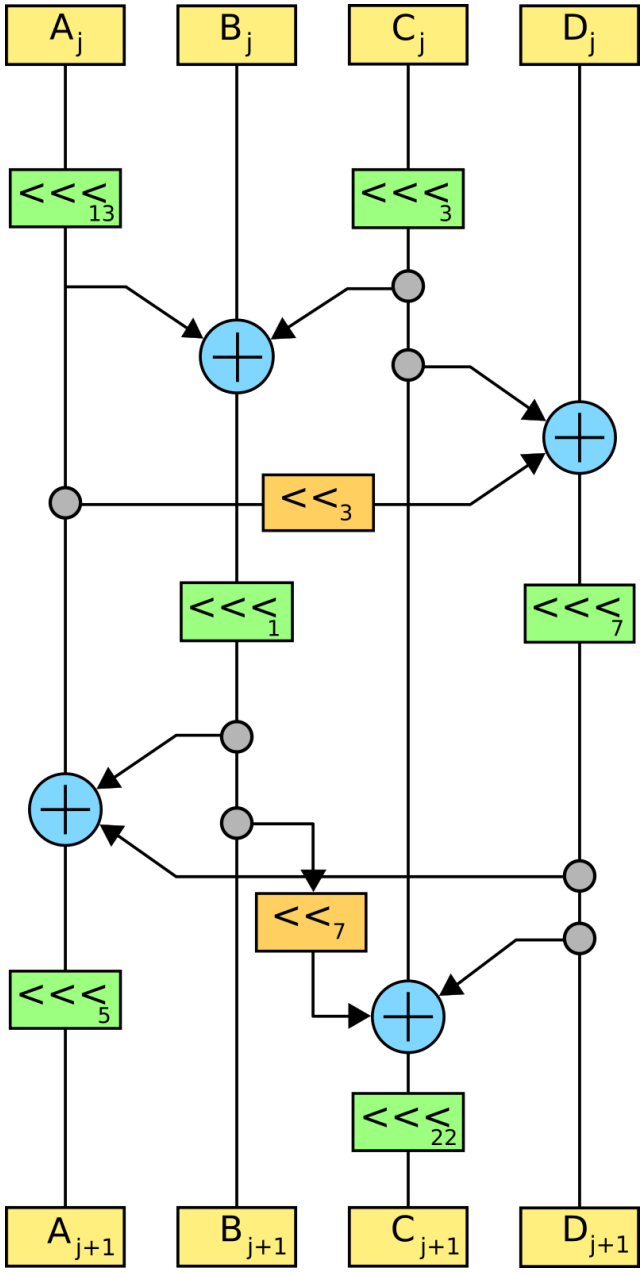
Concrètement, l'utilisation du coffre-fort nécessite quelques étapes préalables :

1. **Enrollement** : Une personne doit se créer un compte. En pratique cela reviendra à créer un répertoire et un couple de clés privée / publique. La clé publique sera le seul élément du coffre-fort qui ne sera pas chiffré.
2. **Dérivation de la clé (KDF)** : L'utilisation ne rentre pas une clé publique directement ... mais un mot de passe. Ce sera à vous, décréter une fonction de dérivation de la clé à partir d'un mot de passe. En pratique on procédera par hashage du mot de passe en utilisant un algorithme de hashage basé sur une fonction éponge avec un grand nombre de phases "d'essorage" afin de pouvoir avoir une clé privée de grande taille.
3. **Authentification à double sens** : Avant chaque utilisation du coffre-fort, l'utilisateur doit demander le certificat auprès du coffre-fort et le vérifier auprès de l'autorité de certification. Une fois cette authentification effectuée, l'utilisation doit s'authentifier auprès du coffre-fort. Pour cela il vous sera demandé d'implémenter une fonction de preuve de connaissance de la clé privée à divulgation nulle (ZPK : Zero-Knowledge-Proof).
4. **Échange de clés** : Une fois cette double authentification effectuée, le client serveur s'échange une clé secrète de session en utilisant le protocole d'échange de clés de Diffie-Hellman.
5. **Dépôt / consultation de fichiers du coffre-fort** : Vous devrez implémenter COBRA, la version modifiée par votre prof sadique du chiffrement symétrique par bloc SERPENT.
   Les échanges entre client et serveur sont chiffrés avec une clé de session et chaque échange est authentifié avec un hashmac.
   Les fichiers stockés sur le serveur sont chiffrés en utilisant la clé privée de l'utilisateur et l'algorithme de chiffrement asymétrique RSA.
6. **Suggestions** : Vous pouvez bien sûr aller plus loin, par exemple en proposant un coffre-fort partagé par 2 utilisateurs (ou plus), faire un système de journalisation, de versionning, des actions / dépôts, proposer un système d'audit (de non-répudiation) pour chacun des fichiers . . .ne vous limitez pas, votre prof est à votre disposition pour discuter d'autres idées, mais avant tout faites ce qui est demandé. Les options ne se substituent pas aux recommandations.

2 Description du chiffrement symétrique COBRA

L'algorithme de chiffrement symétrique (utilisé pour chiffrer les fichiers du coffre-fort) est basé sur Serpent, qui été l'un des finalistes dans le processus de sélection du standard de chiffrement AES. Comme ce dernier, Serpent est basé sur une architecture en réseau de substitution-permutation, il fonctionne sur des blocs de 128 bits (16 octets) et utilise des clés de 128, 192, ou 256 bits, puisque cela été spécifié dans le concours de sélection ....
Serpent est basé sur une permutation initiale, 32 "rondes" (ce grand nombre de "tours" est l'une des raisons pour lesquelles Serpent est souvent considéré comme extrêmement sécurisé, vous pouvez le réduire à 16, voire 12 si votre code est trop lent) et une permutation finale.
chaque itération repose sur 3 principales opérations, mais votre professeur sadique en a rajouté une quatrième :

1. **Add Round Key** : XOR avec la clé d'itération (qui ont donc une longueur de 128 bits ...)
2. **Substitution** : on applique (en parallèle) 32 fois la même *S-Box* de 4 bits.
   Attention 4 S-Boxes différentes sont utilisées (la première pour les itérations de 0 à 7, puis la deuxième pour les itération de 8 à 15, la troisième pour les itérations de 16 à 23 et enfin la dernière pour les itérations de 24 à 31).
   Les S-boxes de Serpent ont été construites en utilisant celles de DES : vous devez procéder de la même façon pour créer vos propres S-boxes.
3. **La-Feistel-de-Réré** : Dans cette troisième étape, nous allons implémenter une fonction de feistel "home-made" (on ne fera que 3 ou 4 rondes par itération) : Le bloc de 128 est découpé en deux blocs de 64 bits, notés L et R et on a $L_1 \leftarrow R$; $R_1 \leftarrow L \oplus F(K_n; R)$ avec la fonction F définie par les 3 opérations suivantes :
   1. Le bloc R est découpé en blocs de 8 bits sur chacun desquels on inverse l'ordre des bits, puis on applique la fonction inverse $f : 0;1^8 \rightarrow 0;1^8$ définie par $f(x) = \left((x+1)^{-1} \mathrm{mod} 257\right) - 1$.
   Le résultat est un bloc noté Z (on pourra tabuler cette fonction ...).
   2. Le bloc Z est ensuite "mélangé" en utilisant une fonction de permutation des bits $P : Z[n] \leftarrow Z[p[n]]$ ou $p[...]$ représente un vecteur de permutation dont la définition est laissée à votre discrétion. Le résultat de ce mélange est noté Y.
   3. Le bloc Y est enfin découpé en blocs de 8 bits (encore ! c'est une obsession !) qui sont tous utilisés comme graine d'un algorithme de génération d'un nombre pseudo-aléatoire cryptographiquement sécurisé afin de générer autant de nombres d'un nombre de 8 bits.
   On xor avec une clé dérivée de la clé d'itération ; là encore, la fonction de dérivation de la clé est laissée à votre discrétion (à partir de la clé d'itération de COBRA, vous pouvez, par exemple, réutiliser la fonction de hashage et votre KDF !)
4. **Transformation Linéaire** : qui effectue des opérations binaires sur 4 "sous-blocs" de 32 bits comme représentés ci-dessous :



Quant à l'algorithme de "*Key Scheduling* ou d'expansion de clé, on reprendra celui de Serpent qui permet de générer une série de clés de tour à partir de la clé secrète initiale. Ce processus prend une clé d'une longueur fixe (128, 192 ou 256 bits) et produit 33 clés de tour de 128 bits chacune. Ces clés seront, naturellement, celles ensuite utilisées pour chaque tour de chiffrement (32 tours au total, avec une clé finale pour un XOR supplémentaire à la fin).
Cet algorithme est détaillé ci-dessous :

1. **Initialisation de la clé** :
   Serpent accepte des clés de taille variable (128, 192 ou 256 bits), mais pour simplifier l'expansion, la clé initiale est toujours étendue à 256 bits, quelle que soit sa taille d'origine :
   Clé de 128 bits : La clé est complétée par des zéros pour obtenir 256 bits.
   Clé de 192 bits : La clé est complétée de manière similaire. Clé de 256 bits : Elle est utilisée

telle quelle.
Cette clé de 256 bits est ensuite divisée en huit blocs de 32 bits chacun : $K_0, K_1, \ldots, K_6, K_7$.

2. **Itérations de l'expansion de clé** :
L'algorithme génère 132 sous-clés de 32 bits (en vue de construire 33 clés de tour de 128 bits) à partir des 8 blocs $K_0, K_1, \ldots, K_6, K_7$.
Cela se fait en utilisant une série d'itérations basées sur une fonction linéaire et non linéaire. Le processus d'expansion est défini par une relation de récurrence. Après initialisation $w_i = K_i$ pour $i < 8$ on applique l'équation récurrente suivante pour générer chaque sous-clé w_i, avec $i = \{8, 9, \ldots, 130, 131\}$:

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

avec :
— $\oplus$ l'opération XOR (ou exclusif).
— $<<<$ l'opération de rotation circulaire à gauche de 11 bits (circular left shift).
— φ dénote le nombre parfait ($\varphi \approx 1.61803398874989$) ; elle est utilisée pour introduire une entropie supplémentaire dans la génération des sous-clés. En pratique, elle est souvent représentée sous une forme simplifiée comme une constante binaire prédéfinie. i : Représente l'index de la sous-clé courante dans la séquence d'expansion.

3. **Transformation non linéaire (S-boxes)** :
Les sous-clés générées à l'étape précédente sont ensuite transformées à l'aide des S-boxes pour introduire une non-linéarité dans la génération des clés de tour.
Chaque groupe de 4 sous-blocs w_i est combiné pour former une clé de tour de 128 bits, et une transformation via les S-boxes est appliquée pour assurer que chaque clé de tour a une structure suffisamment non linéaire. Les S-boxes utilisées pour les sous-clés sont les mêmes que celles utilisées dans le processus de chiffrement.
Les 132 sous-clés de 32 bits générées sont ensuite organisées en 33 clés de tour de 128 bits chacune par concaténation de 4 blocs de 32 bits :

$$W_i\,(w_{4i}, w_{4i+1}, w_{4i+2}, w_{4i+3})\,.$$

Cela donne les 33 clés de tour $W_0, W_1, \ldots, W_{31}, W_{32}$ où W_i sera la "sous-clé" de 128 bits de la i-ième ronde.

3 Description de la preuve de connaissance à divulgation nulle (ZKP)

Le principe du protocole de ZPK (Zero-Knowledge-Proof) est simple : il est demandé à celui qui diffuse une clé publique de prouver qu'il est en possession de la clé privée correspondante, mais sans divulguer aucune information sur cette clé privée ("à divulgation nulle"). Pour cela vous devrez implémenter la méthode de vérification de Schnorr ou celui de Guillou-Quisquater décrit ci-dessous. Par souci de clarté, dans la description ci-dessous, on suppose que Rémi est très suspicieux lorsqu'il échange avec Nicolas. Rémi sera donc le vérificateur et Nicolas celui qui doit prouver sa connaissance de la clé privée.

3.1 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 brave Nicolas choisit $m \in \mathbb{Z}_p$, il calcule $M = a^m$ et l'envoie à Rémi.
Ensuite, Rémi qui challenge le petit Nicolas : Rémi choisit aléatoirement $r \in \mathbb{Z}_p$ et le divulgue à Nicolas.
Enfin, Nicolas doit réussir le "challenge" posé par Rémi en calculant la preuve $Preuve = m - r \times s$ et la retourne à Rémi.
Si Rémi peut vérifier que $M = a^{Preuve} \times pub^r$ alors la vérification est acceptée.

3.2 Protocole de Guillou-Quisquater (RSA)

Dans le cas où ce bon Nicolas 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 un certificat publique de Nicolas $Cert$ pour lequel notre Nicolas doit prouver qu'il en possession du certificat prévée, le secret de Nicolas $S_N = Cert^{-d}$ ...
Pour cela Nicolas choisit $m \in \mathbb{Z}_p$, calcule $M = m^e$ et l'envoie à Rémi.
Rémi choisit ensuite aléatoirement un challenge $r < e$ et le divulgue à Nicolas.
Enfin, Nicolas calcule la preuve de connaissance $Preuve = mS_N^r$ et le retourne à Rémi.
Si Rémi peut vérifier que $Cert^r \times Preuve^e = M$ alors la vérification est acceptée et Nicolas est accepté dans la cour des grands.

4 Description des autres fonctions "avancées"

— Est-ce que l'on garde l'idée du Rachet pour la communication entre le serveur et le client ?
— Est-ce que l'on ajoute un coffre-fort en mode "block chain" le client garde les preuve de calcul (proof-of-work) et doit être capable d'en transmettre une demandée aléatoirement par le serveur ?
— Toute autre idée de ta part sera la bienvenue !

4.1 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.

Description du menu suggérée pour démonter les fonctions de façon "unitaire"

```
Bonjour ô maître T ! 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<- blablabla
->4<- and the list goes on \ldots
```

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 5 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 5 ; les soutenances auront lieu la semaine précédant les examens finaux ; **vous devez réserver un créneau de soutenance.**

Enfin, je vous informe que la notation est faite afin que :
— Un programme qui fonctionne et respecte l'ensemble des exigences se voit attribuer un minimum de 15/20 ;
— toutes les initiatives personnelles seront appréciées et valorisées (mais il est plus important de respecter les consignes) ;

— le respect, en sus, de l'ensemble des "recommandations" et l'ajout de fonctionnalités supplémentaires est le seul moyen pour espérer avoir un 20/20 ;
— 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 appliquée !

---

5 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 (X102). 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 !