Protocole de “Communication Sécurisé”

GS15 - A18 - Projet Informatique

Sujet présenté en cours le 16/10
Rapport et codes à rendre avant le 06/01
Soutenance entre le 07/01 et le 11/01

---

1 Description du projet à réaliser

Le but de ce projet informatique est de vous faire créer un outil offrant la possibilité de créer l'intégralité d'un protocole de communication sécurisé. Éventuellement, selon les développement, vous pourrez utiliser les clés générées lors de précédentes exécutions et “passer certaines étapes”.
Les principales étapes de la mise en oeuvre d'un protocole de communication sécurisé sont les suivantes:;

- Génération d'un couple de clés privée / publique
- Authentification
- Partage d'une clé de symmétrique
- Chiffrement d'un message avec la clé privée + signature.

Le choix des étapes à utiliser est laissé à l'utilisation. Par exemple, lors de l'exécution du programme vous pouvez afficher un menu de choix du type :

```
Quelles opérations souhaitez vous faire ?
->1<- Générer une clé publique / privée.
->2<- Authentifier une clé publique / un certificat.
->3<- Partager une clé secrète.
->4<- Utiliser une clé secrète pour chiffrer un message (et le signer).
->5<- Déchiffrer un message et vérifier la signature.
->6<- La totale : THE FULL MONTY.
```

L'utilisateur entre son choix (1, 2, 3, 4, 5 ou 6) ... et le programme doit ensuite le guider, lui demander de choisir une clé, un fichier, etc. . . .

Les quatre algorithmes qui vous sont demandés sont décrits ci-dessous, respectivement dans les sections 2 pour le chiffrement symétrique, 3 pour le hashage, 4 pour le partage de clé et 5 pour la vérification de la clé publique (certificat).
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 arithmétique (tests de Rabin Miller).

Enfin, le choix du langage de programmation vous appartient, néanmoins votre enseignant n'étant pas omniscient, un soutient n'est assuré que pour les langages Matlab/GMPint et C/GMP. La seule contrainte **obligatoire** est seulement de respecter les consignes données dans la section 6 du présent document.

Il est obligatoire cette année un rapport répondant uniquement aux exigences de la section 6 !!).
Date limite de restitution : **Dimanche 6 janvier à 23h59** (au-delà, un point sera enlevé par minute de retard).
Une soutenance est prévue la semaine précédant les examens finaux, vous devrez vous inscrire pour "réserver" un horaire pour votre présentation.

---

2 Chiffrement Symétrique IDEA

Dans cette partie, le but du projet est de proposer une méthode de chiffrement appelé Threefish. Cette méthode accepte en entrée des blocs de 64 bits. La clé (originale) est de 128 bits, mais vous pouvez adapter le chiffrement en utilisant des clés de 96, 160 ou 256 bits.

La clé originale K est découpée en $N = \{6, 8, 10, 16\}$ sous-clés de 16 bits (en fonction de la taille de la clé originale). Le fonctionnement de l'algorithme nécessite 52 sous-clés de 16 bits qui sont simplement obtenues par permutation circulaire (vers la gauche) de 25 bits de la clé "principale". Chaque permutation de la clé originale permet d'obtenir N nouvelles sous-clés et cette opération est appliquée autant de fois que nécessaire pour obtenir les 52 sous clés nécessaires.

Itération :

Le chiffrement IDEA nécessite 8 itérations et une dernière "demi" itération. La description de chaque itération est donnée ci-dessous (voir Figure 1). on notera que les trois seules opération utilisées sont l'addition (modulo 2^{16}, notée $\oplus$ dans la figure) le XOR binaire (noté $\boxplus$ dans la figure) et enfin la multiplication (**modulo** $2^{16}+1$ où la valeur 2^{16} est notée $000\ldots000$, cette opération est notée $\odot$ dans la figure). Enfin dans cette description on notera $\leftarrow$ l'opération d'affectation, B_1, B_2, B_3, B_4 les blocs du message et $K_1, K_2, K_3, K_4, K_5, K_6$ les sous-clés utilisée pour l'itération:

1. $B_1 \leftarrow B_1 \odot K_1$; Le bloc B_1 est multiplié par K_1 ;
2. $B_2 \leftarrow B_2 \boxplus K_2$; Le bloc B_2 est sommé avec K_2 ;
3. $B_3 \leftarrow B_3 \boxplus K_3$; Le bloc B_3 est sommé avec K_3 ;
4. $B_4 \leftarrow B_4 \odot K_4$; Le bloc B_4 est multiplié par K_4 ;
5. $T_1 \leftarrow B_1 \oplus B_3$; Le bloc T_1 est le résultat du XOR entre B_1 et B_3 ;
6. $T_2 \leftarrow B_2 \oplus B_4$; Le bloc T_2 est le résultat du XOR entre B_2 et B_4 ;
7. $T_1 \leftarrow T_1 \odot K_5$; Le bloc T_1 est multiplié par K_5 ;
8. $T_2 \leftarrow T_2 \boxplus T_1$; Le bloc T_2 est sommé avec T_1 ;
9. $T_2 \leftarrow T_2 \odot K_6$; Le bloc T_2 est multiplié par K_6 ;
10. $T_1 \leftarrow T_1 \boxplus T_2$; Le bloc T_1 est sommé avec T_2 ;
11. $B_1 \leftarrow B_1 \oplus T_2$; Le bloc B_1 est "XORé" avec T_2 ;

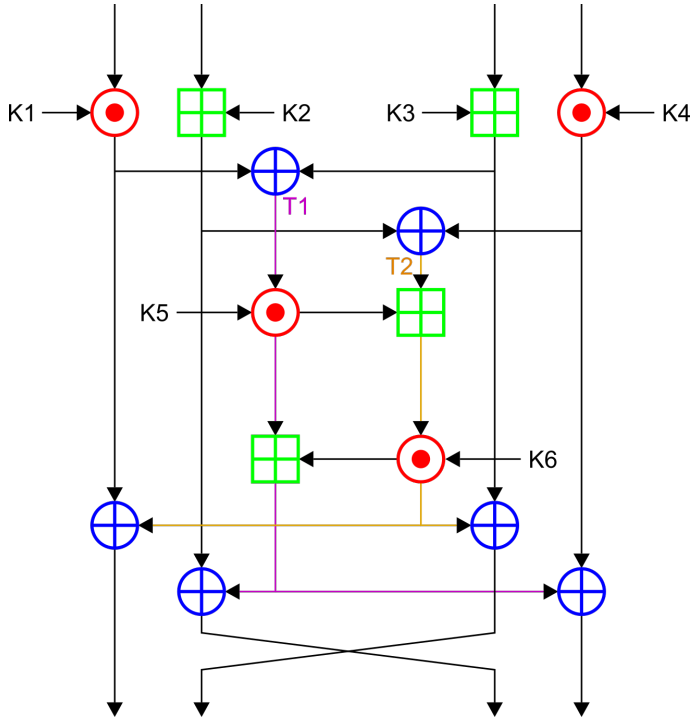

Figure 1: Illustration du fonctionnement de la méthode de chiffrement IDEA

12. $B_3 \leftarrow B_3 \oplus T_2$; Le bloc B_3 est "XORé" avec T_2 ;
13. $B_2 \leftarrow B_2 \oplus T_1$; Le bloc B_2 est "XORé" avec T_1 ;
14. $B_4 \leftarrow B_4 \oplus T_1$; Le bloc B_4 est "XORé" avec T_1 ;
15. $B_2 \leftrightarrow B_3$; Les blocs B_2 et B_3 sont permutés ;

Les données T_1 et T_2 sont des variables intermédiaires représentées en violet et orange respectivement sur la figure 1.

Enfin la dernière demi-itération est uniquement des 4 premières étapes.

Le déchiffrement de chaque bloc de 64 bits se fait en inversant chacune des opérations précédentes. On commencera donc par inverser l'itération "partielle", puis on appliquera les opérations inverses de chacunes des itérations (en partant "de la fin").

Rappelons que l'inversion d'une opération XOR $\oplus$ est efféctué simplement en applique le même XOR : si par exemple $B_4 \leftarrow B_4 \oplus T_1$, alors $B_4 \leftarrow B_4 \oplus T_1 = B_4 \oplus T_1 \oplus T_1 = B_4$.

L'inversion de l'opération d'addition, par T_1, se fait simplement en additionnant le résultat avec $2^{16} - T_1$ (c'est également le complément à deux $\bar{T_1}$ + 1 avec $\bar{T_1}$ le NOT binaire, bit par bit) : $T_1 \leftarrow T_1 \boxplus T_2$ est inversé par $T_1 \leftarrow T_1 \boxplus (2^{16} - T_2) = T_1 \boxplus T_2 \boxplus (2^{16} - T_2) = T_1 \boxplus 2^{16} = T_1$.

Enfin, le plus délicat est la multiplication. En effet, rappelons que les **multiplications se font modulo $2^{16} + 1$ et que ce nombre est premier !** Il est donc possible d'inverser une multiplication en effectuant la multiplication par l'élément inverse : par exemple $B_1 \leftarrow B_1 \odot K_1$ s'inverse par $B_1 \leftarrow B_1 \odot K_1^{-1} = B_1 \odot K_1 \odot K_1^{-1} = B_1$.

2.1 Recommandations / Exigences

Autant que faire se peut, il vous est demandé d'implémenter :

1. le chiffrement en mode ECB, CBC et PCBC ;
2. de proposer à l'utilisateur de choisir la taille de la clé (en dur ou à choisir en ligne).
3. le chiffrement d'un fichier ainsi que son déchiffrement (le chiffré lui aussi étant écrit dans un fichier);
4. le fichier doit être d'une taille adéquate, au moins quelques kilo-octets;

---

3 Hashage: SHA3

L'algorithme qui doit être implémenté pour le hashage (et donc la signature) est SHA3. Ce dernier fonctionne sur le principe des fonctions éponges dont le principe est illustré dans la figure 2 et décrit ci-dessous :
Un bloc est consistué de $b = r + c$ bits (c signifie la "*capacité*" et r le "*rate*" / taux) ;
Une fonction $f0, 1^b \rightarrow 0, 1^b$ transforme un bloc en un autre (cette fonction est assimilable à une permutation pseudo aléatoire de $0, 1^b$ vers $0, 1^b$).

Les opérations utilisées pour efféctué un hashage avec une fonction éponge sont les suivantes :
À l'initialisation, le message qui doit être "hashé" est sujet à un padding pour avoir une taille multiple de r , notés $P = (P_1, \ldots, P_n)$
À l'itération i, le bloc B_{i-1} de b bits est "découpé" en deux partie de r et c bits respectivement. Le bloc de r bits est "XORé" avec le i-ième bloc de r bits du message $r \oplus P_i$. Les résultats du nouveau bloc (avec r modifié) est donné la fonction f pour obtenir le bloc B_i.

Après n itérations, l'ensemble des blocs du message $P_1, \ldots, P_n$ ont été absorbés par la fonction "éponge" et commence alors la phase de "récupération" du hash. Pour obtenir un hash d'une taille $p = m \times r$ bits, la fonction f est appliquée $m - 1$ fois ; lors de chacune de ces itéarations, les r bits correspondant à la sous-partie du bloc précédemment sont expulsés. La concaténation de ces morceaux de hash permet d'obtenir le Hash $H(M) = (r_n | r_{n+1} | \ldots | r_{n+m-1})$ avec | l'opération de concaténation.

La figure 2 illustre le principe de fonctionnement des fonctions "éponges" pour le hashage avec les deux phase "d'absorption" et de "restitution".

Concrètement, il vous ait demandé d'implémenté SHA3 (légèrement simplifié), dont les paramètres sont les suivants :

1. de créer votre propre fonction de padding du message M, pour obtenir un message dont la taille est un multiple de r ;
2. l'initialisation se fait avec un premier bloc B_0 constitué uniquement de 0 ;
3. le nombre d'itération est fixé à $n = 24$;
4. les blocs sont d'une taille 1600 bits, constitué de 5×5 sous-blocs de 64 bits ;

5. les paramètres c et r sont définis en fonction de la taille p du hash souhaité (256, 384 ou 512 bits) par $c = 2p$.
6. les sous bloc r_i et c_i sont respectivement constitués des premiers et des derniers bits du bloc B_i.

La fonction f opère sur des tableaux / matrice de 5×5 sous-blocs de 64 bits; les 1600 bits sont donc représentés par $B[i, j, k]$ avec $0 \leq i \leq 24$, $0 \leq j \leq 24$ et $0 \leq k \leq 63$ (on note dans tout ce qui suit les indices entre 0 et 24 pour i et j et $B[i, j, :]$ représente tout les éléments pour des indices i et j donnés) . À chaque itération, la fonction f applique les 5 opérations suivantes :

1. on remplace chaque bit de chaque sous-blocs de 64 bits par un XOR avec le bit de parité d'une colone adjacente : $B[:, j, k] \leftarrow B[:, j, k] \oplus parite(B[:, j, k - 1])$;
2. on permute les blocs de 64 bits de t bits avec t qui dépend de la position dans le tableaux (vous pourrez définir t en fonction des indices (i, j) comme vous le souahitez ; $B[i, j, :] \leftarrow B[i, j, :] << t(i, j)$;
3. on permute les sous-blocs de 64 bits du tableau : $B[i, j, :] \leftarrow B[j, 2i+3j, :]$ (attention à la permutation des lignes et des colones) avec ici, bien sûr, $2i + 3j \text{mod} 5$;
4. on effectue un XOR entre les lignes : $B[:, j, :] \leftarrow B[:, j, :] \oplus (B[:, j + 1, :] \& B[:, j - 1, :])$;
5. on effectue une dernière opération de XOR entre certains bits des mots de 64 bits : $B[j, j, :] \leftarrow B[j, j, 2^m - 1] \oplus B[j, j, m + 7 \times L(m)]$ avec $m = \{0, \ldots, 6\}$, $2^0 - 1 = -1 \equiv 63 \text{mod} 64$ et $L(m)$ correspondant à la sortie (un bit) d'un LFSR de 8 bits.

Il vous ait simplement demandé d'implémenter SHA-3 avec la possibilité pour l'utilisation de choisir la longueur de son hash (256, 384 ou 512 bits).
En outre, dans le protocole de communication sécurisé, l'utilisation peut demander de signer chacun des ces message chiffrés. Dans ce cas le message clair (avant son chiffrement) est hashé (avec SHA3) puis signé avec une méthode de signature de votre choix mais utilisant la clé publique de Alice (A, α, p) utilisée pour l'échange de clé Diffie-Hellman (voir Section 4 ci-dessous).

---

4 Partage de clé : Diffie Hellman

Nous reviendrons très brièvement sur Diffie-Hellman (DH), cette méthode ayant été présenté en cours de façon détaille. L'échange de clé (secrète / privée) avec ce protocole s'effectue de la façon suivante :

1. Alice choisit un entier premier (grand), noté p ;
2. Alice cherche un générateur α du corps $\mathbb{Z}_p$;
3. Alice choisit un entier $a \in \mathbb{Z}_p$ et calcule $A = \alpha^a$; ce résultat A est envoyé à Bob ;
4. Bob choisit un entier $b \in \mathbb{Z}_p$ et calcule $B = \alpha^B$; ce résultat B est envoyé à Alice ;
5. Alice et Bob calculent respectivement B^a et A^b, les deux résultats étant égaux : $B^a = A^b = \alpha a \times b$! Ceci constitue la clé d'Alice et Bob

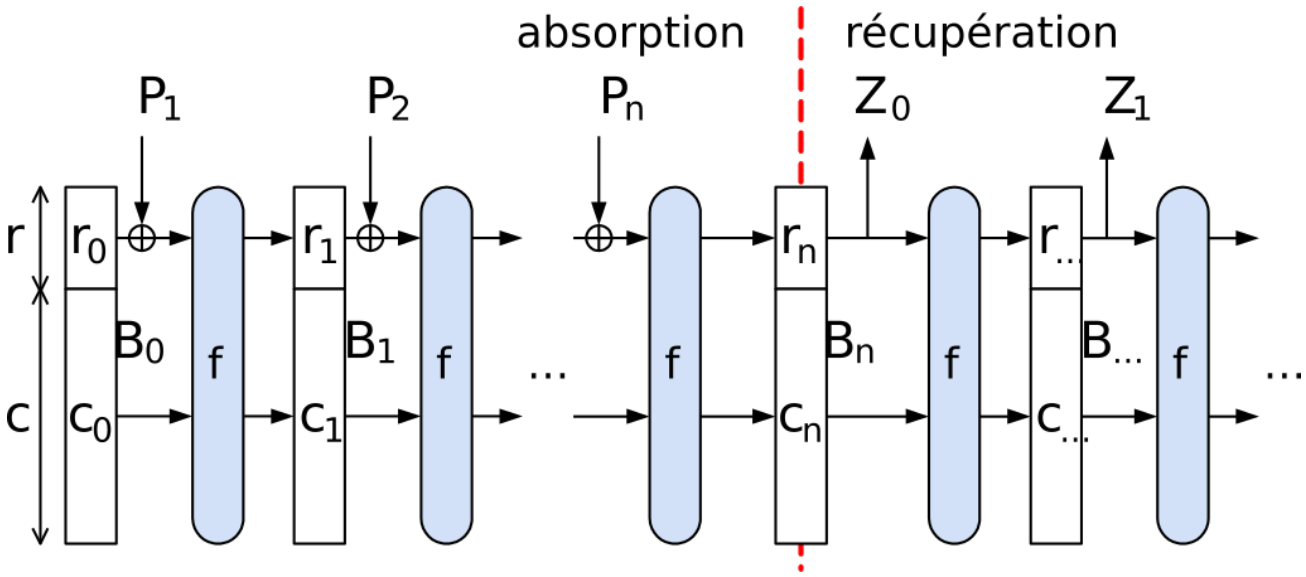

Figure 2: Illustration du principe des fonctions “éponges” pour le hashage.

4.1 Recommandations / Exigences

Autant que faire se peut, il vous est demandé d’implémenter :

1. le test de Rabin Miller pour la recherche de grand nombre premiers ;
2. d’utiliser des nombres premiers d’une taille “réaliste”, i.e 1024 bits.
3. d’utiliser la clé secrète partagée pour générée une clé de chiffrement pour le cryptosystème symétrique IDEA (donc d’une taille inférieur), par exemple en utilisant le hash SHA3 . . .

---

5 Vérifier une clé publique : Certifacat (X509 simplifié)

Dans cette partie il vous est demandé d’implémenter un système simple de signature (proche dans la conception du standard X509):
Bien que cela ne soit pas vu en cours, le principe est extrêment simple. La clé publique de Alice (utilisé pour l’échange de clé avec Diffie-Hellman, voir Section 4) est le triplet (A, α, p). Vous devrez créer une fonction de signature de votre choix utilisant cette clé. Vous devrez en outre générer une seconde clé, celle du “tier de confiance”, l’entité qui va signer le certificat pour Alice (disons l’UTT).
La clé publique de ce tier de confiance sera noté (U, β, q) et sa clé privée u. Le principe est pour Alice de démander à ce “tier de confiance” de signer sa clé publique, elle va donc soumettre sa clé (A, α, p) à ce tier, l’UTT, va lui fournir une signature de sa clé, notée S_u, signature obtenue avec la clé privée de l’UTT u. Alice pour s’authentifier va envoyer l’ensemble (A, α, p, S_u) ; cet ensemble forme un **certificat**.
Bob peut vérifier le **certificat** de Alice en demandant au “tier de confiance” sa clé publique et vérifier la signature S_u comprise dans le certificat d’Alice.

5.1 Recommandations / Exigences

Autant que faire se peut, il vous est demandé d'implémenter :

1. Si vous êtes libre d'utiliser la fonction de signature de votre choix, l'implémentation d'une fonction de signature "sérieuse" est largement préconisée (DSA, El-Gamal ou RSA).

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, l'aide pour la programmation ne sera assuré que pour Matlab/GMPint et C/GMP (et un peu python, mais pas trop quand même).
Par ailleurs, votre code devra être commenté (succinctement, de façon à comprendre les étapes de calculs, pas plus).
De même les soutenance se font dans mon bureau. Si vous avez codés en Matlab/GMPint , C/GMP ou python2/3 vous pourrez normalement faire la démo sur mon PC. Dans tous les cas (notamment si le language est différent) amener 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 (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 excessuf, il est simplement attendu que vous indiquiez les difficultés rencontrées, les solutions mise en oeuvre et, si des choix particulier ont été faits (par exemple quelle fonction de signature) les justifier brièvement. Faites un rapport très court (1 à 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 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 !