Une “Block-chain” simplifiée

GS15 - A20 - Projet Informatique

Sujet présenté en cours le 13/10
Rapport et codes à rendre avant le 09/01
Soutenances entre le 11/01 et le 15/01

---

1 Description du projet à réaliser

Le but de ce projet informatique est de vous faire créer une technologique similaire à une «block-chain simplifiée». L'idée étant de vous faire développer un outil permettant de faire des transactions de façon authentifiée, irrévocable. En outre, un outil de communication chiffrée (qui permettra aux utilisateurs de se mettre d'accord sur les transactions) sera également développé.

Notons en préambule que l'algorithme le plus détaillé est celui qui n'est pas vu en cours, le chiffrement symétrique *KASUMI* modifié par les soins de votre professeur pervers et sadique.

Les autres algorithmes sont décrits de façon très succincte pour deux raisons : (1) car ils ont été (ou seront) étudiés en cours et (2) car **dans ce projet, vous avez beaucoup de liberté quant à l'implémentation pratique des algorithmes.** En effet vous constaterez qu'il vous ait demandé de suivre certaines “règles” à partir desquelles vous pouvez faire le minimum ou en faire beaucoup plus ; des suggestions vous seront données dans les différents points.

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 un message.
->2<- Déchiffrer un message.
->3<- Générer des couples de clés publiques / privées.
->4<- Générer un hash / une empreinte.
->5<- Vérifier un hash / une empreinte.
->6<- Effectuer une preuve de travail.
->7<- Vérifier une transaction (une signature).
->8<- Débuter / incrémenter la Block-chain.
->9<- Vérifier l'intégrité de la block-chain.
->10<- I WANT IT ALL !! I WANT IT NOW !!
```

Les algorithmes que vous devez développer, pour chacun des choix possibles, sont décrits ci-dessous (en commençant pas ceux que vous pouvez le plus implémenter le plus tôt dans le semestre) respectivement dans les sections 2 pour le chiffrement symétrique (Choix `->1<-` et `->2<-`), 3 pour la création d'un

couple de clés publique / privée (Choix `->4<-`), 4 pour le hashage (Choix `->4<- et ->5<-`) et la preuve de travail (Choix `->6<-`), 5 pour la génération et la vérification d'une signature (Choix `->7<-`).

Enfin le principe de fonctionnement d'une block-chain est présenté dans la section 6. Cela vous sera utile autant pour votre "culture" personnelle que pour l'implémentation des opérations de création, incrémentation et vérification d'une block-chain (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. ...).

1.1 Exigences

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 (version 2 ou 3)** ;
2. **respecter les consignes** données dans la section 7 du présent document ;
3. **chaque section contient un paragraphe "exigences" que vous devez suivre** (par exemple utiliser des clés publiques/privées de 512 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 7 ; 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 16/20 ;
- 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 *KASUMI*

Concernant le chiffrement symétrique / à clé privée , il vous ait demandé d'implémenter le chiffrement par bloc *KASUMI*. Cet algorithme de chiffrement repose sur une construction de Feistel à 8 itérations, utilisant

des blocs de 64 bits et des clés de 128 bits. Cet algorithme à été proposé par le est celui au cœur de la confidentialité des échanges dans les protocoles de téléphonie 3G et suivants.

Le fonctionnement global de cet algorithme est brièvement décrit dans le présent document, ci-dessous, illustré dans la figure 1 ; pour plus de détails, vous pouvez consulter la page Wikipédia (en anglais) ou mieux, aller directement consulter la norme telle que fournie par le 3GPP (l'organisme de standardisation pour la communication mobile).

Attention, deux modifications, par rapport à l'algorithme original, sont imposées !

Comme dans tout schéma de chiffrement sur la division du bloc à chiffrer en deux moitiés (notées L_i pour le Left et R_i pour Right) ainsi que sur la relation suivante itérativement appliquée :

$$\begin{cases} R_i &= L_{i-1}\,; \\ L_i &= F(L_{i-1}) \oplus R_{i-1}\,. \end{cases} \tag{1}$$

Naturellement, avant la première itération, le bloc clair (de 64 bits) à chiffrer est divisé en deux parties, respectivement, L_0 et R_0 (de 32 bits) ; les deux derniers blocs L_8 et R_8 sont concaténés pour donner le chiffré.

Comme pour tout schéma de Feistel, ***KASUMI*** est donc principalement défini par la fonction $F(\cdot)$.
L'une des spécifiés de ***KASUMI*** vient du fait que $F(\cdot)$ est légèrement différente suivant que l'indice de l'itération soit pair ou impair. Plus précisément, la fonction $F(\cdot)$ est la composition de deux fonctions dont l'ordre d'application sera inversé. On aura alors :

$$\begin{cases} F_{\text{impair}}(L_{i-1})) &= FO(KO_i, KI_i, FL(KL_i, L_{i-1}))\,; \\ F_{\text{pair}}(L_{i-1})) &= FL(KL_i, FO(KO_i, KI_i, L_{i-1}))\,. \end{cases} \tag{2}$$

avec KL_i, KI_i, KO_i les "sous-clés" de l'itération i.

Cette construction ainsi que les fonctions FL, FO (et FI) sont illustrées dans la figure 1.

La fonction $FI(KL_i, x)$ qui a pour entrée et pour sortie des blocs de 32 bits opère de façon similaire. Le bloc $x = l|r$ est divisé en deux sous-parties on applique alors :
$r' = I\Big(r \oplus [(l \text{AND} KL_{i,1}) <<< 1]\Big)$
puis :
$l' = I\Big(l \oplus [(r' \text{OR} KL_{i,2}) <<< 1] I\Big)$.
avec $<<< 1$ l'opération de décalage circulaire (a.k.a permutation circulaire) de 1 bit vers la gauche.

Modification #1 : dans la version de GS15 il vous est demandé d'appliquer à chacun de ces résultats de 16 bits une opération d'inversion, notée $I\Big(\ \cdot\ \Big)$ dans un corps de Galois (que vous définirez vous-même en utilisant un polynôme irréductible de 16 de votre choix).

La fonction FO est plus complexe, elle repose sur une construction de Feistel (utilisant 3 itérations, des blocs de 32 bits décomposés en deux sous-blocs de 16 bits.
Chaque itération est définie par (attention c'est bien le bloc de droite r_i qui est conservé !) :

$$\begin{cases} l_j &= r_{j-1}\,; \\ r_j &= r_{j-1} \oplus \big(FI(l_j \oplus KO_{i,j}, KI_{i,i})\big)\,. \end{cases} \tag{3}$$

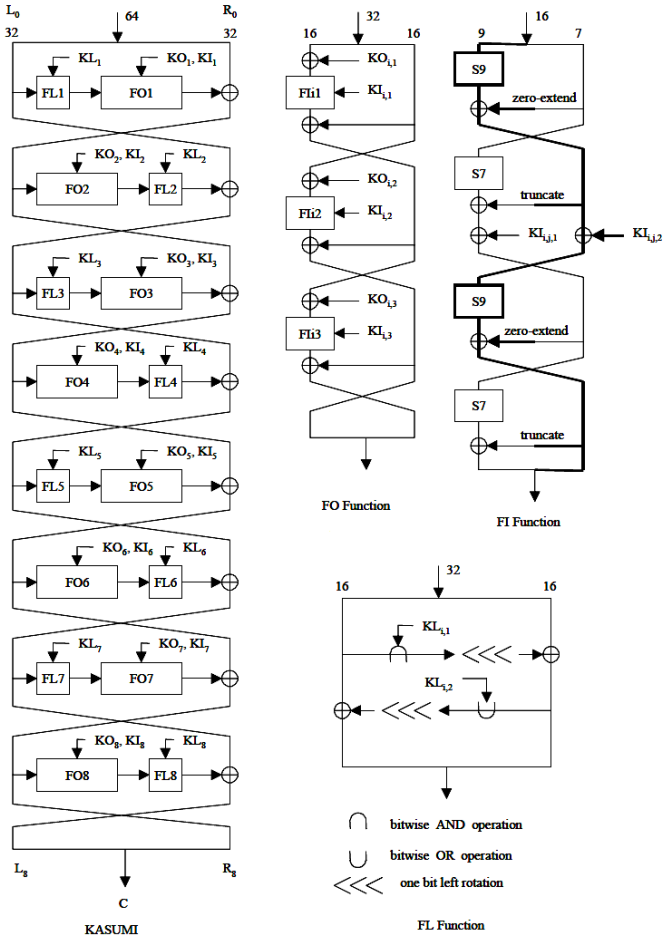

Figure 1: Illustration du fonctionnement de l'algorithme de chiffrement symétrique par bloc *KASUMI*.

avec $KO_{i,j}$ et $KI_{i,i}$ les sous-clés de l'itération (i, j) représentant les indices des deux schéma de Feistel imbriqués.

Modification #2 : la fonction $FI(y, z)$ qui a pour entrée deux blocs de 16 bits fonctionnera, dans ce projet, de la façon suivante : $y >> 2 \oplus \Big(S_{\text{box1}}(z_1)|S_{\text{box1}}(z_2)\Big)$ avec le bloc $z = z_1|z_2$ divisé en deux parties de 8 bits sur lesquelles est utilisée une Sbox distincte.
Vous devrez construire ces deux Sbox en utilisant l'algorithme d'initialisation de RC4 vu en cours.

Le "key scheduling" (génération de sous-clés d'itération à partir de la clé initiale) est laissé à votre discrétion.

2.1 Exigences

Il vous est demandé d'implémenter :

1. le chiffrement en mode ECB, CBC et PCBC ;
2. le chiffrement d'un fichier ainsi que son déchiffrement (le chiffré lui aussi étant écrit dans un fichier);

3. le fichier doit être d'une taille adéquate, au moins quelques kilo-octets;

2.2 Recommandations

Pour aller plus loin, vous pouvez, si vous voulez, regarder les éléments suivants :

1. Proposer une utilisation de l'algorithme de chiffrement *KASUMI* en mode "flux";
2. comprendre et implémenter les modes de chiffrements Counter et GCM (Galois' Counter Mode);
3. vous pourrez paramétrer la création des Sbox en fonction de la clé de sorte que les Sbox ne soient jamais les mêmes;

---

3 Génération d'un couple de clés publique/privée

Nous le verrons en cours durant la seconde moitié du semestre, la génération d'un couple de clés publique/privée nécessite de faire des calculs avec des nombreux grands et notamment de rechercher un nombre premier large (typiquement à 100 chiffres, voire souvent beaucoup plus).

Dans cette partie de cela il est imposé de chercher un entier premier p de au moins 512 bits.
Attention, la signature que nous utilisons étant basé sur la signature de El-Gamal, une fois un grand entier premier p généré, vous devrez trouver un élément α générateur de $\mathbb{Z}_p^\star$.
Pour cela il est clairement illusoire de tester, pour un α donné, toutes ces puissances successives $\alpha^1, \alpha^2, \ldots, \alpha^{p-1}$; il vous faudra donc trouver une astuce ... (indice, reposant sur le théorème de Lagrange).

Cet aspect peut être envisagé plus tardivement dans votre projet, dans un premier temps vous pouvez vous concentrer sur la recherche d'un grand entier premier et l'exponentiation rapide.
Par la suite vous devrez trouver (rapidement) un élément α générateur de $\mathbb{Z}_p^\star$ et générer un couple de clés publique/privée adéquat pour la signature de Diffie-Hellman.

3.1 Exigences

Pour cette partie vous devrez, vous pouvez :

1. Utiliser des fichiers pour gérer, stocker, lire les clés publiques (et privées) ; lors de la soutenance par exemple il pourra être long de générer 3 clés de 1536 bits .....

3.2 Recommandations

Si vous le souhaitez, vous pouvez :

1. Implémenter le partage de clé privée en utilisant le protocole de Diffie-Hellman;
2. Implémenter votre propre générateur de nombre aléatoire (par exemple votre XORshift);

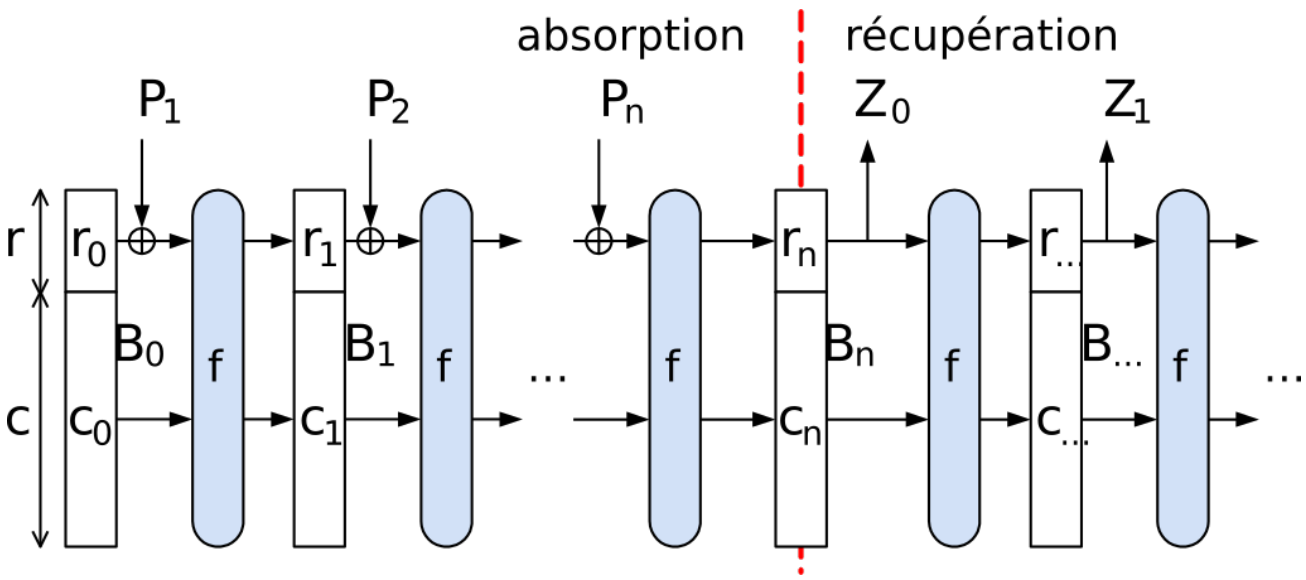

Figure 2: Rappel du principe des fonctions “éponges” pour le hashage.

4 Hashage: Fonction éponge

En ce qui concerne la fonction de hashage, vous pouvez implémenter n’importe quelle fonction de hashage (e.g, MD5, Whirlpool, SHA-256, etc. ...), mais il est impératif de modifier cette fonction afin de l’implémenter dans le cadre d’une “fonction éponge” telle qu’utilisée dans SHA-3 et illustrée dans la Figure 2 ci-dessous.
Si vous souhaitez reprendre une fonction de hashage “standard”, les modifications à faire pour que cette dernière soit utilisable dans le cadre des fonctions éponges sont laissées à votre discrétion. Sinon vous pouvez aussi vous “inspirer” des fonctions de hashage classique pour inventer la vôtre.

Une seule contrainte est demandée, votre fonction de hashage doit appliquée, au moins, 2 fois la fonction afin d’obtenir le hash (votre schéma utilisant la fonction éponge ne doit pas “seulement” ressortir le résultat de la dernière itération utilisant les données du fichier.
Vous pourrez également choisir arbitrairement d’utiliser N fois la fonction de hashage en fin de processus “d’absorption”.

4.1 Exigences

Il vous est demandé d’implémenter :

1. une fonction de hashage reposant sur la construction avec des fonctions éponges ;
2. Utiliser une fonction de hashage non standard (ou modifier une fonction standard pour qu’elle soit utilisable dans le cadre des fonctions éponges);

5 Génération et vérification d'une signature

Une fois la procédure de création d'un couple de de clés publique/privée, décrite dans la section 3, ces dernières seront utilisées afin de sécurité les transactions en utilisant un mécanisme de signature. Vous avez le choix d'utiliser le mécanisme de signature (El-Gamal, DSA, RSA).

On rappellera brièvement que la signature d'un message se compose des étapes suivantes:
En préambule, il faudra générer les couples de clés publique/privée.
Un message sera ensuite hasher, de sorte que la signature soit toujours appliquée sur un bloc de taille constante (qui sera, bien sûr en rapport avec le mécanisme de signature).
Enfin, le hash / l'empreinte est signé en utilisant la clé privée ; en conséquence, la vérification d'une signature se fait avec la clé publique correspondante

Les signatures devront être écrites dans un fichier en utilisant un formatage de votre choix.

5.1 Exigences

Il vous est demandé d'implémenter :

1. Utiliser des grands entiers premiers (au moins 512 bits) ;
2. Proposer une signature alternative, par exemple DSA (avec les tailles de module réduites) ou bien RSA;

5.2 Recommandations

Si vous le souhaitez, vous pouvez :

1. Implémenter un système de certificat (la clé publique de Alice est signée avec la clé privée de Rémi, par exemple, qui est clairement un tiers de confiance);

6 Mais au fait, c'est quoi une "Block-chain" ?!?

Nous allons présenter le fonctionnement d'une block-chain de façon succincte. Pour ceux intéressés vous pouvez lire le livre que votre professeur a ajouté sur le moodle de l'UE ou bien des articles de recherche (voir notamment Google Scholar).

Globalement, le but d'une block-chain est de permettre à n'importe quel utilisateur d'inscrire une transaction (typiquement une opération de paiement) dans la block-chain de sorte que cette dernière peut être vérifiée à tout moment par toute personne sans qu'elle puisse être modifiée.
La block-chain permet donc d'assurer l'authentification (de la personne qui ordonne le virement depuis son compte), l'intégrité et la non-répudiation.

Pour ce faire un bloc est composé d'un petit nombre transactions, typiquement de 1 à 50. Une transaction est une opération du type :

$\left\{ \texttt{ID user\#1 -> ID user\#2 :: montant \$\$} \right\}$ `-> Signature user#1`

qui signifie que l'utilisateur #1 ordonne le virement à l'utilisateur #2 d'un certain montant, l'ensemble de ces informations est signé avec la clé privée de l'utilisateur #1.

Le bloc N est typiquement constitué de la façon suivante :

`Transaction #1`
`Transaction #2`
`Transaction #3`
`Transaction #4`
`Transaction #5`
$\texttt{H}\big(\texttt{Bloc } N-1\big)$
`Random "Salt" String`

La sécurité de l'ensemble de la block-chain repose sur les deux derniers éléments. D'une part, chaque bloc contient le hash de bloc précédent. Ainsi les blocs dépendent tous du bloc précédent créant ainsi une longue chaîne de sorte que le dernier bloc dépend de tous les précédents. Modifier une transaction au milieu de la chaîne n'est possible que si vous arrivez à modifier tous les blocs suivants (en modifiant systématiquement le hash du bloc précédent).

Le second mécanisme repose sur le `Random "Salt" String` ; le rôle de ce dernier est d'assurer que la chaîne est sécurisée en ajoutant volontairement un calcul très complexe pour valider chaque bloc de transaction. L'exemple que nous utiliserons sera le suivant : le `Random "Salt" String` est une suite de bits aléatoirement générer de sorte que le hash de l'ensemble du bloc N (incluant les transactions, mais aussi le hash du bloc précédent et le `Random "Salt" String`) se termine par B bits 0.
Plus ce nombre B est important, plus difficile sera la validation de chaque bloc et, donc, plus il sera difficile d'ajouter et/ou de supprimer des transactions dans les blocs précédents ... car l'ensemble des blocs suivants doivent être modifiés.
Ainsi, ce n'est généralement pas les utilisateurs qui valident eux-mêmes les blocs de transactions, mais des "mineurs" équipés d'une puissance de calcul énorme (reposant massivement sur des ASICs dédiés ou des GPUS et des centaines de kW...).

Quelques généralités sur les block-chain, les transactions sont validées par bloc et de façon non instantanée, puisqu'il faut faire une "preuve de calcul" pour valider le bloc.
La preuve de calcul est valorisée afin d'encourager le plus d'utilisateurs possible à essayer de "miner des blocs" ; le bitcoin (qui est une monnaie reposant sur la block-chain) offrait 1bitcoin par bloc miné (miner des bitcoins signifie calculer le `Random "Salt" String` permettant de valider un bloc).
Enfin, la block-chain est distribuée, ce qui signifie qu'un mécanisme supplémentaire de consensus est à l'œuvre ; typiquement, imaginons le cas où Rémi souhaite faire 1 transaction et Alice aussi. Ils envoient chacun leurs transactions à toutes les autres personnes utilisant la block-chain. Certains utilisateurs reçoivent d'abord la transaction de Rémi qui permet de former un bloc et commencent à miner ... d'autres utilisateurs reçoivent la transaction de Alice, forme un autre bloc, mais minent de leur côté.
Ce cas peut conduire "un fork" dans la block-chain ... mais cette dernière est diffusée en permanence à tout le monde. L'un des deux forks va donc finalement s'imposer à la majorité des utilisateurs, l'autre sera rendue obsolète.

6.1 Exigences

Il vous est demandé d'implémenter :

1. Avant la soutenance, vous devez créer une block-chaine de quelques blocs (disons une dizaine), pour permettre une vérification durant la soutenance ;
2. Bien sûr cela nécessite d'avoir pu créer quelques utilisateurs.
3. La vérification doit être "verbeuse" , vous devez vérifier les signatures dans chaque bloc, vous devez afficher le hash du bloc, le sel, etc....;
4. Toute amélioration est la bienvenue.

6.2 Recommandations

Réaliser l'ensemble de la block-chain est super ; toute amélioration sera la bienvenue.

7 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 soutenances se font dans mon bureau. Vous devriez pouvoir exécuter votre code python2/3 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 block-chain, 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 oeuvre et, si des choix particuliers ont été faits (par exemple utilisation d'une bibliothèque très spécifique, quelle fonction de signature, quelle fonction de hashage, 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 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 !