Compte Rendu TP1-2

Cryptage RSA et Feistel

Abdelhamid KACIMI – Aimeric MONTANGE

MI44-P19

Établissement :

Université de Technologie de Belfort-Montbéliard

Professeur TP :

Mahjoub Dridi

Responsable UV :

Abdeljalil ABBAS-TURKI

- **TP1 Cryptographie RSA (Codage en C) :**

L'objectif de ce TP est de simuler une communication sécurisée RSA pour faire l'échange d'un mot de passe entre deux personnes, Alice et Bob.

Ce programme se détache en deux grandes parties, la première partie permet de générer les clé public et privée des deux personnes en :

- Générant deux nombres aléatoires P et Q
- Calculant N et ϕ(n)
- Générant e (public) tel que PGCD(e, ϕ(n))=1
- Calculant d (privée) tel que d*e(mod ϕ(n))=1

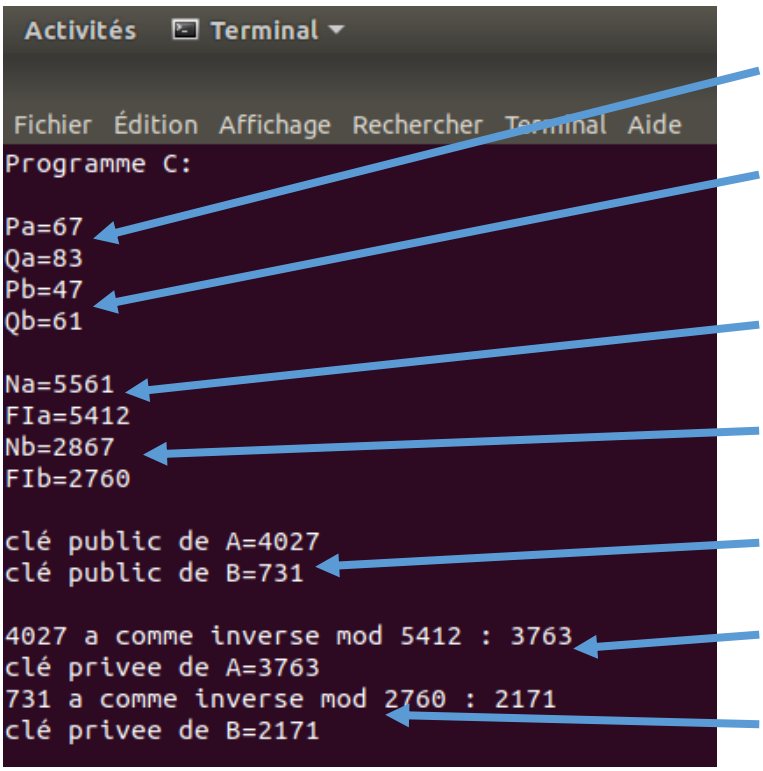
Cette partie du programme comprend les fonctions suivantes :

- *int CreationPetQ() :* Permet de créer deux entiers naturels aléatoires compris entre 1 et 100 pour les attribuer à P et Q (nous avons restreint la dimension de P et Q afin de permettre une compréhension plus rapide de la procédure RSA).

- int Choix_e(int Fi) : Permet créer aléatoire la clé public entre 1 et ϕ(n).

- int algoEuclide (int nb, int mod) : Permet de trouver l'inverse modulaire de nb modulo mod, donc permet d'inverser la clé public modulo ϕ(n) afin d'obtenir la clé privée d.

Jeu d'essai :



Création aléatoire de P et Q pour Alice

Création aléatoire de P et Q pour Bob

Formation de Na et ϕ(n) pour Alice

Formation de Nb et ϕ(n) pour Bob

Choix de la clé public pour Alice et Bob

Calcule de l'inverse de (e mod Fi) pour Alice et création de la clé privée

Calcule de l'inverse de (e mod Fi) pour Bob et création de la clé privée

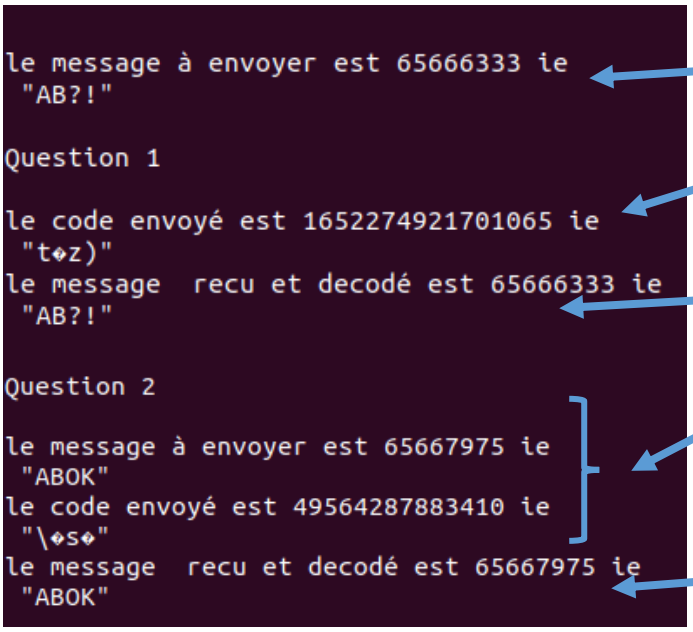
La deuxieme partie de ce programme est de crypter un message avec les caractéristiques d'Alice pour pouvoir l'envoyer à Bob pour que celui-ci le decrypte et puisse lire le message original. Si le message recu est identique alors Bob devra faire de meme pour envoyer une reponse a Alice.

Pour cela nous avons créer plusieurs fonctions :

- int ExponentiationMod(long Messagebis, int Cle, int modulo) : Permet de calculer $M^e mod\ N$ avec e grand, grâce à la méthode d'exponentiation modulaire afin de crypter le message.

- void CodageBinaire(int exposant, int* tableau) : Permet de renvoyer un tableau de 0 et 1 correspondant au code binaire de l'entier, ici l'exposant e, ou d.

Jeu d'essai : Le message à coder est AB?! comme demandé dans l'énoncé.



Formation du code correspondant au message (code ascii de chaque caractère dans notre cas)

Cryptage du message par Alice ($C = M^{eb} mod\ N$b) et envoie du code correspondant : t?z)

Décryptage du message par Bob ($M = C^{db} mod\ N$b) (le code reçu est bien le bon.)

Comme le message reçu est le bon alors Bob crypte le message ABOK et l'envoie à Alice. $C = M^{ea} mod\ Na$

Le code reçu est décrypté par Alice, et le Alice peut lire la réponse de Bob. $M = C^{da} mod\ Na$

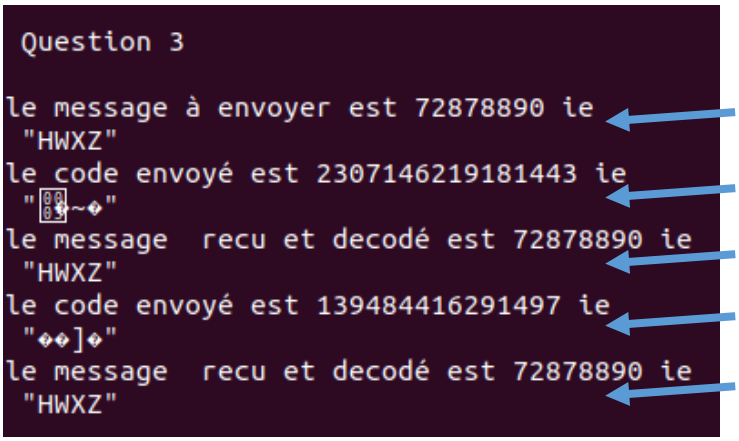
Afin d'afficher les codes des messages envoyés et reçus, nous avons fait plusieurs fonctions d'affichage de tableaux :

- void AfficheTableauNombre (int* tableau) : affiche le code correspondant à un tableau de caractère (table ascii).
- void AfficheTableauCaractere (int* tableau) : affiche le tableau de caractère tel quel
- void MainAfficheEnvoie(int* tableau)
  void MainAfficheEnvoieAvantCodage(int* tableau)
  void MainAfficheRecu(int* tableau) : affichent les codes et les caractères des codes avec une indication pour savoir si il s'agit du code, du message simple ou crypté.

La troisième partie de ce TP est de générer un nombre aléatoire correspondant à 4 caractères, de le crypter avec les caractéristiques de Bob et de l'envoyé à celui-ci. Bob ensuite devra le décrypter et le crypter avec les caractéristiques d'Alice, qui sera par la suite décryptée par Alice.

Les fonctions utilisées sont les mêmes que celle de la partie 2.

Jeu d'essai :



Création aléatoire d'un code correspondant à 4 caractères.

Cryptage de ce code par Alice et envoie

Décryptage du code par Bob

Cryptage de ce code par Bob et envoie

Décryptage du code par Alice

Ainsi l'échange entre Bob et Alice s'est bien déroulé car chacun d'eux est bien retourné sur le message d'origine.

- **TP2 Cryptographie par bloc (Codage en C) :**

L'objectif de ce TP est d'introduire les notions du OU exclusif, de tours et de chiffrement par bloc vus en cours. Il permettra aussi de bien comprendre le cryptage Feistel.

Question 1 : Dans cette question il s'agissait de construire deux fonctions permettant d'encrypter et décrypter les lettres majuscules et quelques caractère spéciaux en binaire. Ses deux fonctions sont :

- void encrypt_binaire (char a, int* enc)

- int decrypt_binaire (int* tab)

Ses deux fonctions stockent le code binaire dans un tableau de 5 int.

Question 2 : Dans cette question il s'agissait de construire une fonction avec comme paramètre un message (2 caractères) et une clé (même taille) et d'effectuer un décalage du message vers la gauche et une addition binaire entre la clé et le message. Voici la fonction et celle utilisée lors de son exécution :

- void function (char* lettrePourCodage, int* key, char* codeRetour)

- void decalage(int* msg) : permet d'effectuer un décalage binaire vers la gauche sur un tableau de 8 caractères.

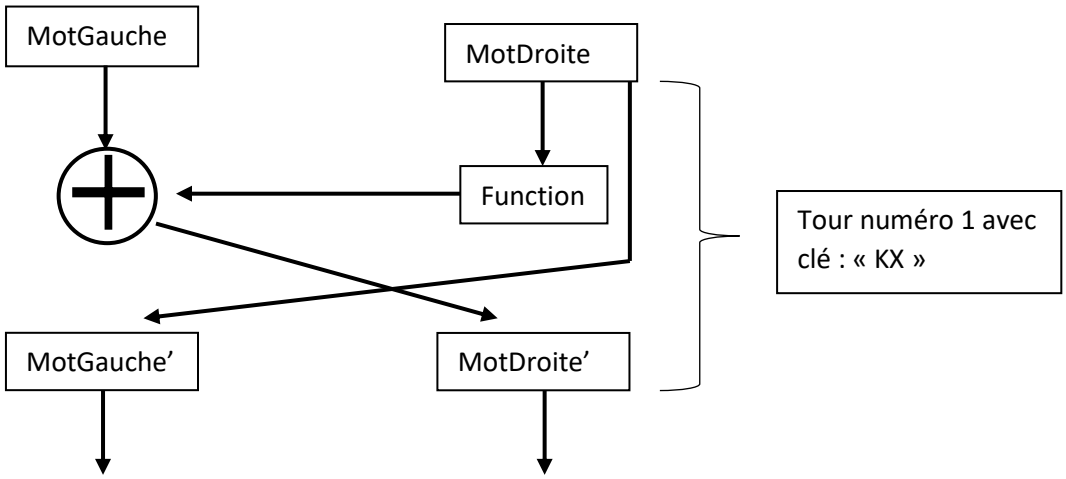
- void concat (int* t1, int* t2, int* msg) : permet de concaténer deux tableau binaire en un unique tableau.

- void addition10 (int* msg, int* key) : permet d'effectuer le ou exclusive entre deux nombres binaire de 10 caractères.

Question 3 et 4 : Il est demandé de former la fonction encrypt_Feistel, permettant de prendre en paramètre un message de 4 caractères et une clé « KXCX » et d'en déduire le chiffrement de quelques messages.

Le principe d'encryptage de Feistel est de prendre un mot ici de 4 caractères, de le diviser en deux, donc de deux caractères (gauche et droite), la clé et les messages sont transcrits en binaire. Avec la partie droite on effectue un décalage gauche d'une unité et on l'additionne à la clé (de deux caractères qui est différentes selon le tour du cryptage), on additionne le résultat obtenu avec la partie gauche du message. Ceci correspond à un tour. On recommence ensuite avec la partie droite du tour précédant comme partie gauche et la partie gauche modifiée du tour précèdent en partie droite du message, la clé change également.

Schéma explicatif du principe :



Dans la fonction encrypt_Feistel, nous utilisons plusieurs fonctions intermédiaires telle que :

- void k_indice (int i, char* mdp, char* K, int* bK): Permet de prendre la bonne clé à chaque tour i, et renvoie donc la clé de deux caractères encryptée en binaire.

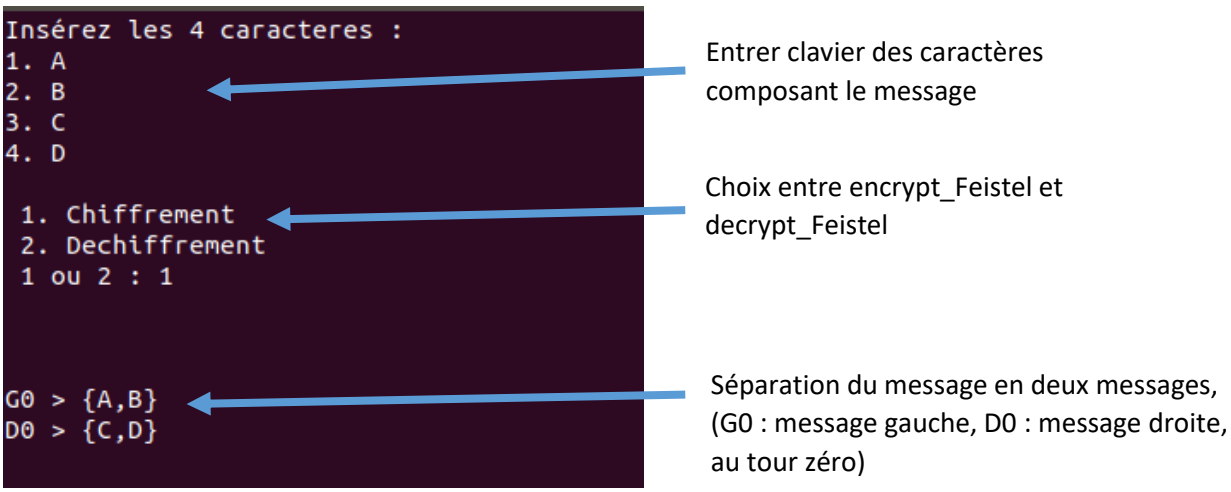
- void affiche10 (int* tab) : permet l'affichage d'un tableau de 10 caractères.

- void affiche5 (int* tab) : permet l'affichage d'un tableau de 5 caractères.

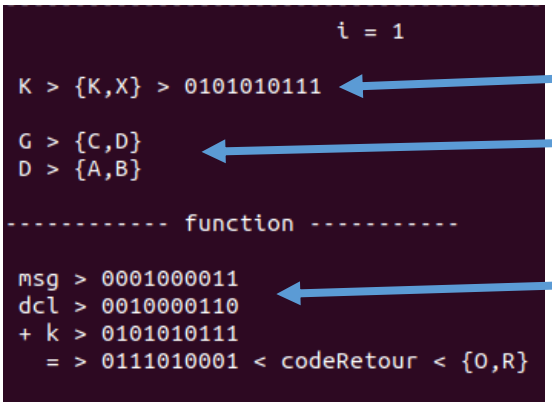
- void function (char* lettrePourCodage, int* key, char* codeRetour) :

- void addition5 (int* rec, int* ajout) : permet d'effectuer le ou exclusive entre deux nombres binaire de 5 caractères.

Jeu d'essai :



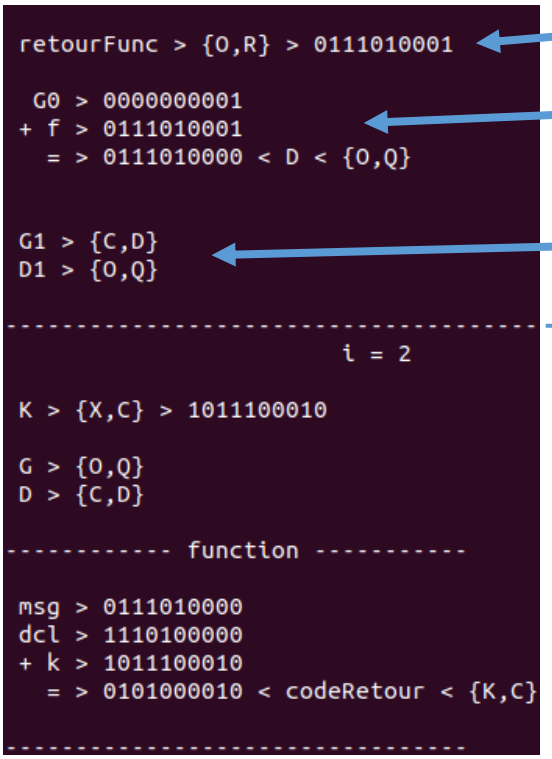
1e Tour :



Clé pour le tour 1 KX : cryptage binaire de la

Echange de position entre gauche et droite du message

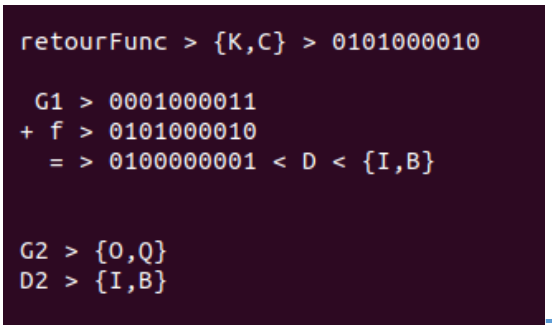
Application de function sur G0, code retour : {0,R}



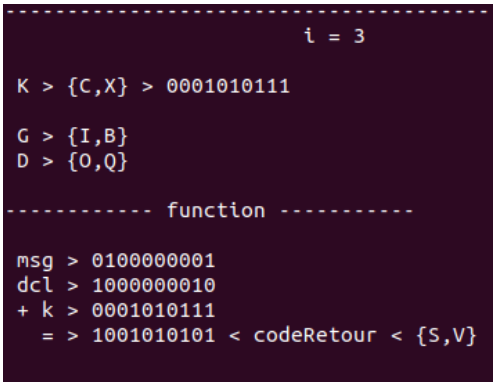
Cryptage binaire du code retour

Application du ou exclusif entre mot gauche et le code retour

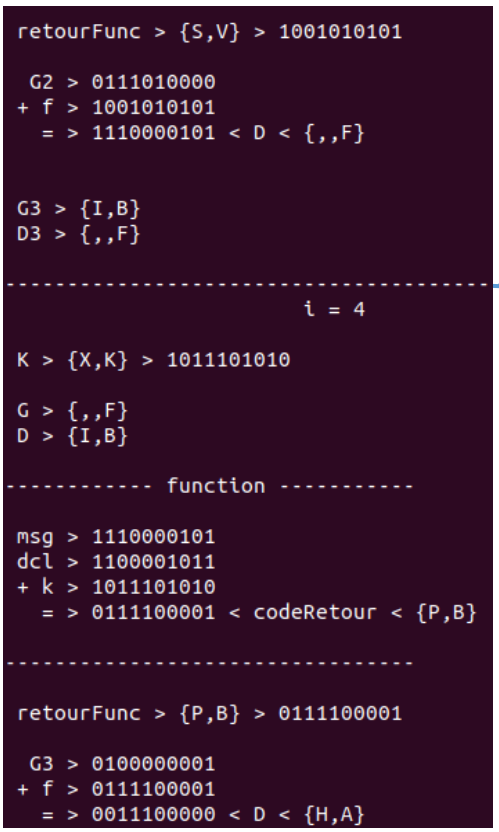
Nouveau mot formé après le tour 1



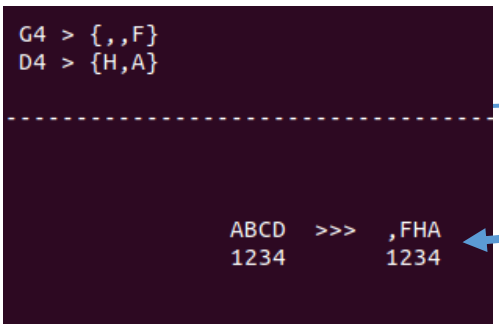
Tour 2 avec la clé XC



Tour 2 avec la clé CX



Tour 4 avec la clé XK

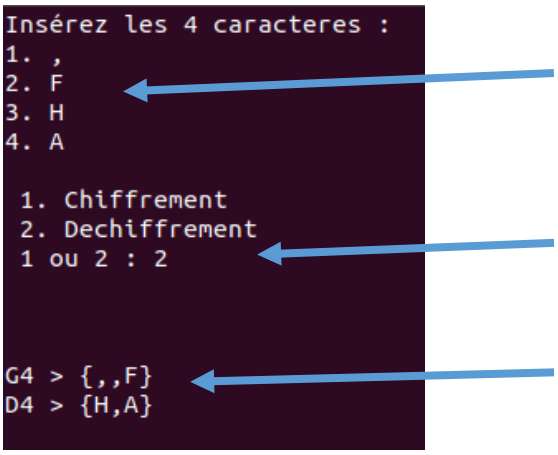


Code du mot « ABCD » : « ,FHA »

Question 5 : On nous demande de reprendre les fonctions précédentes afin de former la fonction inverse de encrypt_Feistel, c'est-à-dire de retrouvé le message à partir d'un code.

Pour cela nous devons effectuer les opérations dans le sens inverse, se sont donc des manipulations d'indice dans les tableaux qui sont gérés.

Jeu d'essai :



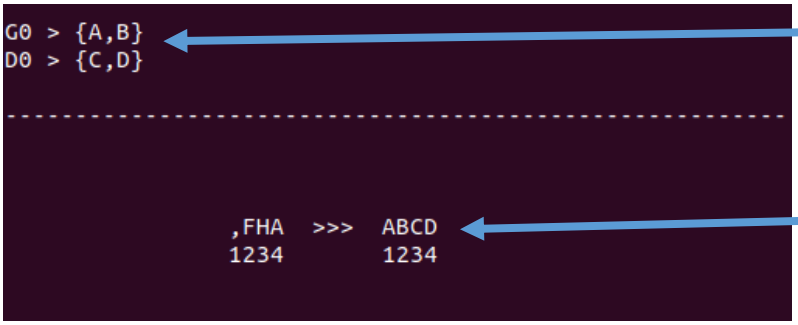
Code a décrypté : « ,FHA »

Mode déchiffrement voulu

Formation des deux parties du code du niveau 4 avant les 4 tours de décodage

Algorithme similaire à l'exemple précèdent avec les 4 tours de codage.

Résultat obtenu :



Mot obtenu après 4 niveau de décodage avec les 4 différentes clés

Le mot correspondant au code est bien « ABCD » comme souhaité

- ## Conclusion

Ce TP nous a permis de bien prendre en main et de comprendre les principes de cryptage par bloc (Feistel), et de cryptage RSA. L'implémentation de ses deux programmes ne sont pas des approches de chiffrement extrêmement fiable mais ceux-ci nous ont aidés à bien saisir les nuances de ses cryptographie.