esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

TP n°4 INF1031

Fichiers

Arnaud Bannier
Nicolas Bodin
Matthieu Le Berre

1. Modification du programme Enigma

Il y a quelques temps, vous avez dû coder le chiffrement Enigma. Lors de la guerre, les paramètres de la machine était distribués à l'ensemble des opérateurs chaque mois sous la forme de livre. L'objectif de cet exercice est de simuler ce concept de distribution de clés et de l'ajouter à votre programme. Le livre est alors vu sous la forme d'un fichier texte composé de plusieurs lignes, et dont chaque ligne indique :

- le numéro du jour du mois sur deux chiffres,
- le numéro du premier rotor à utiliser sur un chiffre suivi de sa position initiale sur une lettre,
- le numéro du deuxième rotor à utiliser sur un chiffre suivi de sa position initiale sur une lettre,
- le numéro du troisième rotor à utiliser sur un chiffre suivi de sa position initiale sur une lettre,
- les permutations du tableau de connexion sur 6 groupes de deux lettres.

Chaque élément est séparé du précédent par un espace, et chaque paramètre journalier est séparé du précédent par un retour à la ligne. Réalisez donc les exercices suivants afin de mettre en place cette procédure.

Exercice 1.

1.1) Rédigez tout d'abord un fichier contenant les paramètres de la machine Enigma sur les 5 premiers jours du mois.

1.2) Utilisez la fonction `fscanf()` afin de récupérer la ligne de paramètres correspondant au jour numéro 1. Définissez et complétez pour cela une structure contenant les champs utiles au stockage des paramètres.

1.3) Votre code contient déjà la déclaration de trois rotors. Modifiez votre fonction de chiffrement afin qu'elle prenne en compte l'ordre de ces rotors donnés par le fichier de configuration.

1.4) Modifiez votre fonction de chiffrement pour prendre en compte la position initiale de chaque rotor.

1.5) Les paramètres du tableau de permutation sont donnés par le fichier de configuration sous la forme de 6 groupes de 2 lettres. Définissez une nouvelle fonction permettant de modifier la permutation `connexions` en fonction de ces paramètres.

1.6) Modifiez enfin votre fonction de chiffrement afin d'écrire et lire les textes chiffrés et clairs directement à partir de fichiers texte passés en argument de la ligne de commande.

2. Chiffrement symétrique

Le *chiffre de Vernam* est un système de chiffrement prouvé sûr ; c'est à dire impossible à casser en théorie. Le principe est le suivant. Soit m le message à chiffrer (*plaintext*) de longueur n_m. Il suffit de créer une clé aléatoire k de longueur n_m et de réaliser l'opération de ou exclusif bit à bit entre le clair et la clé pour produire le message chiffré c. Le XOR étant une fonction involutive ($(A \oplus B) \oplus B = A$), le déchiffrement du message se fait de façon analogue au chiffrement. Le principal problème de ce chiffrement vient du fait qu'il faut que l'émetteur et le destinataire possède la même clé, qui est de la même taille que le message. Étant donnée qu'il est très difficile en pratique d'échanger des clés de façon de sûre, le système est légèrement modifié, pour réaliser un *chiffrement par flot*.

Puisqu'il est pratiquement impossible d'obtenir une clé de la même taille que celle du message, nous utiliserons une clé réduite, qu'un humain peut par exemple retenir. Cette clé k_{init} est donc un entier. Une suite pseudo-aléatoire est créée à partir de la clé, la taille de cette suite étant de n_p octets. Pour générer le premier octet k_0 de ce flot, nous réalisons l'opération suivante :

$$k'_0 = ak_{init} + b \mod n,$$

avec a, b, et n des constantes définies en dur dans le programme. k'_0 étant un entier, il convient de récupérer uniquement les 8 derniers bits afin d'obtenir k_0 qui est formé d'un flux aléatoire de bits. Le message est alors découpé en blocs de 1 octet notés m_i. Le premier octet chiffré est alors $c_0 = k_0 \oplus m_0$. Les octets suivants sont obtenus de façon analogue, en utilisant les formules suivantes :

$$k_{i+1} = ak_i + b \mod n \text{ et } c_i = k_i \oplus m_i \text{ pour } 0 \leq i \leq n_p.$$

Exercice 2.

Cet exercice consiste à chiffrer un message en suivant le protocole précédemment décrit. Vous devez donc :

2.1) ouvrir le fichier contenant m.

2.2) ouvrir le fichier qui contiendra c en s'assurant qu'il n'existe pas déjà. Si c'est le cas, vous devez demander à l'utilisateur ce qu'il souhaite faire (écraser ou nouveau fichier).

2.3) parcourir le fichier contenant le message, octet par octet.

2.4) pour chaque octet, le chiffrer à l'aide d'un XOR avec la clé k_i calculée à l'aide de la formule ci-dessus, et écrire ce bloc dans le nouveau fichier.

2.5) fermer les fichiers ouverts.

Aucun prototype ne vous est donné, à vous de choisir quelles fonctions peuvent vous être utiles.