TD 1 de Cryptographie

INSA CVL

18 septembre 2020

Récupérez sur Célène l'archive correspondant au TP 1. Elle contient deux fichiers sources : `otp.c` et `cesar.c`. Dans ce premier TP, nous allons travailler sur deux chiffrements très simples, le chiffrement par masque jetable et le chiffrement de César. En général :

- Un chiffrement utilise une clé générée aléatoirement. On suppose que cette clé est secrète, c'est à dire qu'elle n'est connue que des quelques personnes qui utilisent le chiffrement pour communiquer.
- Un algorithme de chiffrement, qui prend en entrée un message en clair (souvent une chaîne de caractères) et la clé secrète, et qui génère un message chiffré.
- Un algorithme de déchiffrement, qui prend en entrée un message chiffré et la clé secrète, et qui renvoie le message en clair.
- Le but d'un chiffrement est qu'il soit impossible de retrouver le message en clair depuis le message chiffré lorsqu'on ne connaît pas la clé secrète. Lorsqu'ils existent, les algorithmes qui consistent à retrouver le message sans connaître la clé sont appelés des attaques.

1 Chiffrement par Masque Jetable

Exercice 1. Le Chiffrement par masque jetable consiste à appliquer un *"ou exclusif"* bit à bit entre le message en clair et la clé secrète : soit un entier l représentant la taille du message, un message $m \in \{0,1\}^l$ et une clé $k \in \{0,1\}^l$, le calcul du chiffré c revient à calculer $c = m \oplus k$.

a. Rappelez la table de vérité du $\oplus$. À votre avis, comment déchiffre-t-on les messages ?

b. Implémentez le *ou exclusif* entre deux chaînes de caractères dans le fichier `otp.c`. Le prototype de la fonction est le suivant : `Xor(char * x, char * y, char * z)`. À la fin de la fonction, z doit contenir la valeur de $x \oplus y$. *(rappel : en C, le "ou exclusif" bit à bit se fait a l'aide de l'opérateur " ^ ").*

c. Implémentez les fonctions `encrypt` et `decrypt`. Testez que tout fonctionne bien en exécutant la fonction `main`.

d. La sécurité de ce chiffrement est considérée comme étant *parfaite*, du moment que l'on utilise la clé (le *masque*) pour ne chiffrer **qu'un seul message**, d'où le nom *masque jetable*. La sécurité parfaite veut dire que pour générer le chiffré c, il est équivalent de choisir une clé k aléatoirement dans $\{0,1\}^l$ et de calculer le chiffré $c = m \oplus k$, ou de choisir c aléatoirement dans $\{0,1\}^l$. Ainsi, le message chiffré semble être parfaitement aléatoire pour un utilisateur qui ne connaît pas la clé. Démontrez ce résultat.

e. Si on chiffre plusieurs messages avec la même clé, on s'expose à des attaques dites *à clairs choisis*. On va considérer qu'un attaquant possède deux chiffrés ayant utilisé la même clé, et que ces chiffrés chiffrent deux messages ayant été choisis dans un espace restreint d'une dizaine de messages (par exemple ceux décrits dans le tableau `tabl` du fichier `otp.c`). Proposez une attaque qui permet de retrouver les deux messages chiffrés, ainsi que la clé de chiffrement, à partir de la connaissance des deux chiffrés et des 10 messages possibles.

f. Implémentez la fonction `void generate_challenge(char * cipher_r, char * cipher_s)` qui génère une clé aléatoire, chiffre deux messages choisis aléatoirement dans `tabl`, et qui stocke les deux chiffrés obtenus dans dans `cipher_r` et `cipher_s`.

g. Implémentez votre attaque dans la fonction `void attack(char * cipher_r, char * cipher_s)`. La fonction devra afficher le déchiffrement de `cipher_r` et `cipher_s`, et la clé secrète correspondante. Combien trouvez-vous de clés possibles ? Expliquez.

2 Chiffrement de César

Exercice 2. Le Chiffrement de César consiste, pour une clé secrète k (avec $0 \leq k \leq 25$), à décaler chaque lettre du message de k lettres dans l'alphabet (lorsque l'on atteint la lettre "Z", on revient à la lettre "A"). Par exemple, "CRYPTO" chiffré avec la clé $k = 3$ donnera "FUBSWR".

a. Implémentez les fonctions `encrypt` et `decrypt`. Testez que tout fonctionne bien en exécutant la fonction `main`.

b. Sachant que la lettre "E" est (de loin) la plus fréquente en français, proposez une attaque pour déchiffrer un message en français chiffré à l'aide du chiffrement de César, sans utiliser la clé secrète.

c. Implémentez la fonction `void generate_challenge(char * cipher)` qui génère une clé aléatoire, chiffre un messages (par exemple `str`), et qui stocke le chiffré obtenu dans `cipher`.

d. Implémentez votre attaque dans la fonction `void attack(char * cipher)`. La fonction devra afficher le déchiffrement de `cipher`, et la clé secrète correspondante.

3 Chiffrement de Vigenère (Optionnel)

Exercice 3. Le chiffrement de Vigenère est une généralisation du chiffrement de César : la clé est un mot dont chaque lettre correspond à un décalage (avec "A"=1, "B"=2, ..., "Z=26"). Chaque lettre du message est décalé dans l'alphabet à l'aide de la lettre correspondante de la clé. Par exemple, "CRYPTO" chiffré avec la clé "ABCDEF" donnera "DTBTYU". Si le message est plus long que la clé, on recommence du début de la clé.

a. Montrez qu'il est possible d'attaquer le chiffrement de Vigenère de la même façon que le chiffrement par masque jetable. Montrez que si le message est deux fois plus long que la clé, on peut adapter l'attaque pour qu'elle fonctionne même si l'attaquant ne connaît qu'un seul message chiffré.

b. Expliquez comment étendre l'attaque basée sur la fréquence de la lettre "E" au cas de Vigenère si l'on suppose que la taille du message est beaucoup plus longue que la taille de la clé, et que la taille de la clé est connue.

c. Si vous êtes motivé, vous pouvez implémenter le chiffrement de Vigenère et les deux attaques que vous avez proposées.