PRESENT24

Ce devoir-maison consiste d'abord en l'implantation en Python d'un chiffrement par bloc donné et en l'implantation ensuite de l'attaque par le milieu sur la version double de ce chiffrement

Ce devoir-maison comporte 3 fichiers .py :

-chiffrement.py
-dechiffrement.py
-attaque.py

1) chiffrement.py

Fonctionnalités

Chiffrement: Cette fonction prend une liste de 64 bits en entrée ainsi qu'une liste de 11 sous-clés de 48 bits chacune, et renvoie une nouvelle liste de 64 bits obtenue à partir d'une série d'opérations de substitution et de permutation.
Elle utilise les fonctions Permutation, Substitution et Cadencement_de_clé qui ont été implémentés suivant les indications donnés .

2) dechiffrement.py

Fonctionnalités

Déchiffrement : Cette fonction prend en entrée l'état initial Etat et la clé de chiffrement clé en format binaire et renvoie le message déchiffré en hexadécimal.

La fonction print affiche un message de bienvenue ainsi que les options de menu pour chiffrer ou déchiffrer un message.

Le programme de menu permet à l'utilisateur de choisir entre le chiffrement ou le déchiffrement d'un message. Il récupère le message clair ou chiffré ainsi que la clé à utiliser pour le chiffrement ou le déchiffrement, puis affiche le résultat.

3) attaque.py

Fonctionnalités :

En plus des fonctions pour le chiffrement et le déchiffrement, ce code implémente une attaque par le milieu sur la version double de l'algorithme de chiffrement PRESENT24.
La fonction Attaque(clair, chiffre) prend en entrée le texte clair et le texte chiffre
Tout d'abord, elle crée toutes les combinaisons possibles de clés de chiffrement en remplissant une liste 'cles_possibles'.
Ensuite, pour chaque clé possible, elle chiffre le texte clair en utilisant la fonction 'chiffrement(clair, cle)' et stocke le resultat dans une liste 'chiffres'. Elle déchiffre également le texte chiffré avec la même clé en utilisant la fonction 'Dechiffrement(chiffre,cle) ' et stocke le resultat dans une liste 'clairs '
Les listes 'chiffres ' et 'clairs' sont ensuite triés par ordre croissant en utilisant la fonction 'sorted()' en spécifiant le paramètre 'key' pour trier les elements entiers hexadécimaux.
La fonction 'Trouver_elements_communs(liste1,liste2) est appelée avec les listes 'chiffres' et 'clairs' en tant que paramètres. Cette fonction renvoie tous les éléments communs aux deux listes.
Enfin la fonction Attaque(clair, chiffre) renvoie la liste des éléments communs aux deux listes triées, qui représentent les clés potentielles de chiffrement.

Étant donné qu'il y a beaucoup de clés possible avec un seul couple clair/chiffré, on réitère l'attaque sur un deuxième couple clair/chiffré et on compare les couples de clés potentielles trouvés a la première attaque sur m1/c1 ainsi que les couples de clefs potentielles de la deuxième attaque sur m2/c2. Les couples de clefs identiques sont candidates pour être la bonne clé.

Dans notre cas nous avons utilisé les couples clairs/chiffrés suivant : (couples d'Alexandre Mihet)
(m1,c1) = (ddae7b, 856673)
(m2,c2) = (fd1dc1,5aad58)

Notre attaque a donné les **deux couples de clés potentielles** suivantes :
Couple1 :
k1 = d4b8d8 k2 = f7e4c5
Couple 2 :
k1 = b21a67 k2 = 05ac25

4)Auteurs
Ce code a été écrit par Billal MEDOUR et Alexandre MIHET