DM: Implantation d'un chiffrement par blocs en `C`

christina.boura@uvsq.fr

7 avril 2016

1 Généralités

Le but de ce devoir-maison est d'implanter la procédure de chiffrement d'un chiffrement par blocs en `C`. Il ne s'agit pas d'un algorithme existant, mais ses principes de conception sont très proches de certains des chiffrements par blocs modernes.

Modalités pratiques Vous pouvez travailler seuls ou en groupe de 2 personnes au plus. Bien évidement, chaque équipe devra travailler indépendamment des autres. La date limite pour rendre votre projet est le **jeudi 28 avril** à **23 heures**. Aucun retard ne sera toléré et les projets rendus tardivement ne seront pas notés. Les projets doivent être énvoyés par mail à l'adresse `christina.boura@uvsq.fr` sous la forme d'un fichier `.zip` ayant comme nom votre nom de famille (ou les deux noms si vous travaillez en binôme).

Il y a plusieurs façons de coder l'algorithme décrit dans la section suivante. Vous êtes libres de choisir le mode d'implémentation qui vous semble le plus adapté. Cependant, les bons choix d'implémentation qui rendront votre code plus efficace et plus compact recevront des points supplémentaires. Chaque projet qui compilera et qui fonctionnera correctement, c'est-à-dire qui reproduira les vecteurs de test donnés à la fin du sujet, recevra 12 points. 2 points seront donnés pour la lisibilité du code et sa facilité d'utilisation (code bien commenté, présence d'un makefile (si plusieurs fichiers) et d'un fichier README.txt). Finalement, les bons choix d'implementation seront jugés sur 6 points. Un code qui fonctionne et qui utilise peu de mémoire et/ou qui est rapide sera donc mieux noté qu'un code qui fonctionne bien mais qui n'est pas très optimal. Votre code peut être suivi d'un fichier `.txt` ou `.pdf` dans lequel vous expliquerez rapidement vos choix d'implémentation (optionnel).

2 Spécifications du chiffrement par blocs

Le chiffrement par blocs que vous devez implanter, prend en entrée un bloc de message de `64` bits, une clé de `80` bits et produit un bloc chiffré de `64` bits. Il appartient à la famille des chiffrements dits **SPN** (Substitution-Permutation Network). Cette construction est la deuxième famille largement utilisée pour concevoir un chiffrement par blocs, l'autre étant les réseaux de Feistel. Le principe d'un chiffrement SPN est simple : chaque tour du chiffrement est constitué de trois couches :

— Une addition (XOR) de la sous-clé du tour à l'état ;
— une couche de substitution pour assurer la confusion ;
— une couche linéaire pour assurer la diffusion.

Le nombre de tours de ce chiffrement est fixé à `30`.

Algorithme de cadencement de clé La clé du chiffrement est constituée de 80 bits. On l'appellera *clé maître*. Un algorithme, appelé algorithme de cadencement de clé (ou key schedule en anglais) est appliqué à la clé maître afin de produire à partir de celle-ci 31 sous-clés K_i, $1 \leq i \leq 31$, de 64 bits chacune. Les 30 premières sous-clés seront utilisées pour l'opération de l'addition de la sous-clé aux 30 premiers tours, tandis que la dernière sous-clé K_{31} sera additionnée à l'état final pour produire le chiffré.

On note `Etat` le registre de 64 bits qui contiendra l'état. Le bit de poids faible de ce registre, ainsi que de toutes les autres valeurs utilisées, est considéré à droite. On appèle également `Substitution` la

fonction non-linéaire et `Permutation` la fonction linéaire. Chacune de ces fonctions, décrites dans les sections suivantes, prennent en entrée le registre `Etat` et donnent en sortie le même registre après l'avoir mis à jour. L'algorithme suivant donne une description de l'algorithme de chiffrement.

Algorithme 1 : Fonction de chiffrement

Données : Un message m de 64 bits et 31 sous-clés K_i, $1 \leq i \leq 31$, produites par l'algorithme de cadencement de clé

Sortie : *Un message chiffré c de 64 bits*

`Etat` $\leftarrow m$

pour $i = 1$ *jusqu'à* 30 **faire**

  `Etat` $\leftarrow$ `Etat` $\oplus K_i$

  `Etat` $\leftarrow$ `Substitution(Etat)`

  `Etat` $\leftarrow$ `Permutation(Etat)`

`Etat` $\leftarrow$ `Etat` $\oplus K_{31}$

$c \leftarrow$ `Etat`

retourner c

2.1 Addition de la sous-clé à l'état

Étant donnée la sous-clé du tour i, $K_i = \kappa^i_{63}\kappa^i_{62}\cdots\kappa^i_0$ pour $1 \leq i \leq 31$ et l'état actuel `Etat` $= b_{63}b_{62}\cdots b_1b_0$, cette étape consiste juste en l'opération

$$b_j \leftarrow b_j \oplus \kappa^i_j,$$

où $\oplus$ représente l'opération XOR.

2.2 La fonction `Substitution`

La substitution est effectuée à l'aide d'une boîte-S, qu'on notera S, qui prend en entrée 4 bits et qui donne en sortie 4 bits, décrite par le tableau ci-dessous :

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
S[x]	b	f	3	2	a	c	9	1	6	7	8	0	e	5	d	4

Chaque entrée et sortie de la boîte-S est constituée de 4 bits et peut être donc représentée par un chiffre hexadécimal. C'est cette notation hexadécimale qui est utilisée pour décrire l'action de la boîte-S. Par exemple, $\mathtt{0101}_2 = \mathtt{5}_{16} \rightarrow \mathtt{c}_{16} = \mathtt{1100}_2$.

Pour cette étape, l'état de 64 bits $b_{63}b_{62}\cdots b_1b_0$ est divisé en 16 mots de 4 bits $w_{15}w_{14}\cdots w_0$, où $w_i = b_{4*i+3}||b_{4*i+2}||b_{4*i+1}||b_{4*i}$ pour $0 \leq i \leq 15$. La boîte-S est appliquée alors à chaque mot w_i de l'état, transformant tous les mots w_i en $S[w_i]$, pour $0 \leq i \leq 15$.

2.3 La fonction `Permutation`

La couche linéaire est effectuée à l'aide d'une permutation bit-à-bit P. Le bit i de l'état bouge après la couche linéaire à la position $P(i)$, comme décrit par la table suivante :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

2.4 L'algorithme de cadencement de clé

On utilise un registre K de 80 bits pour sauvegarder la clé maître qu'on représente comme $k_{79}k_{78}\dots k_0$. Au tour i, la sous-clé K_i de 64-bits $K_i = \kappa_{63}\kappa_{62}\dots\kappa_0$ est constituée de 64 bits les plus à gauche du registre K :

$$K_i = \kappa_{63}\kappa_{62}\dots\kappa_0 = k_{79}k_{78}\dots k_{16}.$$

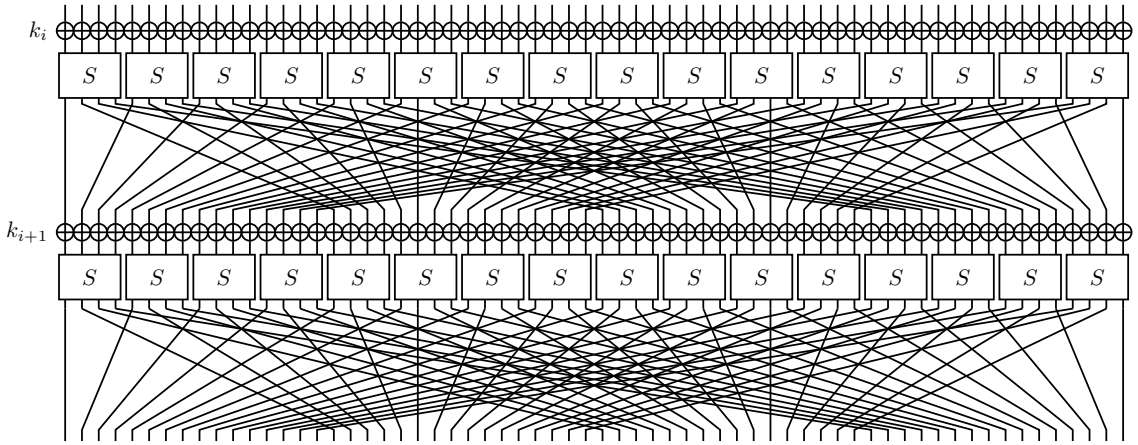
Après avoir extrait la sous-clé du tour, le registre $K = k_{79}k_{78}\dots k_0$ est mis à jour de la façon suivante :

1. $[\mathtt{k}_{79}\mathtt{k}_{78}\dots\mathtt{k}_1\mathtt{k}_0] = [\mathtt{k}_{18}\mathtt{k}_{17}\dots\mathtt{k}_{20}\mathtt{k}_{19}]$
2. $[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}] = \mathtt{S}[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}]$
3. $[\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] = \mathtt{S}[\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] \oplus i$

Dans la première étape le registre est simplement pivoté de 61 positions vers la gauche. Dans l'étape suivante la même boîte-S que celle utilisée pour le chiffrement est appliquée aux 4 bits les plus à gauche du registre. À la dernière étape, on additionne (avec un XOR) les bits $\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}$ du registre à une valeur i qui correspond au numéro du tour que nous sommes en train de traiter. Donc pour le tour 1 on ferra un XOR avec $i = 1$, pour le tour 2 avec $i = 2$ etc.

2.5 Deux tours du chiffrement

La figure suivante représente graphiquement deux tours du chiffrement (tours i et $i+1$).



2.6 Exemple de chiffrement sur 1 tour

Supposons que le message à chiffrer est le m =`fedcba9876543210` et que la clé de chiffrement est la `00000000000000000000`. Le registre `Etat` est alors initialisé avec le message m.

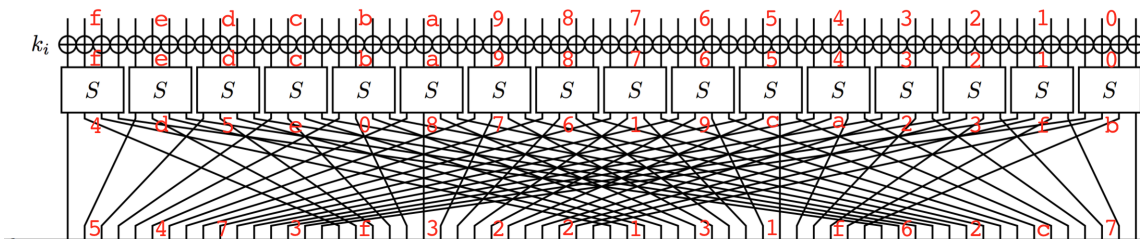
La première sous-clé générée avec l'algorithme de cadencement de clé est K_1 = `0000000000000000`.

La première opération est l'addition bit-à-bit (XOR) de la sous-clé K_1 avec le message. Après cette opération l'état contient alors la valeur `fedcba9876543210`.

Après l'application de la couche de substitution, l'état est mis à jour et contient maintenant la valeur `4d5e087619ca23fb`.

L'étape suivante est l'application de la couche linéaire. Après cette étape, et par conséquent, après la fin du premier tour, l'état contiendra la valeur `5473f322131f62c7`.

Cette procédure peut être visualisée ci-dessous.



3 Vecteurs de test

Les vecteurs de test suivants sont donnés en notation haxadécimale. Le bit le plus faible de chaque valeur est à droite.

Message clair	Clé maître	Message chiffré
0000000000000000	00000000000000000000	83e43b5285ce1abc
0000000000000000	ffffffffffffffffffff	f8606c052dfa323b
fedcba9876543210	ffffffffffffffffffff	23ecf5764ae19d75