

Déchiffre et Delette Partie 1

Chiffrement symétrique avec clé unique.

Introduction :

Nous n'allons pas voir en détail toute l'histoire de la cryptographie.

Depuis le fameux code César qui avait une clé à un nombre unique qu'on additionnait à chaque caractère du message.

Exemple si A=1 ; B=2 ; etc.

et que la clé est 6.

Le A est codé G. Le B est codé H.

Le message 'BABA' donne 'HGHE'.

Ce système est très mauvais car il suffit d'essayer toutes les clés de 1 à 26 ou même de 1 à 100 si on a chiffrer les majuscules, minuscules, caractères spéciaux, les accents, espace, les points, etc.

Meilleur système, on peut avoir une clé différente pour chaque caractère.

Mais tous les systèmes substituant un code différent pour chaque lettre mais identique pour une même lettre (chiffrement mono-alphabétique) est relativement facile à craquer plus le texte est long, si on connaît la langue du message clair. Car les lettres ont une fréquence d'apparition connue pour chaque langue.

Exemple en français la fréquence du "e" dans un texte est plus ou moins 17,26%, le "a" 8,4%, etc.

On va donc programmer :

Un système à clé secrète qui sous certaines conditions est "théoriquement" inviolable.

A la fin de cet atelier, nous verrons une simulation du transfert "quantique" de clé privée dont la sécurité est basée sur la certitude que la clé n'a pas été piratée (ou au contraire qu'elle l'a été. Dans ce cas on ne transmet pas le message).

Liens :

vidéo : <https://www.youtube.com/watch?v=zx9XkeX2YaY>

Théorie

Avec clé privée:

Identique pour le chiffrement et déchiffrement,

la plus longue possible,

les lettres identiques n'ont pas toutes le même chiffrement, contrairement au code césar de base ou par substitution.

En particulier un théorème de Claude Shannon (Ingénieur et mathématicien) prouve en 1949, qu'un message chiffré avec :

une clé de même longueur que le message,

réellement aléatoire, (ne pas prendre un chapitre de livre au hasard)

à usage unique (One Time Pass)

est **indéchiffrable par quiconque ne connaissant pas la clé.**

<https://www.lavachequicode.fr/cryptographie/cle-a-usage-unique>

http://www.acrypta.com/telechargements/fichecrypto_300.pdf

Dans la version moderne de ces systèmes on prend comme valeur du caractère "clair" non pas sa position dans l'alphabet mais son code informatique c-à-d son code ASCII (ou Unicode).

Et l'opération de codage n'est pas une addition mais un "OU exclusif" ; ce qui est très proche :

Un message est une suite de bits.

Ex: "AB" est égal à en ascii à 0100.0001|0100.0010

Une clé est aussi une suite de bits aléatoires.

Ex: 0101.1001|1111.0110 ("Yö")

On chiffre ainsi

M= 0100.0001|0100.0010

XOR

K= 0101.1001|1111.0110

donne

C= 0001.1000 ...etc

On déchiffre ainsi

C= 0001.1000

XOR

K= 0101.1001

donne:

M= 0100.0001

Il vaut mieux grouper les octets de la clé et du message clair par **4** et stocker la valeur numérique de ces 4 octets dans un tableau.

Pourquoi ? Parce que en JS l'opérateur XOR (^) s'opère sur 32 bits max. $32=8 \times 4$.

Pour ça on fera tous les 4 caractères (de gauche à droite) :

$C1 \times 256 \times 256 \times 256 + C2 \times 256 \times 256 + C3 \times 256 + C4 \Rightarrow$ 1 élément du tableau

Explications

$M = C1 \times 256 + C2$ ~ Rajouter 8 zéro binaire à droite et additionner le caractère suivant.

En base 10 pour ajouter 8 zéros on multiplie par $10^8 = 100\,000\,000$

et bien

en base 2 pour ajouter 8 zéros on multiplie par $2^8 = 100\,000\,000$ binaire ~ 256

puis

$M = M \times 256 + C3$ puis $M = M \times 256 + C4$;

Inconvénient : Le transfert sécurisé des clés secrètes (une par message)

Pratique :

Création de la clé aléatoire :

Note :

Dans le prog. principal, on a

```
var o=createKey (plainTxt.length);
```

```
var key=o.key;
```

On écrit donc la fonction :

```
/**
 * Random key creation
 * @param len          Length of plain text
 * @return object with
 *                    key    Created key
 *                    binKey  binary string of key (only to display it)
 */
function createKey (len) {
    var key="";var binKey8=""; var binKey=""; var splitter=".";
    for (i=0;i<len*8;i++) {
        var bit=random();
        binKey8+=""+bit; // binKey8+= String(bit);
        if ( i%8==7) {
            if (i==(len*8)-1) splitter="";
            var c = parseInt(binKey8, 2);
            key+=String.fromCharCode(c);
            binKey+=binKey8+splitter;
            binKey8="";
        }
    }
    return {binKey:binKey , key:key};
}

// ...

// Random utililty function

function random () {
    return Math.floor(Math.random() * 2); // Expliquer
};
```

Notes :

Dans le prog. principal, après

```
var o=createKey (plainTxt.length);
```

```
var key=o.key;
```

On a dans key une clé de même longueur que la message (var plainText)

Regroupement de 4 octets par une valeur de 32bits et rangement dans un tableau (Array).

Note :

Dans le prog. principal, on a

```
// Convert plain text to array of 4 bytes elements
```

```
var arrBy= toArrayOfBy(plainTxt,by);
```

et

```
// Convert key to array of 4 bytes elements
```

```
var keyBy=toArrayOfBy(key,by);
```

La fonction est déjà écrite :

```
/**
 * Creation of an array of 4 bytes (32 bits) elements
 * @param str      A text string (message or key)
 * @param by       Number of bytes (by=4)
 * @return An array of 32 bits elements
 */
function toArrayOfBy (str,by) {
    var arrOut=[]; var c=0;
    for (i=0;i<str.length;i++) {
        c=c*256+str.substr(i,1).charCodeAt(0) ;
        if (i%by==(by-1)) {
            arrOut.push(c);    c=0;
        }
    }
    if (c!=0) arrOut.push(c);
    return arrOut;
}
```

Notes :

Dans le prog. principal, après

```
var arrBy= toArrayOfBy(plainTxt,by);
```

et

```
var keyBy=toArrayOfBy(key,by);
```

On a 2 tableaux de même longueur contenant chacun des valeurs de 32 bits.

Chiffrement (encodage)

Note :

Dans le prog. principal, on a

```
// Ciphering
```

```
var cipherArr=cipher (arrBy,keyBy);
```

On écrit donc le corps de la fonction :

```
/**
 * Ciphering
 * @param msgArr      Message's 32 bits array
 * @param keyArr      Key's 32 bits array
 * @return Ciphered message's 32 bits array
 */
function cipher (msgArr,keyArr) {
    var arrOut=[]; var j=0;
    for (var i in msgArr) {
        if (j>keyArr.length-1) j=0 ;
        arrOut.push((msgArr[i] ^ keyArr[j]));
        j++;
    }
    return arrOut;
}
```

Déchiffrement

Note :

Dans le prog. principal, on a

```
// Ciphering
```

```
var cipherArr=cipher (arrBy,keyBy);
```

On écrit donc le corps de la fonction :

```
/**
```

```
* Unciphering
```

```
* @param cipherArr      Message's 32 bits array
```

```
* @param keyArr          Key's 32 bits array
```

```
* @return Ciphered message's 32 bits array
```

```
*/
```

```
function uncipher (cipherArr,keyArr) {
```

```
    var arrOut=[]; var j=0;
```

```
    for (var i in cipherArr) {
```

```
        if (j>keyArr.length-1) j=0 ;
```

```
        arrOut.push((cipherArr[i] ^ keyArr[j]));
```

```
        //log("cipherArr[i]="+cipherArr[i]+" / keyArr[j]="+keyArr[j]);
```

```
        j++;
```

```
    }
```

```
    return arrOut;
```

```
}
```

Conversion de tableau déchiffré en texte

Note :

Dans le prog. principal, on a

```
// Uncipher array to string
```

```
var unciphText=arrayByToString (uncipherArr,by)
```

La fonction est déjà écrite :

```
/**
 * Reconstruction of plain text
 * @param arr          Unciphered array
 * @param by           Number of bytes (by=4)
 * @return Original text reconstituted
 */
function arrayByToString (arr,by) {
    var str=""; var strBy=""; var c;
    for (var i=0;i<arr.length;i++) {
        var nBy=arr[i];
        for (var j=0;j<by;j++) {
            c=nBy%256;
            if (c!=0) {
                strBy=String.fromCharCode(c)+strBy;
                nBy=Math.floor(nBy/256);
            }
        }
        str+=strBy;strBy="";
    }
    return str;
}
```

*** ON TESTE avec des messages + ou – longs ***

Puis je fais ma démo du programme de
simulation quantique.