Déchiffre et Delettre 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 'HGHG'.

Ce système est très mauvais car il suffit d'essayer toutes les clés de 1 à 26 ou même de 1 à 127 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:
simulation transfert quantique: http://www.pixaline.net/intra/Atelier-2019/crypto/done/quantic/
risque si réutilisation de clé: https://cryptosmith.com/2008/05/31/stream-reuse/
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 Pad ou Masque jetable)

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 (*explication rapide du binaire*)
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=8x**4.**

Pour ça on fera pour chaque caractère (de gauche à droite) :

M= M décalé de 8 bits vers la gauche. On a donc ajouter 8 zéros à droite;
M= M+ caractère suivant.
M est stocké comme élément d'un tableau mémoire (array)

Au tableau: Montrer un exemple de compactage, en décimal sur des formats de 4 chiffres.

Explications

Rajouter 8 zéros à droite et additionner le caractère suivant.

En base 10 si on ajoute 8 zéros à un nombre, on obtient n00 000 000 (équivalent à 10^8) .
On peut dire que multiplier un nombre par 100 000 000 ajoute 8 zéros à ce nombre.

Et bien:

En base 2 si on ajoute 8 zéros à un nombre, on obtient n00 000 000 binaire (équivalent à 2^8 = 256)
On peut dire que multiplier un nombre par 256 ajoute 8 zéros à ce nombre en binaire.

****

Inconvénients: Le transfert sécurisé des clés secrètes (une par message) n'est pas adapté aux transmissions fréquentes et anonymes d'internet par exemple.

En ligne (ou sur mon pc)

Avant la pratique faire la démonstration expliquée du programme terminé...

Pratique:

Création de la clé aléatoire:

Note:
Dans le prog. principal, on a
```
  var o=createKey (plainTxt.length);
  var key=o.key; //var binKey=o.binKey;
```
On écrit donc la function:

```
/**
* 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 k=0; var binKey=""; // var binKey8="";
        for (i=0;i<len*8;i++) {
                var bit=random();
                var k=k+bit;
                // binKey8+= bit.toString() ; // or // binKey8+=""+bit; // binkey8=binkey8+String(bit);
                if ( i%8==7) {
                        key+=String.fromCharCode(k);
                        // binKey8=""; binKey+=binKey8+".";
                        k=0;
                }
                k=k<<1; // or // k*=2;
        }
        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 en une valeur de 32bits et rangement dans un tableau (Array).

Note:
Dans le prog. principal, on a
const by=4; // au début
by donne le nombre de caractères qui vont être regroupés
// Convert plain text to array of 4 bytes elements
var arrBy= stringToArrayBy(plainTxt,by);
et
// Convert key to array of 4 bytes elements
var keyBy=stringToArrayBy(key,by);

La function 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 stringToArrayBy (str,by) {
        var arrOut=[]; var c=0;
        for (i=0;i<str.length;i++) {
                c=(c<<8)+str.substr(i,1).charCodeAt(0); // (c*256)
                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= stringToArrayBy(plainTxt,by);
et
var keyBy=stringToArrayBy(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 function:

```
/**
* 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) { // or // for (var i=0; i<msgArr.length; i++) {
                // add // if (j>keyArr.length-1) j=0 ;
                arrOut.push((msgArr[i] ^ keyArr[j]));
                j++;
        }
        return arrOut;
}
```

Note:
remarquer que si on laisse comme ça, on se demande pourquoi utiliser j!
En fait j servira au cas où la clé est plus petite que le message mais il faut enlever le commentaire.

Déchiffrement

Note:
Dans le prog. principal, on a
```
  // Unciphering
  uncipherArr=uncipher(cipherArr,keyBy);
```

On écrit donc le corps de la function:
```
/**
* 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) {
        return cipher (cipherArr,keyArr) ;
}
```

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 & parseInt("11111111",2);                 // extract current right byte value
                                                                        // or c=nBy&255 ;
                                                                        // or c=nBy%256;
                                                                        // or c=nBy & parseInt("11111111",2);
                                                                        //    because 11111111 binary == 255
                        if (c!=0) {                                     // If it's not the last octets
                                strBy=String.fromCharCode(c)+strBy;     //    put char at the left of tmp string
                                nBy=nBy>>8;                             //    suppress right byte
                                                                        //    or nBy=Math.floor(nBy/256);
                        }
                }
                str+=strBy;strBy="";
        }
        return str;
}
```

Note: Expliquer avec le programme dans 'done/' et représenter au tableau :

P l e a
01010000.01101100.01100101.01100001

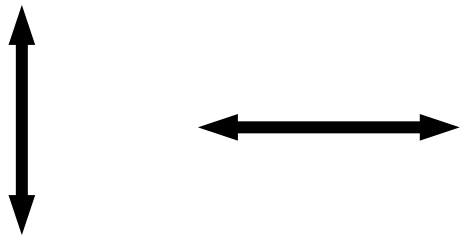
*** ON TESTE avec des messages + ou – longs ***

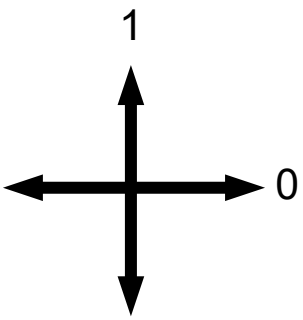
Démo de simulation de transmission quantique de clés.

La cryptographie quantique n'existe pas. Le chiffrement se fait avec la méthode de clé secrète à usage unique vu précédemment.
Ce qui existe est la transmission de clés secrètes en étant sûr que la clé n'a pas été piratée sur le réseau. Si au contraire il y a un doute, on jette la clé. Et c'est cette garantie qui est obtenu grâce à certaines principes de la physique quantique.
En particulier on utilise la polarisation de photons unique.
Un photon peut être représenté comme étant un grain de lumière (même si toute représentation d'un photon que ce soit un corpuscule ou une onde, est aussi fausse l'une que l'autre)

Au tableau:

Un photon peut être polarisé (sens de sa vibration) :

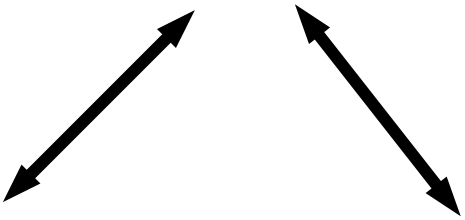


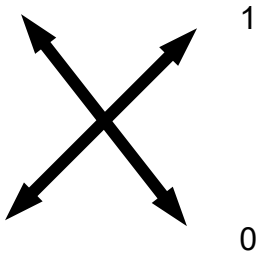


Verticalement ou Horizontalement;

Bits: 1 0

On va parler de base Orthogonal





Diagonale ou Anti-diagonale

Bits: 1 0

On va parler de base Croisée

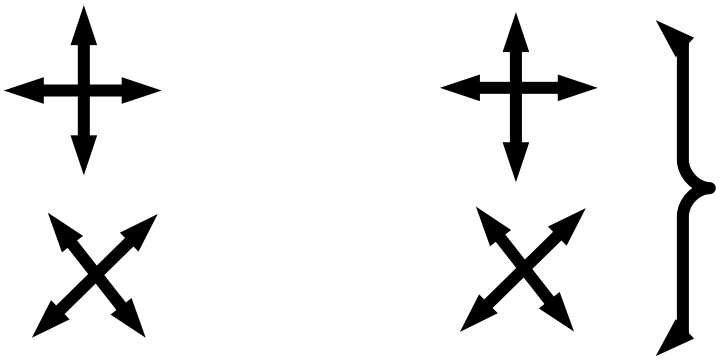
Les principes fondamentaux:
Si le photon est créé et lu dans la même base il sera lu correctement.
Si le photon est crée dans une base et lu dans l'autre il sera correctement lu que dans 50 % des cas.

Digression: Dans le monde du chiffrement on nomme **Alice** et **Bob** les émetteur/récepteur du message (Le pirate est Eve en anglais prononcer iv' . Nous l'appellerons **Yves**)

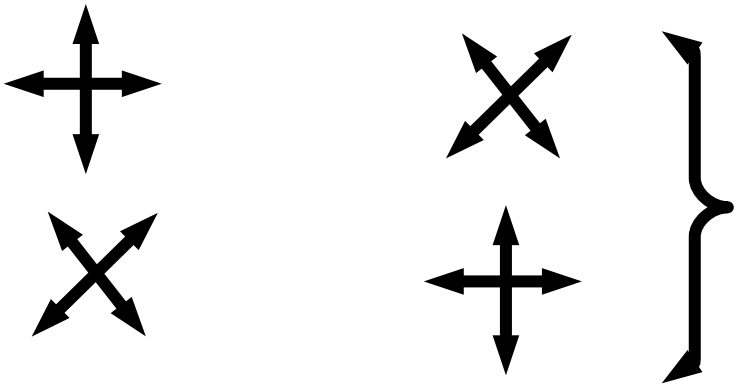
Au tableau:

Alice créé en

Bob lit en



==>> C'est déterministe c'est l'info



==>> C'est probabiliste à 50% c'est pas de l'info !

Notes:

Le choix des bases X et + est aléatoire chez Alice et Bob et indépendamment l'un de l'autre.
Le choix 1 ou 0 est aléatoire chez Alice.

Fonctionnement (avec le programme de simulation http://www.pixaline.net/intra/Atelier-2019/crypto/done/quantic/)

Alice envoie 1000 photons par exemple et sauve les valeurs des bits et des bases en interne.

Bob reçoit les photons et sauve les valeurs des bits et des bases en interne.

Alice envoie 10 % des bits choisis au hasard avec leur base de création sur canal classique pour tester.
(c'est 10% ne seront pas utilisés pour former la clé)

Bob ne va comparer les valeurs des bits créés et lus par une même base.

Si il n'y a pas avoir d'erreur alors la transmission n'a pas été piratée.

Les 10% sont jetés.

Les bases des autres photons sont échangées entre Alice et Bob. (Pas les valeurs des bits!)

Ainsi les 2 peuvent reconstituer la clé à partir des bits déterministes c-à-dire de base identique chez Alice et Bob.

Si il y a des erreurs (+ ou – 25 %):

Le processus est abandonné (Pas de clé , pas de transmission)

On ne connaît pas en détail ce qu'a fait Yves le pirate mais la seule chose qu'il pouvait faire est:

Choisir 1 base au hasard comme Bob. Sans connaître celle d'Alice. Et donc ajouter 50 % d'erreur sur les photons qui seront testés par Alice et Bob (rappel: de bases identiques pour ces 2 derniers).

Puis créer un nouveau photon dans la même base et de même valeur que celle qu'il a lu,
A noter qu'on ne peut pas cloner un photon. On ne peut que le détruire en le lisant puis en créer un autre. On ne peut pas anticiper la polarisation d'un photon.

Bob lira le photon de Yves avec une probabilité de 50% de rétablir la bonne valeur. Ce qui explique que la probabilité d'erreurs est de 25%, Dans les faits cela oscille entre 15 et 35 % dans ce programme de simulation.

Pour un rappel aller voir cette vidéo:
https://www.youtube.com/watch?v=zx9XkeX2YaY