Travaux Pratiques
Séance n° 1

Dans ce TP on va s'intéresser à l'arithmétique des corps finis binaires et à leurs applications en cryptographie symétrique.

Arithmétique modulaire dans $\mathbb{F}_2[X]$.

Pour manipuler des éléments de $\mathbb{F}_2[X]$ en SageMath, on peut commencer par créer l'anneau de polynômes `R` correspondant avec la commande `R.<X> = PolynomialRing(GF(2))`. Ici `GF(2)` désigne le corps $\mathbb{F}_2$ à deux éléments, et le `X` entre crochets désigne l'indéterminée (que l'on peut remplacer par un autre symbole). La somme, le produit s'exécutent avec `+` et `*`, le quotient et le reste d'une division euclidienne avec `//` et `%` (attention aux parenthèses !).

1. (a) Que fait la commande `R(n.bits())`, où `n` est un entier ?
   (On suppose que la commande `R.<X> = PolynomialRing(GF(2))` a déjà été exécutée.)
   (b) Que fait la commande `ZZ(P.list(),2)`, où `P` est un polynôme ?
   (c) Expliquer pourquoi, pour $d \in \mathbb{N}^*$, on peut identifier l'ensemble des polynômes de degré strictement inférieur à d avec $\{0,1\}^d$.
   (d) Si `m` et `n` sont deux entiers, comparer `R((m ^^ n).bits())` avec `R(m.bits())+R(n.bits())`. Que constate-t-on ?
2. Calculer à la main les opérations suivantes, puis vérifier le résultat avec SageMath :
   (a) $(1+X+X^3)(1+X^2+X^4)$
   (b) $(1+X+X^2)^3$
   (c) division euclidienne de X^5 par X^2+1
   (d) pgcd de $X^6+X^5+X^4+1$ et X^3+1
3. Calculer (pas à la main !) toutes les puissances successives de X modulo le polynôme X^4+X+1. Que peut-on constater ?
   Observe-t-on la même chose avec les puissances de X^3+X ?
4. À l'aide de la commande `xgcd`, déterminer deux polynômes U et V tels que $(X^3+X^2)U+(X^4+X+1)V=1$.
   Que vaut $(X^3+X^2)U$ modulo X^4+X+1 ? Retrouver ce résultat à l'aide de la question précédente.

Si $P \in \mathbb{F}_2[X]$ est un polynôme irréductible de degré d, alors tous les polynômes non nuls de degré strictement inférieur à d sont premiers avec P et vont donc posséder un inverse modulo P. On dit alors que $\mathbb{F}_2[X]/(P)$ est un corps, contenant 2^d éléments.

5. Déterminer les polynômes P vérifiant l'équation $(X^4+1)P+X^2 = X^3+X \bmod X^5+X^2+1$.

Un mode de chiffrement par blocs authentifié : le mode GCM

Lorsque l'on utilise un chiffrement symétrique (par blocs ou par flux), il est **indispensable** d'utiliser un mécanisme permettant de s'assurer qu'un message n'a pas été modifié lors de la transmission, ou

en tout cas de détecter une telle modification — ce qu'on appelle l'**intégrité** des données transmises, et qui est pertinent même en dehors du chiffrement. C'est d'autant plus vrai avec un chiffrement par flux, ou par blocs en mode CTR, à cause de l'attaque par changement de bits (*bit-flipping attack*).

L'expéditeur d'un message calcule pour cela un MAC (*message authentication code*), qui est une donnée courte (de la taille d'un bloc) mais qui dépend de l'ensemble du message et de la clef secrète ; il le transmet avec le chiffré. Le receveur, lui, vérifie la transmission en recalculant un MAC à partir du message reçu : si le résultat est différent du MAC transmis, c'est qu'il y a eu des modifications.

Pour qu'un tel système soit sûr, il faut qu'il soit très difficile, pour un adversaire ignorant la clef, de modifier un message et le MAC de façon concordante. Idéalement, cela doit être aussi difficile que de retrouver la clef.

Notons qu'en général un message contient des données en clair en plus des données chiffrées : par exemple des entêtes ou métadonnées qui peuvent être nécessaires à l'acheminement, ou des informations sur la version du protocole utilisée, etc. Pour la sécurité, l'intégrité de ces données doit aussi être assurée ; dans la description du MAC on parle de *données additionnelles à authentifier*.

Le mode GCM (*Galois Counter Mode*, où Galois fait référence au fait que l'on travaille dans un corps fini, aussi appelé corps de Galois) ajoute au mode CTR le calcul d'un MAC. Il est conçu pour une taille de bloc de 128 bits. Son usage avec AES, défini dans les RFC 5288 et 5289, a été introduit dans la version 1.2 du protocole TLS ; c'est l'un des trois seuls modes de chiffrement symétrique autorisés dans la version 1.3.

Le principe (légèrement simplifié) est le suivant (cf Fig. 1) :

- Tout bloc de 128 bits est identifié avec un polynôme de $\mathbb{F}_2[X]$ de degré strictement inférieur à 128. Ces polynômes sont manipulés modulo le polynôme irréductible $X^{128}+X^7+X^2+X+1$; on travaille donc en fait dans le corps $\mathbb{F}_2[X]/(X^{128}+X^7+X^2+X+1)$.
- On commence par calculer un élément secret de ce corps, qui est $H = E(k, 0\dots0)$ (chiffré du bloc clair contenant 128 zéros), vu comme un polynôme.
- Le message est chiffré de façon standard avec le mode CTR, sauf que le premier bloc, donné par $E(k, ctr_0)$ est mis de côté ; on a ensuite $c_i = m_i \oplus E(k, ctr_i)$ pour tout $1 \leq i \leq q$, où q est le nombre de blocs à chiffrer.
- On découpe les données additionnelles à authentifier en une série de blocs de 128 bits $a_1, \dots, a_p$, que l'on concatène avec les blocs chiffrés pour obtenir une suite

  $$(S_1, S_2, \dots, S_{p+q}) = (a_1, \dots, a_p, c_1, \dots, c_q).$$

  Chacun des blocs S_i est vu comme un élément de $\mathbb{F}_2[X]/(X^{128}+X^7+X^2+X+1)$.
- On calcule les termes d'une suite (T_k), à valeurs dans $\mathbb{F}_2[X]/(X^{128}+X^7+X^2+X+1)$, définie par :

  $$\begin{cases} T_0 = 0 \\ T_{k+1} = (T_k + S_{k+1}).H \mod X^{128}+X^7+X^2+X+1 \text{ pour } k \geq 0 \end{cases}$$

- Le MAC est $T_{p+q} \oplus E(k, ctr_0)$, vu comme un bloc de 128 bits.

6. Expliquer comment le receveur vérifie l'intégrité du message reçu.
7. Démontrer l'égalité

   $$MAC = S_1.H^{p+q} + S_2.H^{p+q-1} + \dots + S_{p+q-1}.H^2 + S_{p+q}.H + E(k, ctr_0) \mod X^{128}+X^7+X^2+X+1.$$

   Pourquoi parle-t-on de MAC polynomial ?

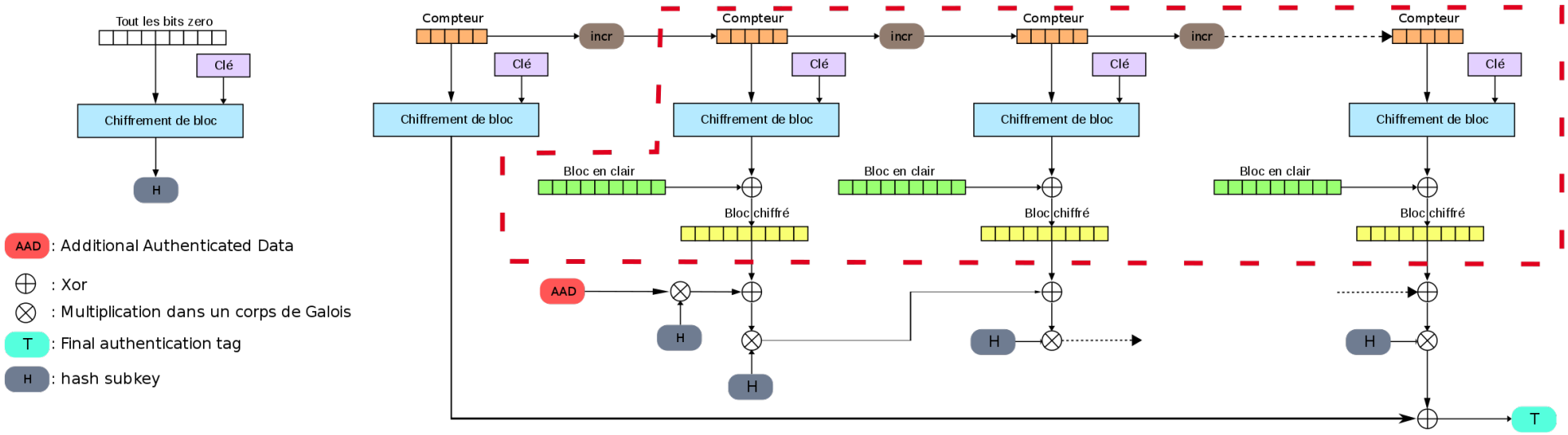

FIGURE 1 – Représentation schématique du mode GCM. La partie encadrée en rouge correspond au mode CTR classique.

8. Écrire une fonction `GalMAC(S,H,C0)` qui prend une liste de données à authentifier `S` et deux éléments `H` et `C0`, censés représentés $E(k, 0 \dots 0)$ et $E(k, ctr_0)$, et calcule le MAC correspondant.
   Les éléments de `S` ainsi que `H` et `C0` seront représentés par des entiers entre 0 et $2^{128} - 1$, à convertir en polynômes. On pourra afficher le résultat sous forme hexadécimale (méthode `hex()`).
9. Générer aléatoirement une liste de 20 données à authentifier, et deux éléments `H` et `C0`. On pourra utiliser la commande `ZZ.random_element(2^128)`.
   Tester la fonction `GalMAC` sur cet exemple.
10. Modifier un des bits d'une des données de `S` et comparer le nouveau MAC obtenu avec le précédent. Que constate-t-on ?
11. Attaques potentielles.
    (a) Montrer qu'un adversaire qui connaitrait la valeur de H est capable de modifier un message et son MAC de façon concordante (bit-flipping).
    (b) Montrer qu'un adversaire qui connaitrait la valeur de $E(k, ctr_0)$ peut essayer de retrouver H sans passer par une recherche exhaustive sur la clef.
    (c) On suppose qu'un protagoniste maladroit a chiffré deux messages différents avec le même vecteur d'initialisation (c'est-à-dire la même suite (ctr_i)). Expliquer comment un adversaire peut alors essayer de retrouver H (en plus des messages clairs).

Les calculs dans le corps $\mathbb{F}_2[X]/(X^{128} + X^7 + X^2 + X + 1)$ sont assez rapides, surtout depuis que les processeurs disposent d'instructions dédiés aux produits de polynômes à coefficients dans $\mathbb{F}_2$ (on parle aussi de multiplication sans retenue, *Carry-less Multiplication*, et d'instructions CLMUL).

Pour accélérer encore les calculs, on peut employer un compromis temps-mémoire. En effet, on multiplie toujours par le même élément H, y compris pour calculer les MAC de messages différents, tant qu'on utilise la même clef k. On peut donc précalculer, et tabuler, les valeurs de $X^i.H \mod X^{128} + X^7 + X^2 + X + 1$ pour $0 \leq i < 128$.

12. Créer une table `Tab` telle que pour tout i entre 0 et 127, la valeur de `Tab[i]` soit le polynôme $X^i.H \mod X^{128} + X^7 + X^2 + X + 1$ où H a été généré aléatoirement précédemment.
13. Expliquer comment l'utilisation de cette table permet de calculer $P.H \mod X^{128} + X^7 + X^2 + X + 1$ pour tout polynôme P (de degré < 128) avec uniquement des additions (c'est-à-dire des x-or). Implémenter cette méthode.
14. Écrire une nouvelle fonction `GalMAC2(S,Tab,C0)` qui utilise la table précalculée.
15. Pour encore plus d'efficacité, écrire une version qui se passe complètement de la représentation sous forme de polynômes et travaille uniquement avec les bits (commande `bits()`) et des x-or (commande `^^`).