Compte rendu du projet sur les polynômes de F2[X]

Objectifs :

L'objectif de ce projet est de réaliser une bibliothèque de fonctions en C permettant de manipuler les polynômes appartenant à F2[X].
Pour représenter un polynôme on utilise la représentation en binaire d'un entier de 64 bits non signé (avec les bits de poids fort correspondant au coefficient du monôme de plus haut degré). Avec cette représentation, des opérations élémentaires, comme l'addition (ou la soustraction) de deux polynômes, consistent en un simple XOR des bits des deux entiers représentant les polynômes.

Remarques :

- Les polynômes sont représentés par le type **uint64_t** et les constantes sont considérées comme des **int**, il faut donc spécifier que les constantes sont de type uint64_t quand on en a besoin. Par exemple pour mettre à 1 le coefficient 45 d'un polynôme nul on ajoute à ce polynôme : **(f2_poly_t)1 << 45**.

- Dans **arithm.c** on utilise la fonction **pp_diviseur_premier**(n) qui renvoie le plus petit nombre premier qui divise n. La fonction renvoie 1 si n = 1 et 0 si n = 0, si le dernier bit de n est nul alors 2 | n et donc la fonction renvoie 2. Ensuite on teste pour tout i = 3 + 2k, i < racine carré(n) + 1, k entier, si i divise n. On teste les i par ordre croissant, dès qu'un i divise n c'est le plus petit diviseur premier (on est sûr qu'il est premier, sinon on aurait renvoyé un de ses facteurs avant).

- Fonction **f2_poly_deg**(P) **:** Cette fonction bien que simple est utilisée dans presque toutes les autres fonctions il est donc important de trouver un algorithme rapide. J'utilise un algorithme de recherche par dichotomie pour trouver le degré, à chaque étape je découpe P en deux parties, puis je regarde si la partie de gauche est nulle. Si elle est non nulle j'augmente le degré. L'algorithme s'effectue en $\log_2(64)=6$ étapes. J'ai considéré le polynôme nul comme de degré 0 mais ça ne cause pas de problème car je traite à part le cas du polynôme nul lorsque c'est nécéssaire.

- Fonction **f2_poly_div :** On effectue une division euclidienne, la multiplication par un X^n revient à faire un simple décalage de n position et la soustraction un XOR.

- Pour **f2_poly_rem** et **f2_poly_gcd** on utilise la division euclidienne et l'algorithme de calcul du pgcd.

- Pour **f2_poly_times** on s'inspire de l'algorithme d'exponentiation rapide modulaire. On fait le calcul de A x B mod N, à chaque étape i on multiplie B par X modulo N puis si le coefficient du monôme de A de degré i vaut 1 on ajoute B au résultat.

- Pour **f2_poly_x2n** fait juste n-1 élévations au carré modulo N en utilisant f2_poly_times.

- Pour **f2_poly_parity** qui calcule le reste de la division d'un polynôme P par X+1, on fait juste un xor de tous les bits de P, en effet si on effectue la division euclidienne par X+1, à chaque étape on enlève un monôme (disons de degré n) du dividende P ainsi que le monôme suivant (de degré n-1). Donc si on note Ri le reste de la division euclidienne à l'étape i (R0 = P)
Soit d le degré de Ri,

Lionel Beltrando M2 MIC

Si Ri ne possède pas de monôme de degré d-1, alors Ri+1 est de degré d-1
Sinon Ri+1 est de degré au plus d-2.
Donc la division par X+1 consiste bien à faire un xor de tous les bits.
Pour faire un XOR de tous les bits on découpe à chaque étape P en deux parties et on affecte a P le résultat du XOR des deux parties. On a donc $\log_2(64)=6$ étapes.

- Pour **f2_poly_recip** on lit les bits de P dans un sens (en se bornant au degré de P) et on les réécrit dans l'autre sens. Je pense qu'il doit exister une façon plus optimisée pour inverser des bits.

- Pour **f2_poly_irred** on utilise un critère d'irréductibilité des polynômes à coefficients dans Fq :
Un polynôme P de degré n > 0 sur Fq est irréductible ssi :
  - $P | X^{q^n} - X$
  - P est premier avec les $X^{q^r} - X$ où $r=\frac{n}{d}$, d diviseur premier de n.
- Pour **f2_poly_xn** on fait une exponentiation rapide modulaire en utilisant f2_poly_times.

- Pour **f2_poly_primitive** on utilise le critère pour qu'un polynôme P de degré n > 0 soit primitif :
  - P est irréductible
  - P ne divise pas les $X^{\frac{q^n-1}{r}}$ où r est un diviseur premier de q^n-1

- Pour la fonction **f2_poly_irred_order**(P) on renvoie 0 si le polynôme P de degré n > 0 n'est pas irréductible, sinon on renvoie l'ordre multiplicatif d'une racine (X mod P). Dans **testAuto** on calcule 2^n-1 , si c'est égal à l'ordre multiplicatif c'est que le polynôme est primitif. J'ai testé **f2_poly_irred**, **f2_poly_prim** et **f2_poly_irred_order** sur les polynômes polAES, polA51a, polA51b et polA51c que vous nous avez fourni pour tester les programmes et je trouve des résultats cohérents. (Le résultat est dans testAuto).

- Pour les fonctions qui génèrent des polynômes aléatoires, on utilise **/dev/urandom** qui nous fournit une graine aléatoire que l'on va utiliser pour générer une suite pseudo aléatoire pour déterminer les coefficients d'un polynôme. Pour **f2_poly_random**(d) on utilise le résultat de f**2_poly_random_inf**(d) en imposant que le coefficient du monôme de degré d soit 1. Pour générer un polynôme irréductible (respectivement primitif) on génère un polynôme, on vérifie s'il est irréductible (respectivement primitif), s'il l'est on le renvoie, sinon on en génère un autre. On est assuré de tomber assez rapidement sur un polynôme irréductible (respectivement primitif) car il y en a beaucoup.

- Pour compter les polynômes irréductibles (respectivement primitifs) de degré n > 0, on génère tous les polynômes de degré n et on vérifie s'ils sont irréductibles (respectivement primitifs).

Utilisation des programmes :

Mon Makefile génère quatre exécutables :

- **testAuto** qui s'utilise sans argument et sert à vérifier 14 fonctions de la bibliothèque sur des exemples prédéfinis (comme le degré, l'affichage, l'irréductibilité, la multiplication de polynômes modulaire etc.) sur quelques exemples.

- **testRandom** qui prend en argument un degré n et permet d'utiliser les fonctions qui génèrent les polynômes aléatoires. L'exécutable génère un polynôme de degré inférieur ou égal à n, un polynôme de degré n, un polynôme irréductible de degré n et un polynôme primitif de degré n.

- **irred** qui renvoie le nombre de polynômes irréductibles pour chaque degré.
Si on utilise le programme sans argument, il renvoie le nombre de polynômes irréductibles du degré 1 à 15.
Si on entre un argument n le programme renvoie le nombre de polynômes irréductibles du degré n.
Si on entre deux arguments x et y, le programme renvoie le nombre de polynômes irréductibles du degré x à y.

- **prim** qui fonctionne comme irred mais renvoie le nombre de polynômes primitifs. On peut vérifier que le nombre de polynômes primitifs de degré n est bon grâce à la formule :

$$\text{Nombre de polynômes primitifs de degré } n = \frac{phi(2^n - 1)}{n}$$