Le chiffrement BGV

Passons maintenant à un exemple de fonction homomorphique complète : le chiffrement BGV.

Etant un chiffrement complet, BGV permet non seulement des additions, mais aussi des multiplications sur les données chiffrées, comme nous l'avons vu tout à l'heure. Il repose sur la difficulté d'un problème mathématique moderne appelé Learning With Errors (LWE) ou sa version à structure algébrique Ring-LWE. Sans rentrer davantage dans les détails, il fonctionne sur des polynômes cyclotomiques dans des anneaux de la forme :

$$\mathbb{Z}[X]/(\Phi_m(X))$$

LWE constitue un problème calculatoire supposé difficile et resistant aux attaques quantiques.

Nous allons aborder dans la sous-section qui suit les grandes étapes de son fonctionnement.

1. **Génération des clés**

On fixe des paramètres :

- **n :** degré du polynôme (doit être une puissance de 2 pour l'efficacité).
- **q :** un grand entier premier (modulo). Il faut qu'il soit grand[1], sinon on a des débordements et le résultat final est biaisé.
- **f(x) :** un polynôme cyclotomique, généralement de la forme $f : x \mapsto x^n + 1$.

Et on choisit :

- Une clé secrète s(x) $\in \mathbb{Z}_q$, petit polynôme aléatoire[2].
- Une clé publique pk=(a(x),b(x)) constituée de a(x)$\in$ R_q (aléatoire) et : b(x) = -a(x)*s(x) + e(x) mod(q)

où e(x) est un petit bruit (petit <=> |e(x)*r(x)|<=δ).

La clé publique est donc : pk=(a(x),b(x)), et la clé secrète est s(x).

2. **Encodage et chiffrement**

Pour chiffrer un message m(x)$\in$ R_q (avec petits coefficients) :

- On commence par encoder m avec un facteur d'échelle δ={pipe_bas}q/t {pipe_bas}tel qu'on ai m'(x)=m(x)δ. *L'objectif est que m'»e*r, pour assurer que le bruit ne perturbe pas le message lors du déchiffrement.
- On prend un petit vecteur aléatoire r(x)$\in$ R_q
- On calcule : c_0(x)=b(x)*r(x)+m'(x) mod(q) c_1(x)=a(x)*r(x) mod(q)

Le chiffré est donc : c(x)=(c_0(x),c_1(x))

3. **Calculs sur les messages chiffrés**

Additions : - (c_0,c_1) + (c_0',c_1') = (c_0 + c_0', c_1 + c_1')

Multiplications : - Le produit nécessite une étape supplémentaire (appelée relinearization) car le degré du chiffré augmente. La relinearisation utilise des clés de rélinearisation (générées à partir de la clé secrète) pour ramener le chiffré à deux composantes tout en préservant l'homomorphisme. Le schéma BGV applique un traitement pour revenir à une forme standard à 2 composantes. Concrètement : Quand on multiplie deux chiffrés : $c^{(1)=(c\_0}(1),c\_1^{(1)),c}(2)=(c\_0^{(2),c\_1}(2))$

On obtient : $c^{(1)c^{(2)=(c\_0}(1)c\_0}(2)$, $c\_0^{(1)c\_1^{(2)+c\_1}(1)c\_0}(2)$, c_1^(1)*c_1^(2))

[1]Pour une sécurité classique de 128 bits, les implémentations modernes recommandent pour BGV $q \approx 2^{50}$ à 2^{200}, selon la profondeur des circuits

[2]de degré $\leq$ n et de coefficients petits, généralement dans {-1;0;1}. On veut en effet garder le produit $s \times r$ peu bruyant

→ C'est un triplet, donc on sort du format à 2 composantes. → Il faut "relinéariser" pour revenir à un format à deux composantes.

4. **Déchiffrement**

- Le détenteur de la clé privée peut supprimer le bruit et extraire le message clair à partir du polynôme :

Si c(x)=(c0,c1), alors le message clair est obtenu par : m'(x)=c_0(x)+c_1(x)*s(x) mod(q) m(x) ={pipe_bas}m'(x)/δ [pipe]

Sous réserve que le bruit ne soit pas trop grand, ce message est exact. (Remarque : le bruit augmente à chaque opération homomorphe, surtout les multiplications. Le schéma BGV inclut donc des techniques comme le modulus switching pour maintenir le bruit dans des bornes correctes tout au long du calcul. Nous n'aborderons pas cela ici.)

Application du chiffrement BGV

Paramètre	Valeur	Justification
t	64	Message clair : on veut 4, 5, 9, 20 $\in$ [0, 63]
q	65537	Grand modulo premier, $q > \Delta^2$
Δ	{pipe_bas}q / t{pipe_bas} = 1024	Facteur d'échelle, assure que $m' \gg bruit$
s	1	Clé secrète simplifiée
a	1234	Échantillon aléatoire de $\mathbb{Z}_q$
e	1	Bruit minimal pour garantir exactitude
r	1	Aléa petit et constant pour simplicité

1. Génération de clés On calcule : b=-a*s+e=-1234*1+1=-1233 mod(65537)=64304 -> Clé publique : pk=(a(x)=12345,b(x)=20424) -> Clé secrète : s(x)=1
2. Chiffrement (on veut faire 4+5) Encodage avec δ=1024 : m_1=4*Rightarrow*m_1'=4*1024 = 4096 m_2=5Rightarrowm_2'=5*1024 = 5120

Chiffrement Rappel : c_0=b*r+m',c_1=a*r

Pour m_1 : c_0^(1) = 64304+4096 = 68400 mod(65537) = 2863 c_1^(1) = 1234

Pour m_2 : c_0^(2) = 64304+5120 = 69424 mod(65537) = 3887 c_1^(2)=1234

3. Addition : $(c\_0^{(1)+c\_0}(2), c\_1^{(1)+c\_1}(2))$=(2863+3887, 1234+1234)=(6750, 2468)

Déchiffrement : m'^+=c_0+c_1*s = 6750+2468 = 9218 mod(65537) m={pipe_bas}9218/1024[pipe]={pipe_bas}9.002[pipe]=9

4. Multiplication

Produit brut (avant relinearisation) : Formule du produit: c_0^× = c_0^(1)*c_0^(2) = 2863*3887 = 11128481 mod(65537) = 52728 c1^× = $c\_0^{(1)c\_1^{(2)+c\_1}(1)c\_0}(2)$ = 2863*1234 + 1234*3887 = 3532942 + 4796558 = 8329500 mod(65537) = 6301 c2^ ×= c_1^(1)*c_1^(2) = 1234*1234 = 1522756 mod(65537) = 15405

Triplet : (c_0,c_1,c_2)=(52728, 6301, 15405)

Relinearisation (simplifiée ici) : On simule que c_2*s est injecté dans c_0 c_0'=c_0+c_2*s = 52728+15405 = 68133 mod(65537) = 2596 c_1'= c_1 = 6301

5. Déchiffrement final

m'^× = c_0'+c_1'*s = 2596+6301=8897 mod(65537) = 8897

On utilise : $m = pipe_b as m'Ö\delta^2[pipe] = pipe_b as 8897/1024^2[pipe] = 0$

:(ça marche pas.... Pourquoi ?

Toute l'info a été perdue dans la réduction modulo q car le produit chiffré a dépassé q. Il faut m_1*m_2 δ ^2 < < q.

Trouver un q minimal, c'est le noise budget (ici on n'aborde pas ça).

m_1'*m_2'=4096*5120=20971520 plus grand que 65537

En refaisant avec q = 2^36 = 68719476736 :

b = -a*s+e = -1234*1+1 = -1233 mod(68719476736) = 68719475503 -> Clé publique : pk=(a(x)=12345,b(x)=68719475503) -> Clé secrète : s(x)=1

Rappel : c_0=b*r+m',c_1=a*r

Pour m_1 : c_0^(1) = 68719475503+4096 = 68719479599 mod(68719476736) = 2863 c_1^(1) = 1234

Pour m_2 : c_0^(2) = 68719475503+5120 = 68719480623 mod(68719476736) = 3887 c_1^(2)=1234

Et la multiplication

Produit brut (avant relinearisation) : Formule du produit: c_0^× = c_0^(1)*c_0^(2) = 2863*3887 = 11128481 mod(68719476736) = 11128481 c_1^× = c_0^(1)*c_1^(2)+c_1^(1)*c_0^(2) = 2863*1234 + 1234*3887 = 3532942 + 4796558 = 8329500 mod(68719476736) = 8329500 c_2^×= c_1^(1)*c_1^(2) = 1234*1234 = 1522756 mod(68719476736) = 1522756

Triplet : (c_0,c_1,c_2)=(11128481, 8329500, 1522756)

Relinearisation (simplifiée ici) : On simule que c_2*s est injecté dans c_0 c_0'=c_0+c_2*s = 11128481+1522756 = 12651237 mod(68719476736) = 12651237 c_1'= c_1 = 8329500

Déchiffrement final :

m'^× = c_0'+c_1'*s = 12651237+8329500 = 20980737 mod(68719476736) = 20980737

On utilise : $m = pipe_b asm'Ö\delta^2[pipe] = pipe_b as20980737/1024^2[pipe] = 20$

Ça fonctionne !

sources : https://www.youtube.com/@CipheredDuck