Uma breve introdução ao FHE e Bootstrapping

...

Aluna

Luiza Barros Reis Soezima

Orientação

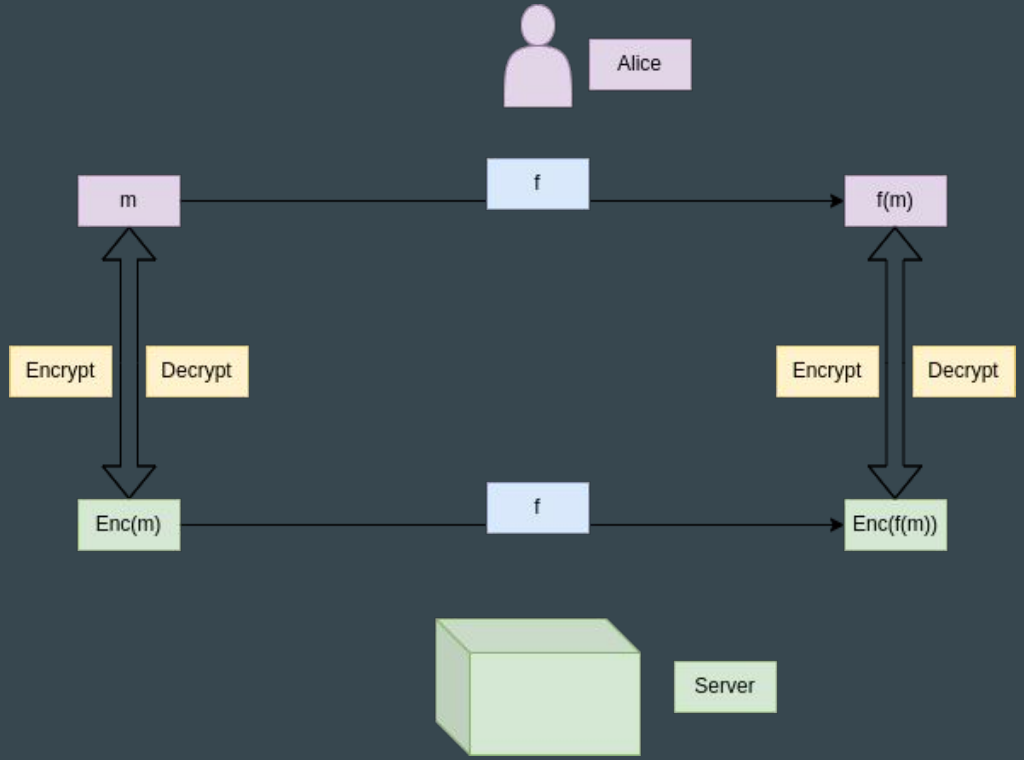
Hilder Vitor Lima Pereira

Conteúdo

I. Motivação
II. Homomorphic Encryption
III. Definições Importantes
IV. Bootstrapping
V. DGHV e GSW
VI. Conclusão

I. Motivação

A **criptografia homomórfica** é caracterizada por permitir a realização de operações matemáticas no dado encriptado sem que seja necessário descriptografá-lo para realizar as operações.



II. Homomorphic Encryption

Podemos definir FHE matematicamente com a seguinte propriedade:

Seja o texto cifrado **c** e o texto original **m** tal que:

$$\text{Enc}(\mathbf{m}) = \mathbf{c}$$

$$\text{Dec}(\mathbf{c}) = \mathbf{m}$$

onde os elementos **m** e **c** são elementos de um mesmo anel (vale as operações de multiplicação e adição), então:

$$\text{Dec}(\mathbf{c1} + \mathbf{c2}) = \mathbf{m1} + \mathbf{m2}$$

$$\text{Dec}(\mathbf{c1} \cdot \mathbf{c2}) = \mathbf{m1} \cdot \mathbf{m2}$$

Ou seja, temos que a descriptografia é homomórfica quanto às operações de soma e multiplicação.

Veja que isso significa que sempre que tivermos uma função **f** composta finitamente de operações de soma e multiplicação no anel, então:

$$\text{Dec}(\mathbf{f(c_1, \ldots, c_i)}) = \mathbf{f(m_1, \ldots, m_i)}$$

Assim, podemos definir uma segurança na qual o servidor computa **f(m1, . . . , mi)** a partir do texto cifrado do cliente **(c1, . . . , ci)** sem que ele (o servidor) aprenda nada sobre o texto pleno **(m1, . . . , mi)**.

Também é necessário que o tamanho do texto **(c1, . . . , ci)** seja limitado independentemente de **f**, de modo que as mensagens e textos cifrados sempre estejam no mesmo anel.

Do que foi apresentado por Craig Gentry em 2009, utiliza-se a encriptação de bit e aplica-se portas lógicas (AND, OR, NOT).

As cifras completamente homomórficas cifram a mensagem **m** na seguinte forma:

- combinam a chave secreta **sk** com algum valor aletório, obtendo **a • sk**;
- adiciona-se um ruído **r**, que é apenas um valor aletório pequeno, obtendo **a • sk + r**;
- finalmente, adiciona-se a mensagem, então o criptograma é da forma **a • sk + r + m**.

Veja, que a cada operação pode aumentar a quantidade de ruído r, assim, executa-se o bootstrapping para transformar uma cifra completamente homomórfica que trata o ruído crescente no criptograma.

III. Definições Importantes

O problema do divisor comum aproximado (ACD)

Nesse problema, recebe-se **i** múltiplos próximos **(m1, m2, . . . , mi)** de um inteiro positivo **p** desconhecido tal que vale o sistema:

$$
\begin{aligned}
m1 &= p \cdot q1 + r1,\ 0 \le r1 < p,\\
m2 &= p \cdot q2 + r2,\ 0 \le r2 < p,\\
&\vdots\\
mi &= p \cdot qi + ri,\ 0 \le ri < p,
\end{aligned}
$$

Então, queremos encontrar esse inteiro desconhecido **p**, sem o conhecimento de **qi** ou **ri.**

Lattice

A criptografia baseada em retículos permite baixo custo computacional ao realizar operações. Aqui dizemos que a dificuldade(hardness) do pior caso para resolver problemas de retículo significa que quebrar o esquema criptográfico é no mínimo tão difícil quanto quebrar muito problemas de retículos juntos no pior caso.

Um retículo L é uma estrutura algébrica definida como o conjunto da combinação linear de vetores ($\mathbf{b1, b2, \ldots, bn} \in \mathbb{R}m$) linearmente independentes com coeficientes em $\mathbb{z}$. Sendo $\mathbf{b1, b2, \ldots, bn}$ a base do retículo L:

$$L = \{a1b1 + a2b2 + \ldots + anbn : a1, a2, \ldots, an \in \mathbb{z}\}$$

LWE

Em linhas gerais, o problema do LWE seria a dificuldade de resolver sistemas de equações com erros. O LWE é a recuperação da chave secreta **s** dadas várias equações lineares aproximadas sobre o elemento **s.**

Aqui, sabemos que $\mathbf{b} \approx \mathbf{A} \cdot \mathbf{s}$, ou seja, $\mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e}$, onde **e** é um inteiro pequeno. E do artigo de Regev, temos que resolver LWE para um retículo de tamanho n equivale a resolver outro sistema equivalente de tamanho $\sqrt{n}$. Assim, resolver esse sistema seria exponencial no tamanho do retículo.

Esses ruídos e são ruídos gaussianos discretos(inteiros) e seguem uma distribuição normal(gaussiana), assim, os ruídos **e** são pequenos e próximos da média.

Search-LWE

Nesse problema queremos recuperar o segredo **s** dadas uma sequência aleatória de equações lineares aproximadas. Assim, o erro é um erro pequeno definido por uma distribuição normal.

Decision-LWE

Nesse problema queremos determinar para as m amostras de equações lineares (ai,bi) com distribuição normal, o que é de fato dado aleatório e o que é amostra com ruído.

RLWE

Aqui, as operações do LWE são definidas no anel R como forma a reduzir a expansão e fazer com que n bits sejam encriptados, em vez de 1 apenas. A partir do momento que $a \cdot s$ é um polinômio, pode-se guardar mais informações

IV. Bootstrapping

A ideia de Gentry...

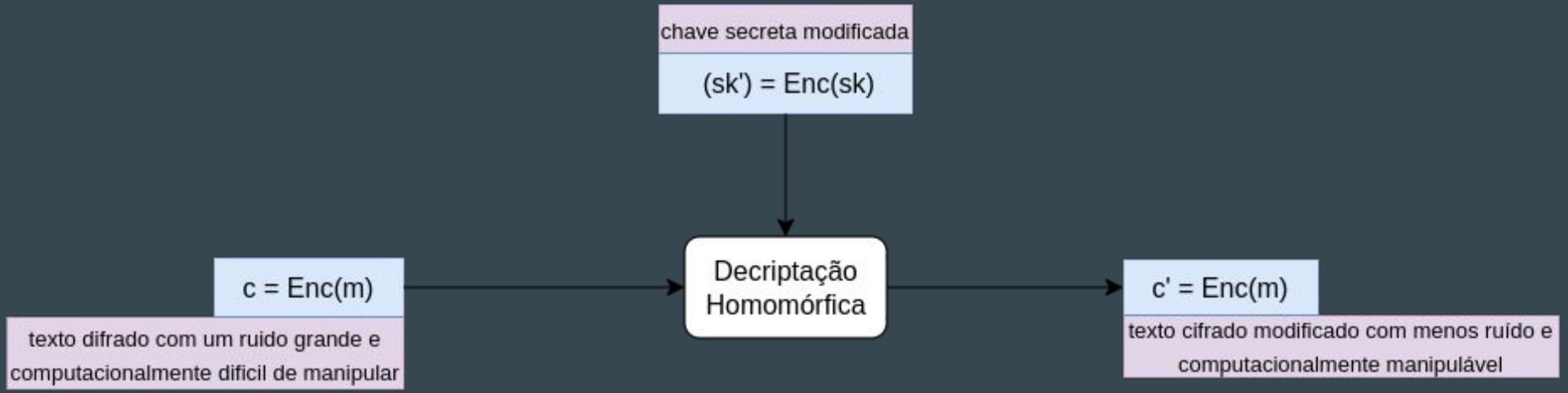
Se por um lado temos um esquema SHE (Somewhat Homomorphic Encryption), no qual aplicam-se as propriedade homomórficas, mas em quantidade limitada (somas e multiplicações limitadas), onde o ruído **r** que contribui para o texto cifrado **c** cresce de forma que se torna **impossível** realizar a operação de decifração, a propriedade de compactação do texto cifrado não é mais válida.

Na ideia de Gentry, para tratar esse problema, inclui-se uma encriptação da chave privada como parte da chave pública. Assim, quando o texto cifrado **c** fica grande e com muito ruído, o encriptador aplica o esquema SHE para avaliar a função de descriptografia usando a chave privada previamente encriptada. Esse processo de re-criptografar produz uma nova criptografia do texto original **m,** mais compacto e menos ruidoso. Ou seja, o ruído passa a ser limitado superiormente.

Da definição do dicionário, temos que bootstrap é colocar-se acima pelos seus próprios bootstrap. No contexto de HE, significa que o esquema avalia homomorficamente seu procedimento de descriptografia usando alguma operação (ou algumas operações) a mais.

Para que isso seja possível de fazer, o esquema SHE utilizado deve prover segurança circular, ou seja, o esquema deve conseguir criptografar sua própria chave secreta.

O esquema apresentado por Gentry, sugere um “squeeze” do circuito de decriptografia de modo que o esquema SHE seja avaliável de forma homomórfica.



Squashing

O processo de squashing ou “esmagamento” consiste na multiplicação do texto cifrado com elementos de um conjunto de vetores para reduzir o grau polinomial do circuito de descriptografia.

Bootstrapping

O processo de bootstrapping, em linhas gerais seria que dado um texto criptografado **c** com ruído **r** a saída do bootstrap é um texto cifrado **c’** com ruído **e’ < e.**

V. DGHV e GSW

DGHV : Uma cifra homomórfica simples baseada no problema ACD

Essa cifra propõe, inicialmente, um esquema de criptografia simétrica Todos os parâmetros são polinomiais em λ, e η é o tamanho da chave p. O gerador de chaves gera uma chave p com η-bit, que é um número primo. Usamos também um primo q aleatório grande e um número pequeno $r < p$.

Encrypt

A entrada é uma mensagem m, uma chave p, um primo q e um inteiro r.

A saída é o texto cifrado c:

$$c = Enc(m) = m + 2r + pq$$

Decrypt

A entrada é um texto cifrado c e uma chave p.

A saída é uma mensagem m:

$$m = Dec(c) = (c \bmod p) \bmod 2$$

Nessa cifra, a multiplicação aumenta muito mais o ruído do que a soma. Na ideia dada do circuito composto de portas lógicas, os criptogramas, a cada nível do circuito, possuem o ruído aumentado no fator $2^{(n\rho)}$. Assim, no pior caso, a profundidade multiplicativa do circuito corresponde ao log do nível do criptograma.

Logo, nesse esquema, sem o bootstrpping, cifrar mensagens leva tempo exponencial em λ.

Adicionando bootstrapping ao DGHV

Nessa modificação, uma parte do trabalho de deciframento foi transferida para a encriptação, que passa a encontrar valores **z** operados sobre soma. Isso é protegido pelo problema do SSP, que basicamente quer encontrar uma soma de S inteiros que gera um valor T. Esse é um problema np-difícil.

Diminuindo ruído...

Do que foi visto até agora, fica óbvio que as adições não afetam tanto o ruído, mas as multiplicações afetam muito o resultado. Idealmente, queremos reduzir o crescimento de **r** multiplicando eles por valores pequenos, o que afetaria menos o resultado.

Uma ideia é ao invés de multiplicar criptogramas **c** diretamente, aplica-se uma decomposição para algum dos criptogramas e calculamos o produto da decomposição com o criptograma, ou seja.

Para realizar a decomposição dos criptogramas como mencionado, relembramos o RLWE como sendo uma generalização do LWE. Definimos o conjunto de base B, o vetor $g = (B^0, B^1, \ldots, B^{(l-1)})$ e o vetor $g^{-1} = (a0, a1, \ldots, al-1)$ como decomposição na base B.

Assim, definimos a matriz G gadget de modo a aplicar $G^{-1}(c)$. Isso equivale a recalcular o criptograma c na multiplicação homomórfica, reduzindo o ruído.

GSW : cifra com ruído linear

Nesse método, aplica-se o método de aproximação do autovetor. Aqui, cada criptograma é uma amostras do RLWE e não apenas uma amostra. Veja que usa-se mais memória, mas o ruído é diminuído uma vez que é absorvido linearmente. Ou sejam o produto **c0 ⋅c1** gera um criptograma novo **c** tal que

$$\text{erro}(c) = O(\text{erro}(c0) + \text{erro}(c1))$$

O esquema...

- **GSW.ParamGen**(1 ^λ): Escolha um valor real σ > 0 e inteiros N,B e l, sendo N uma potência de dois. Defina q := B^l . Os parâmetros N,q e σ devem guarantir λ bits de segurança considerando o problema RLWE. Devolva params := (N,q,σ,B,l).
- **GSW.KeyGen**(params): Defina z(x) = ∑ ziXi com cada zi amostrado uniformemente de {−1,0,1}. Devolva sk := z.
- **GSW.Enc**(sk,m,params): Para cifrar um polinômio m ∈ R cujos coeficientes pertencem a ZB, amostre (ai ,bi) e defina C' com cada linha i igual a (ai ,bi). Note que C‘ é da forma [a,b] com b = az+e (mod q). Finalmente, devolva C = C‘ + m·G mod q.
- **GSW.Dec**(sk,C,params): Seja (a,b) ∈ R^2_q a última linha de C. Calcule u := b−a · sk ∈ Rq, interprete u como um polinômio em Z[X] e devolva round(B · u/q) mod B.

Veja que é possível mostrar as operações de soma e multiplicação do GSW:

- **GSW.Add**(C0,C1): simplesmente adicione as entradas correspondentes, i.e., devolva Cadd := C0 +C1 $\in$ R^(2lx2) q .
- **GSW.Mult**(C0,C1): decomponha C0 em base B e multiplique por C1 fazendo operações em Rq, i.e., devolva Cmult := G-^{1} (C0)·C1 $\in$ R^(2l×2) q .

Veja, que na demonstração da multiplicação, temos que o produto e0·e1 não aparece, na verdade temos:

$$\mathbf{e}\text{mult} := G^{-1}(C0)\cdot e1 + m1e0$$

Enfim, temos que o ruído é limitado tal que:

$$||ek'|| \leq \beta + 2kNBl\,\beta$$

Assim, se todos os criptogramas tiverem ruído menor que um valor β, o ruído aumenta em O(kβ)

O que é claro até agora

Temos algumas constatações claras até agora:

- No **GSW** cada criptograma tem tamanho essencialmente linear no parâmetro de segurança λ e quadrático no número de bits k.
- No **DGHV** manipulamos criptogramas com tamanho quadrático em k e cúbico no parâmetro de segurança λ.

Adicionando Bootstrapping...

Se por um lado a mensagem não crescesse a ponto de ser exponencial em k, podemos diminuir ainda mais o ruído do GSW. Da forma como é apresentado sem bootstrapping, temos que a mensagem tem um tamanho limitado.

Como o GSW possui limitação de profundidade de circuito, se aplicado o fato de que o ruído cresce lentamente e usando a porta NAND (NOT+AND), podemos avaliar circuitos com qualquer profundidade L.

Para transformar o GSW em completamente homomórfico, queremos que ele seja capaz de avaliar seu próprio circuito de decriptografia. Ou seja, basta identificar a profundidade L desse circuito e escolher parâmetros que permitam avaliar circuitos de profundidade L+k, para alguma constante pequena k que permita que outras operações úteis sejam feitas entre os bootstrappings

Uma alternativa super rápida...

Aplicamos o bootstrapping numa cifra homomórfica simples baseada no decisional-LWE, pois a amostra (a,b') é indistinguível de um vetor uniformemente aleatório, assim (a,b') + (0,μ) mod q é indistinguível de um vetor uniforme para qualquer μ.

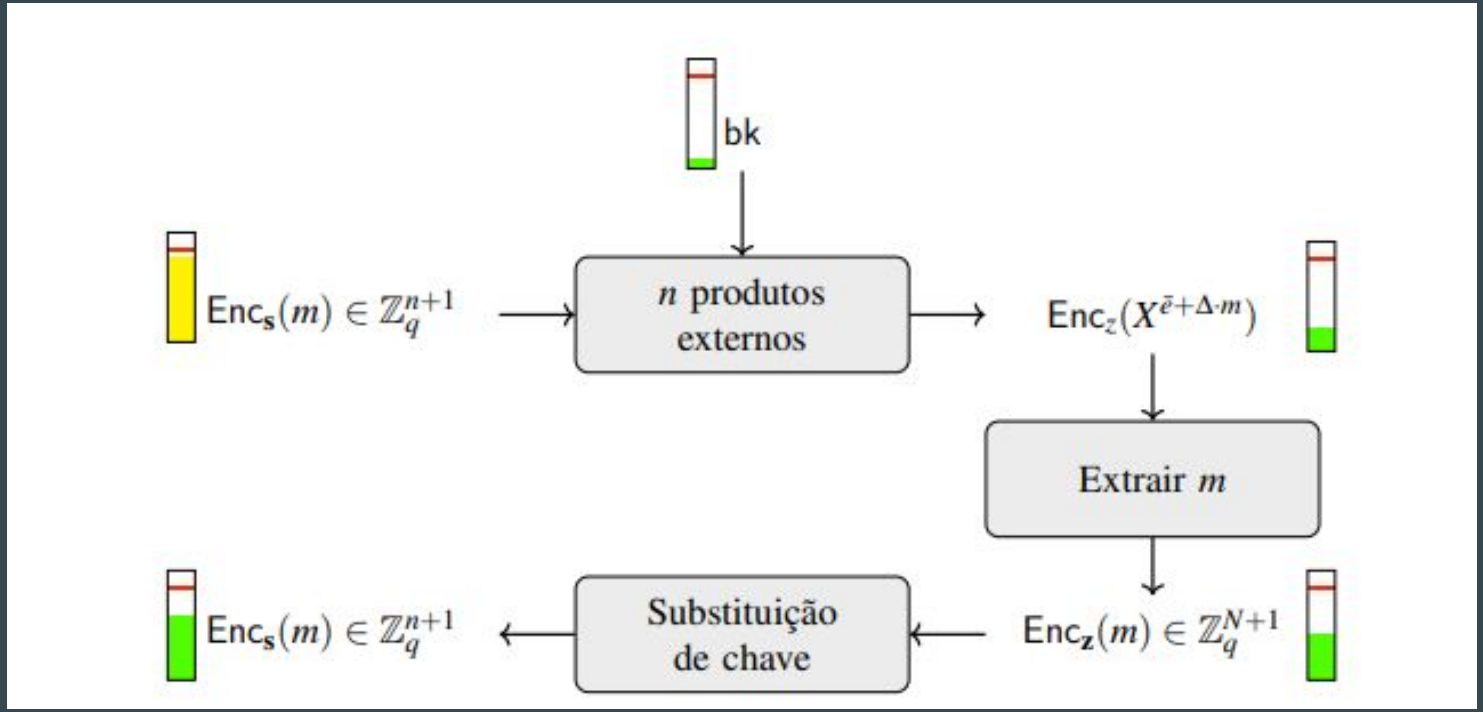
Veja que é preciso garantir que tanto os criptogramas devolvidos pela função Enc quanto o s obtidos após o bootstrapping sejam válidos, pois isso garantirá que qualquer função possa ser avaliada homomorficamente, já que, dado um criptograma válido, será possível aplicar a porta NAND e em seguida o bootstrapping, que produzirá novamente um criptograma válido.

Após a aplicação do NAND homomórfico, o objetivo do bootstrapping é decifrar **c** homomorficamente e obter um criptograma válido.

O primeiro passo do bootstrapping é calcular o produto escalar a ·s e para fazê-lo, temos criptogramas GSW cifrando cada coordenada si de s. Chamamos de chave do bootstrapping bk. esse produto escalar.

Da sugestão do TFHE, com apenas uma multiplicação, se supormos que a chave secreta s é binária, portanto, reduzimos o custo do bootstrapping para $\Theta(n)$ multiplicações homomórficas. Além disso aplica-se o produto externo homomórfico mais rápido em $\Theta(\log q)$.

O produto externo homomórfico é efetuada decompondo a amostra RLWE e multiplicando por C, ou seja, multiplicando um vetor por uma matriz, o que gera novamente uma amostra RLWE, ou seja, um vetor.



Aqui é a definição segundo o texto dado para a aplicação do bootsrapping com o GSW. Veja que o ruído é reduzido significativamente após o produto externo.

A chave de bootstrapping é um conjunto de criptogramas tidos a partir de GSW, em que cada um cifra uma entrada **si** da chave secreta **s** do esquema de base. Então, definimos:

$$bk := \{bki := \textbf{GSW.Enc}(si) : 0 \leq i < n\}.$$

A **primeira etapa** do bootstrapping recebe um criptograma do esquema de base, c = (a,b) e usa a chave bk calculada para calcular um criptograma RLWE. Com a nova chave c'=(a',b') e bk, queremos uma encriptação RLWE, aqui aplicamos cmux que é um encriptação GSW de X, com um ruído de **e** igual ao ruído de **e'.**

Após os produtos externos obtém-se um criptograma RLWE c da forma c = (a,b), agora, na **segunda etapa,** queremos transformar c em um criptograma c que cifra m e é baseado no problema LWE em vez do RLWE.

Em linhas gerais, isso consiste em aplicar uma função φ , onde o criptograma obtido cifra **m** sob a chave **z =** $\varphi(z)$.

Na **última etapa** aplica-se a substituição de chaves, onde transforma-se um criptograma c do esquema de base, no nível zero e com muito ruído, em um criptograma c' também baseado no problema LWE, no nível um e com pouco ruído.

Se c é cifrado sob a chave secreta s, mas c' usa a chave $\mathbf{z} = \varphi(z)$, , onde z é a chave secreta nova obtida pela cifra GSW. Queremos aplicada a porta NAND falada antes em c', aqui cria-se uma chave pública ksk que cifra z sob s que substitui a chave de c' por s.

Juntando as 3 etapas, observamos que o bootstrapping recebe como entrada a chave bk, que consiste em um conjunto de criptogramas com pouquíssimo ruído, então, cada uma das três etapas acumula ruído sobre bk de tamanho $O(\sigma)$.

Como os ruídos seguem uma distribuição gaussiana, o valor esperado para somas dessa forma é na verdade √ k · max(e1,..., ek). Com isso, é possível fazer uma análise de caso médio da seguinte forma:

- A magnitude esperada para o ruído obtido na primeira etapa é O(σ · sqrt(nN log(q)))
- Na segunda etapa, espera-se que a magnitude do ruído seja ∑ N i=1 O(σ · p nN log(q)) = O(σN · p nlog(q)).
- A terceira etapa então adiciona o termo O(β · √ N logq), onde β = O(1) é um limitante para o ruído de ksk. Então, com grande probabilidade, o ruído no final do bootstrapping é limitado por O(σN ·sqrt(nlog(q)) +sqrt(N logq)) = O(σN · sqrt(nlog(q)))

VI.Conclusão

Comparando...

O esquema **DGHV** apresenta um modelo com o ruído crescendo de forma exponencial e nada ótima para a vida real.

O esquema **DGHV com bootstrapping** apresenta uma modificação no tratamento da decifração que trata de um novo problema de somas de inteiros.

Comparando...

O esquema **GSW** apresenta um modelo com um ruído crescendo linearmente, mas com limitação no tamanho do circuito e portanto no tamanho da mensagem.

O esquema **GSW com bootstrapping** apresenta o **TFHE** como sendo uma forma mais eficiente de tratar o ruído de forma a executar o bootstrapping com nível de segurança a $\lambda = 128$ em cerca de 0.1 segundo.

Se considerarmos que os cálculos homomórficos serão normalmente executados em servidores “na nuvem”, que são muito mais potentes que computadores usais, e também que esses procedimentos são altamente paralelizáveis, essa estratégia de computação homomórfica feita com a avaliação de uma porta lógica seguida de um bootstrapping extremamente rápido se mostra muito promissora.

Referências

- Hilder V. L. Pereira, Eduardo Morais, Introdução à criptografia completamente homomórfica com implementação em Sage
- V. F. Rocha, Julio López, An Overview on Homomorphic Encryption Algorithms
- Alice Silverberg, Fully Homomorphic Encryption for Mathematicians
- Craig Gentry Amit Sahai, Brent Waters, Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based