Relatório sobre a implementação do bootstraping amortizado

Aluno: Gustavo Esteche Araujo
Orientador: Prof. Hilder Vitor Lima Pereira

1 Introdução

Aviso: Ao comparar com o relatório parcial, note que este relatório melhora partes presentes no anterior e adiciona os progressos feitos na pesquisa. Para ver apenas o que foi substancialmente melhorado ou adicionado, comece a leitura pela seção 4 sobre duais. O código foi suprimido em várias seções por razões de comprimento, cheque o github para mais detalhes.

Neste relatório está escrito o que foi produzido em relação a pesquisa sobre a implementação do bootstraping amortizado descrito em [2] e [3]. A parte inicial da pesquisa se resumiu a um estudo de nivelamento sobre álgebra abstrata, para melhor compreender os objetos matemáticos relacionados à pesquisa. A pesquisa prosseguiu com a implementação das operações essenciais para viabilizar o bootstrapping proposto, de modo que os algoritmos descritos nos artigos pudessem, por fim, ser integralmente implementados e, assim, ter sua viabilidade de utilização devidamente analisada. O relatório foi produzido compilando os resultados de cada seção individual. Para acessar o conteúdo escrito e acessar também as implementações e testes, segue o link para o github `https://github.com/gustavoesteche/ic-bootstraping`.

2 Traço

2.1 Descrição do problema

Seja $K \subseteq L$ um extensão de corpo finito e $Aut_K(L)$ o subconjunto de automorfismos de L que fixam elementos de K, ou seja, se $\sigma \in Aut_K(L)$ e $\alpha \in K$, logo $\sigma(\alpha) = \alpha$. Outra notação para σ utilizada sera $\sigma : L \hookrightarrow K$. Então o Traço de L em K de um elemento $a \in L$ é definido por

$$Tr_{L/K}(a) = \sum_{\sigma \in Aut_K(L)} \sigma(a) \tag{2.1}$$

Agora se torna necessário definir o tipo de anel e corpo utilizado no *framework* proposto. Sendo p um primo e $n \in \mathbb{Z}^+$, considere o seguinte anel quociente:

$$R_{p^n} = \frac{\mathbb{Z}[x]}{< \Phi_{p^n}(x) >} \tag{2.2}$$

Note que esse anel também pode ser escrito como $\mathbb{Z}(\zeta_{p^n})$, onde o isomorfismo é visto claramente como $\zeta_{p^n} = x$.

O objetivo inicial é computar o traço de um elemento a entre os anéis R_{p^n} e $R_{p^{n-1}}$, denotado como $Tr_{R_{p^n}/R_{p^{n-1}}}$ (fonte sobre o problema). Note que o problema foi definido sobre os corpos e não anéis, porém como o automorfismo respeita a estrutura do anel, o traço permanece bem definido. Como dito anteriormente, basta computar a soma de todos os automorfismos de R_{p^n} que fixam os elementos de $R_{p^{n-1}}$, então resolver o problema se resume a encontrar os automorfismos.

2.2 Solução teórica

Os automorfismos em anéis polinômiais são da forma $X \mapsto X^i$, então para encontrar os automorfismos basta encontrar os expoentes que satisfazem a propriedade de fixação requerida. Outro ponto importante a ser mencionado antes de se iniciar é que a quantidade de operações para encontrar todos os expoentes por força bruta ou por crivo (no caso dos expoentes serem inteiros coprimos), e depois aplicar o automorfismo de expoentes, é computacionalmente custoso, então uma determinação matemática é muito bem-vinda. Para facilitar determinação dos automorfismos, é interessante separar entre dois casos: quando o expoente é 1 e quando não é.

2.2.1 Computando o traço para $n = 1$

Esta discussão é mais simples, uma vez que computar o traço de $R_p/\mathbb{Z}$ é trivial, porque teremos apenas $\varphi(p) = p-1$ automorfismos, que serão definidos por todos os expoentes coprimos com p, ou seja $[1, p-1]$. Assim, considerando os automorfismos $\sigma_i(X) = X^i$, o traço é calculado por:

$$Tr_{R_p/\mathbb{Z}}(a) = \sum_{i=0}^{p-1} \sigma_i(a) \tag{2.3}$$

2.2.2 Computando o traço para $n > 1$

É fácil verificar que devemos ter $\varphi(p^n)/\varphi(p^{n-1}) = p$ automorfismos de um anel para o outro, e é claro que precisamos encontrar os automorfismos que isolam todos os expoentes múltiplos de p para que se movam para o corpo apropriado (ou, neste caso, anel). Assim, agora é necessário definir o grupo de automorfismos que possui esse efeito. Vamos provar que os p automorfismos para $n > 1$, na situação descrita são

$$\sigma_k(x) = x^{kp^{n-1}+1}, 0 \leq k \leq p-1 \tag{2.4}$$

Então, se $f(x) \in \frac{\mathbb{Z}[x]}{<\Phi_{p^n}(x)>}$, então temos que $f(x) = \sum_{i=0}^{\varphi(p^n)-1} a_i x^i$ que pode ser escrito como $f(x) = \sum_{i=0}^{\varphi(p^n)/p} \sum_{j=0}^{p-1} a_{ij} x^{pi+j}$, então

$$f(x) = \sum_{i=0}^{\varphi(p^n)/p} x^{pi} \sum_{j=0}^{p-1} a_{ij} x^j \tag{2.5}$$

Agora podemos aplicar os automorfismos ao polinômio, levando a:

$$\sigma_k(f(x)) = \sum_{i=0}^{\varphi(p^n)/p} x^{pi(kp^{n-1}+1)} \sum_{j=0}^{p-1} a_{ij} x^{jkp^{n-1}+j} \tag{2.6}$$

Observando o primeiro expoente, temos $kip^n + pi$, mas como a operação está no anel e $x^{kip^n} \equiv 1\, mod(\Phi_{p^n}(x))$, esse expoente torna-se pi. Assim, o automorfismo k torna-se:

$$\sigma_k(f(x)) = \sum_{i=0}^{\varphi(p^n)/p} x^{pi} \sum_{j=0}^{p-1} a_{ij} x^j x^{jkp^{n-1}} \tag{2.7}$$

Note que $R_{p^n} \simeq \mathbb{Z}(\zeta_{p^n})$ e $R_{p^{n-1}} \simeq \mathbb{Z}(\zeta_{p^{n-1}})$ e pela definição de raiz da unidade temos que $\zeta_{p^{n-1}} = \zeta_{p^n}^p$. Dessa forma, $\forall g(x) \in R_{p^{n-1}}$ temos:

$$g(x) = \sum_{i=0}^{\varphi(p^{n-1})} b_i \zeta_{p^{n-1}}^i \tag{2.8}$$

Substituindo pelas relações mencionadas,

$$g(x) = \sum_{i=0}^{\varphi(p^n)/p} b_i x^{pi} \tag{2.9}$$

Finalmente, utilizando a relação (2.7) temos que $\sigma_k(g(x)) = g(x)$, provando que os automorfismos definidos em (2.4) são os automorfismos que fixam R_{p^n} em $R_{p^{n-1}}$. Com os automorfismos devidamente definidos, procede-se ao cálculo do traço, realizando-se a soma sobre todos os p possíveis automorfismos.

$$\sum_{k=0}^{p-1} \sigma_k(f(x)) = \sum_{i=0}^{\varphi(p^n)/p} x^{pi} \sum_{j=0}^{p-1} a_{ij} x^j \sum_{k=0}^{p-1} x^{jkp^{n-1}} \tag{2.10}$$

Perceba que é uma boa ideia separar o caso $j = 0$ do restante, para selecionar apenas os expoentes múltiplos de p e também garantir que a última soma seja definida por uma série geométrica.

$$\sum_{k=0}^{p-1}\sigma_k(f(x)) = \sum_{i=0}^{\varphi(p^n)/p} x^{pi}\sum_{j=1}^{p-1} a_{ij}x^j \sum_{k=0}^{p-1} x^{jkp^{n-1}} + \sum_{i=0}^{\varphi(p^n)/p} x^{pi}a_{i0}\sum_{k=0}^{p-1} 1 \tag{2.11}$$

Agora, foquemos na soma da série geométrica, $\sum_{k=0}^{p-1} x^{jkp^{n-1}} = \frac{x^{jp^n}-1}{x^{jp^{n-1}}-1}$, mas note a seguinte propriedade dos polinômios ciclotômicos: $\prod_{d|n}\Phi_d(x) = x^n - 1$. Então,

$$\sum_{k=0}^{p-1} x^{jkp^{m-1}} = \frac{x^{jp^n}-1}{x^{jp^{n-1}}-1} = \frac{\prod_{d|jp^n}\Phi_d(x)}{\prod_{d|jp^{n-1}}\Phi_d(x)} \tag{2.12}$$

Como p^n divide jp^n e não divide jp^{n-1}, a soma $\sum_{k=0}^{p-1} x^{jkp^{n-1}}$ é um múltiplo de $\Phi_{p^n}(x)$, logo $\sum_{k=0}^{p-1} x^{jkp^{n-1}} = 0$ no anel $\frac{\mathbb{Z}[x]}{<\Phi_{p^n}(x)>}$. Assim, a soma dos automorfismos é:

$$\sum_{k=0}^{p-1}\sigma_k(f(x)) = \sum_{i=0}^{\varphi(p^n)/p} x^{pi}a_{i0}\sum_{k=0}^{p-1} 1 = p\sum_{i=0}^{\varphi(p^n)/p} x^{pi}a_{i0} \tag{2.13}$$

Se necessário podemos visualizar o polinômio resultante como um polinômio em R_{p^n} ou em $R_{p^{n-1}}$ com uma base diferente, no segundo caso basta apenas aplicar o mapeamento $X \mapsto Y^{1/p}$ ou $\zeta_{p^n}^p \mapsto \zeta_{p^{n-1}}$, e verificar o resultado no anel $R_{p^{n-1}}$. Uma propriedade interessante dos anéis escolhidos é a possibilidade de calcular o traço através de uma torre de traços. Ao invés de realizar traço direto de $R_{p^n}/\mathbb{Z}$, é possível realizar o traço entre $R_{p^n}/R_{p^{n-1}}/.../\mathbb{Z}$. Como o cálculo do traço foi discutido, vamos implementar o traço usando a definição da soma dos p automorfismos e testar se a implementação funciona conforme a solução matemática encontrada em (2.13).

2.3 Solução computacional usando SageMath

Primeiro, a implementação trivial para um caso geral R_m até $\mathbb{Z}$:

```
Zx = PolynomialRing(ZZ, 'x')

def trivial_trace_m_to_1(poly, m):
    '''Computa o traço do anel Z[x]/phi_m(x)
    para os inteiros, somando todos os
    automorfismos definidos pelos expoentes
    coprimos com m.'''
    f = cyclotomic_polynomial(m)
    coprimes = [x for x in range(1, m) if gcd(x, m) == 1]

    response = Zx(0)
    for i in coprimes:
        term = Zx(poly(x=x^i)) % f
        response = response + term
    return response
```

Somando todos os automorfismos de expoentes coprimos para computar o traço.

Agora, vamos à implementação da soma dos automorfismos e a abordagem otimizada, que leva em conta o resultado teórico.

```
def trace_pm_to_pm_1(p:int, m:int, poly):
    '''Computa o traço do anel Z[x]/phi_p^m(x) para
    o anel Z[x]/phi_p^(m-1)(x)'''
    f = Zx(cyclotomic_polynomial(p^m))
    response = Zx(0)

    if(m == 1):
        for i in range(1,p):
            response = response + Zx(poly(x=x^(i))) % f
    else:
        for i in range(p):
            response = response
                    + Zx(poly(x=x^(i*(p^(m-1)) + 1))) % f

        response = Zx(response(x=x^(1/p)))

    return response

def fast_trace_pm_to_pm_1(p, m, poly):
    '''Computa o traço de forma mais eficiente do anel
    Z[x]/phi_p^m(x) para o anel Z[x]/phi_p^(m-1)(x)'''
    f = Zx(cyclotomic_polynomial(p ** m) )

    response = Zx(0)
    if(m == 1):
        for i in range(1,p):
            response = response + Zx(poly(x=x^(i))) % f
    else:
        for i in range(0, poly.degree()+1, p):
            response = Zx(response + Zx(poly[i] * p * x^i)) % f

        response = Zx(response(x=x^(1/p)))
    return response
```

Como pode ser visto, ambos os códigos filtram os expoentes do polinômio que são coprimos a p e os multiplicam por p quando $m > 1$, além de computar a soma dos automorfismos (2.3) quando $m = 1$. Também é interessante mencionar o método de torre para o cálculo do traço de R_{p^m} para $\mathbb{Z}$, aqui está a implementação usando ambas as funções descritas anteriormente.

```
def trace_pm_to_1(p, m, poly):
    '''Computa o traço do anel Z[x]/phi_p^m(x) para
    os inteiros'''
    response = poly
    for i in range(m, 0, -1):
        response = trace_pm_to_pm_1(p, i, response)

    return response

def fast_trace_pm_to_1(p, m, poly):
    '''Computa o traço mais eficientemente do anel
    Z[x]/phi_p^m(x) para os inteiros'''
    response = poly
    for i in range(m, 0, -1):
        response = fast_trace_pm_to_pm_1(p, i, response)

    return response
```

Como é possível notar, ambas as abordagens diferem apenas por qual função é aplicada para realizar o traço entre R_{p^n} e $R_{p^{n-1}}$. Além disso, note que a abordagem de torre tem uma redução substancial de complexidade, para abordagem direta o traço de $R_{p^n}/\mathbb{Z}$ tem complexidade $O(\varphi(p^n)) \approx O(p^n)$ enquanto a abordagem em torre $O(\sum_{i=0}^{n} \frac{\varphi(p^i)}{\varphi(p^{i-1})}) = O(pn-1) \approx O(pn)$, diferença essa que pode ser observada na prática até com pequenos valores de n.

2.4 Traços $\mathrm{Tr}_{K/K_{13}}$ e $\mathrm{Tr}_{K/K_{12}}$

No contexto do artigo em que o *batch bootstrapping* é proposto [2], consideramos, para cada $i \in \{1,2,3\}$, um corpo $K_i = \frac{\mathbb{Q}[x]}{\langle \Phi_{p_i^{n_i}}(x) \rangle}$, em que p_i são primos distintos, ou seja, $p_1 \neq p_2 \neq p_3$, e $n_i \in \mathbb{Z}^+$. Considere por R_i o anel de inteiros do corpo K_i. No *framework* proposto a opera-se com elementos em $K = K_1 \otimes K_2 \otimes K_3$, e é necessário efetuar traços em $\mathrm{Tr}_{K/K_{12}}$ e $\mathrm{Tr}_{K/K_{13}}$, onde $K_{13} = K_1 \otimes K_3$ e $K_{12} = K_1 \otimes K_2$. Esta seção tem por objetivo desenvolver primeiramente uma abordagem tensorial para melhor compreensão dos objetos e depois tratar dos elementos diretamente em K.

2.4.1 Definições do Problema

Sejam os corpos K, K_1, K_2, K_3 e seus respectivos anéis de inteiros tais como definidos no parágrafo anterior. No *framework* proposto cada criptograma também carrega consigo um modo, e quando uma operação entre dois criptogramas é realizada temos um resultado que depende da aplicação de $Tr_{K/K_{12}}$ ou $Tr_{K/K_{13}}$. Para simplificação, trataremos na implementação o $mode = 2$ para o primeiro caso e $mode = 1$ para o segundo, respectivamente. Para resolver tal problema, vamos resolver o problema geral de computar o traço quando se *remove* apenas um corpo do produto tensorial de corpos.

2.4.2 Solução Teórica em Tensores

Seja $K = \bigotimes_l K_l$ e $K' = \bigotimes_{l,l\neq q} K_l$. O problema é calcular o traço entre K/K' do elemento $a = \bigotimes_l a_l \in K$. Usando a definição de traço, obtemos

$$Tr_{K/K'}(a) = \sum_i \sigma_i(a) = \sum_i \sigma_i^{(l)}(\bigotimes_l a_l) \tag{2.14}$$

Ao usar a propriedade dos *cannonical embeddings*, temos

$$\sum_i \sigma_i^{(l)}(\bigotimes_l a_l) = \sum_i \bigotimes_l \sigma_i^{(l)}(a_l) = \sum_i (\bigotimes_{l,l\neq q} \sigma_i^{(l)}(a_l) \otimes \sigma_i^{(q)}(a_q)) \tag{2.15}$$

Agora, usando a propriedade de produto misto e que um homomorfismo sempre mapeia as identidades, podemos ver isso como

$$\sum_i \sigma_i^{(l)}(\bigotimes_l a_l) = \sum_i((\bigotimes_{l,l\neq q} \sigma_i^{(l)}(a_l) \otimes 1_{K_q})\sigma_i^{(q)}(a_q)) \tag{2.16}$$

Como o produto tensorial dos *cannonical embeddings* $\bigotimes_l \sigma_i^{(l)}(a_l)$ deve fixar o elemento no corpo K' por definição, temos que $\bigotimes_{l,l\neq q} \sigma_i^{(l)}(a_l) \otimes 1_{K_q} = (\bigotimes_{l,l\neq q} a_l)$. Substituindo isso na equação:

$$Tr_{K/K'}(a) = (\bigotimes_{l,l\neq q} a_l)(\sum_i \sigma_i^{(q)}(a_q)) \tag{2.17}$$

Como $Tr_{K_q/\mathbb{Q}} = (\sum_i \sigma_i^{(q)}(a_q))$ pela definição de traço, obtemos a relação desejada de

$$Tr_{K/K'}(a) = (\bigotimes_{l,l\neq q} a_l)(Tr_{K_q/\mathbb{Q}}(a_q)) \tag{2.18}$$

Um breve comentário é necessário para justificar como a troca do corpo K pelo anel de inteiros correspondente R não interfere nas relações mostradas, a única propriedade faltante no anel de inteiros é a ausência de inversos multiplicativos para todo elemento não-nulo, propriedade que não é utilizada nas demonstrações.

2.4.3 Solução Teórica em Polinômios

Em vias de fato, os tensores são apenas uma representação mais amigável do que na realidade é representado por um polinômio $f(x) \in K$. No desenvolvimento a seguir vamos tratar novamente do caso onde queremos eliminar um corpo do traço, ou seja, fixar os elementos dos outros corpos. Tome os mesmos corpos definidos anteriormente. Como K é dado por um produto tensorial, para $f(x) \in K$ temos:

$$f(x) = \sum_i^{\phi(m1)-1} a_i x^{im_2m_3} \times \sum_j^{\phi(m2)-1} b_j x^{jm_1m_3} \times \sum_k^{\phi(m3)-1} c_i x^{km_1m_2}$$

onde cada termo estava originalmente em K_1, K_2, K_3 respectivamente. Assuma sem perda de generalidade que queremos fixar os elementos dos dois primeiros corpos. logo, o automorfismo a tem a forma:

$$x^{a(im_2m_3+jm_1m_3)} = x^{im_2m_3+jm_1m_3}$$

Então $am_2m_3 \equiv m_2m_3$ e $am_1m_3 \equiv m_1m_3$, para isso $a = k \times m_1m_2$. Porém, é importante que $am_1m_3 \not\equiv m_1m_3$, portanto temos que remover esses múltiplos apropriadamente. A solução pode ser vista na função *find_aut* da seção seguinte.

2.4.4 Implementação em SageMath

Existem duas funções, uma para um cenário mais geral e outra que já trata da avaliação do traço dependendo do modo, apresento apenas a mais geral.

```
def trace_Rr1_to_R(poly1, p1, m1, poly_m, index = 0):
    '''
    Computes Tr_{Rr1/R} based on the index
    '''
    trace = fast_trace_pm_to_1(p1, m1, poly1)

    trace_res = []
    for poly in poly_m:
        trace_res.append(poly)

    trace_res[index] = trace_res[index] * trace
    return trace_res
```

Para o caso mais relevante da aplicação do traço em polinômios, a implementação mais compreensível pode ser resumida a encontrar os automorfismos definidos e computar sua soma.

```
def find_aut(m, p, n):
    mi = p ** n
    fator = m/mi

    k = p-(fator % p).inverse_mod(p)
    rest = [i for i in range(p) if i!=k]
    p_power = mi//p
    aut = [(i*p + j)*fator+1 for i in range(p_power) for j in rest]
    return aut

def generic_trace(poly, m, p, n):
    m_i = p ** n
    factor = m / m_i
    f = Zx(cyclotomic_polynomial(m))
    auto = find_aut(m, p, n)
    ans = Zx(0)
    for i in auto:
        ans = ans + (Zx(poly(x=x^i)) % f)
    return ans
```

Além disso, no github há vários testes feitos para as implementações, onde utiliza-se a propriedade de "torre"dos traços":

$$Tr_{K/\mathbb{Q}}(a) = Tr_{K'/\mathbb{Q}}(Tr_{K/K'}(a)) \tag{2.19}$$

É possível calcular cada etapa da torre e comparar entre a abordagem tensorial e a polinomial e verificar sua igualdade. No final da torre ainda é possível verificar sua igualdade com a alternativa geral, dando mais robustez a implementação.

3 Composição de Anéis

3.1 Objetivo

A ideia principal é entender como representar elementos em um anel maior usando de elementos em anéis menores para computação mais rápida, e então implementar/testar algumas propriedades derivadas.

3.2 Ferramentas Matemáticas

3.2.1 Definições

Para o nosso problema, considere um inteiro m, então será analisado como compor o anel quociente $R_m = \mathbb{Q}[X]/ < \Phi_m(X) >$ utilizando anéis quocientes menores. Observe que este anel quociente também é um corpo, já que $\mathbb{Q}$ é um corpo, e por definição toda raiz de $\Phi_m(x)$ está em $\mathbb{C}$, então $\Phi_m(x)$ é um polinômio irredutível sobre $\mathbb{Q}$, logo $R_m = \mathbb{Q}[X]/ < \Phi_m(X) >$ é um corpo. Outra definição útil é: seja K_m a extensão de corpo tal que $K_m = \mathbb{Q}(\zeta_m)$. É trivial ver que $R_m \cong K_m$, pelo isomorfismo $X \mapsto \zeta_m$, portanto o desafio de decompor R_m é essencialmente o mesmo que decompor K_m.

3.2.2 Base de Potências

A base de potências ou base canônica dos polinômios é a base trivial de polinômios, podendo assim ser usada para definir os elementos no anel quociente $R_m = \mathbb{Q}[X]/ < \Phi_m(X) >$. Para qualquer inteiro m, a base pode ser escrita como $B = \{1, x, x^2, ..., x^{\phi(m)-1}\}$ e, é claro, cada elemento da base é linearmente independente e o envoltório linear da base é igual a todos os possíveis elementos em R_m. Como $R_m \cong \mathbb{Q}(\zeta_m)$, a base de potências também pode ser vista como $\{1, \zeta_m, \zeta_m^2, ..., \zeta_m^{\phi(m)-1}\}$, devido ao isomorfismo mencionado anteriormente.

3.2.3 Composição via produto tensorial e *Powerful basis*

Seja m um inteiro com fatoração em primos $m = \prod_l m_l$ onde cada m_l é uma potência de primo. Conforme descrito em [1] o corpo $K_m = \mathbb{Q}(\zeta_m)$ é isomorfo ao produto tensorial:

$$K_m \cong \bigotimes_l K_{m_l} \tag{3.1}$$

De forma equivalente, em termos dos anéis polinomiais é possível vê-lo como

$$K_m \cong \bigotimes_l \mathbb{Q}[X_l]/ < \Phi_{m_l}(X_l) > \tag{3.2}$$

Adotando a interpretação polinomial de K_m a partir da equação anterior, para fins de concretude, note que uma base natural sobre $\mathbb{Q}$ é o conjunto de multinômios $\prod_l X_l^{j_l}$ para cada escolha de $0 \leq j_l < \phi(m_l)$, e como $\phi(m) = \prod_l \phi(m_l)$ esta escolha pode ser feita. Este conjunto é chamado de *"powerful"basis* de K_m.

A *powerful basis* $\vec{p}$ de $K_m = \mathbb{Q}(\zeta_m)$ pode ser definida como o produto tensorial das bases de potências(ou *powerful*) $\vec{p_l}$ de cada K_{m_l}. Note que o conjunto de índices no tensor resultante do produto tensorial pode ser achatado, resultando em um tensor de *rank* 1, conforme visto em [1]. Entretanto, o que realmente importa no cenário proposto é como efetivamente realizar o produto tensorial entre os elementos dos anéis R_{m_l}.

Agora, para uma descrição mais concreta de como realizar a composição, o produto tensorial externo atuará como uma multiplicação polinomial e o isomorfismo que mapeia cada X_l em X será: $X_l \mapsto X^{m/m_l}$. Este isomorfismo pode parecer um tanto não natural, mas ao observar a raiz da unidade isso se torna mais natural, pois $\zeta_{m_l} = \zeta_m^{m/m_l}$. O mesmo procedimento pode ser feito para encontrar a mencionada *powerful basis* achatada. Por exemplo, para $m = 15$, $p_3 = \{1, x_3^1\}$ e $p_5 = \{1, x_5^1, x_5^2, x_5^3\}$, depois de realizar os cálculos, a base resultante será $p = \{1, x^3, x^5, x^6, x^8, x^9, x^{11}, x^{14}\}$ mais uma vez confirmada pelo artigo [1] após o isomorfismo $x \mapsto \zeta_m$.

3.2.4 Decomposição do Anel

Na seção anterior descrevemos como efetuar a composição de elementos dos corpos K_{m_l} em um elemento no corpo K_m. Portanto, seria natural tentar realizar o caminho inverso, encontrando uma forma explícita

de decompor um elemento em K_m nos seus correspondentes em cada K_{m_l}. Mas acontece que não existe uma operação de produto tensorial "inversa"trivial. Isso pode ser visualizado pelo seguinte: sejam A e B duas matrizes e ψ tal que $\psi = A \otimes B$. Usando notação da soma de índices:

$$\psi_{ijkl} = A_{ki} B_{jl} \tag{3.3}$$

Usando qualquer λ não-nulo no corpo no qual as operações estão definidas, temos $\psi_{ijkl} = (\lambda A_{ki})(\lambda^{-1} B_{jl})$, portanto não há unicidade da solução. Alguém poderia argumentar que ao operar no anel dos inteiros, apenas 1_R teria um inverso, levando a apenas três representações. Mas ao invés disso, podemos propor um problema clássico de álgebra linear: Note que, como os valores em ψ são arbitrários, temos $|\psi| = |A||B|$ elementos a serem representados, e apenas $|A| + |B|$ elementos de variáveis, então podemos até chegar a um caso degenerado, onde não há uma decomposição válida a ser proposta. Felizmente, este problema não parece ter aplicação no desenvolvimento do batch bootstrapping proposto, então por enquanto será ignorado.

3.3 Propriedades

A partir da composição por produto tensorial surgem algumas propriedades interessantes, como segue:

3.3.1 Propriedades de produto misto

Assuma que K, L são extensões de corpo de $\mathbb{Q}$, e que $a \in K$, $b \in L$. Então o produto tensorial de corpos $K \otimes L$ é definido como o conjunto de todas as combinações $\mathbb{Q}$-lineares de tensores puros $a \otimes b$, onde o produto tensorial é bilinear em relação aos racionais, essas operações concebidas no produto tensorial de corpos devem satisfazer a propriedade de produto misto, i.e.,

$$\begin{aligned} (a_1 \otimes b) + (a_2 \otimes b) &= (a_1 + a_2) \otimes b \\ (a \otimes b_1) + (a \otimes b_2) &= a \otimes (b_1 + b_2) \\ e(a \otimes b) &= (ea) \otimes b = a \otimes (eb) \\ (a_1 \otimes b_1)(a_2 \otimes b_2) &= (a_1 a_2) \otimes (b_1 b_2) \end{aligned} \tag{3.4}$$

Observe que, como estamos lidando com os produtos tensoriais externos da seção anterior, todas essas propriedades devem se aplicar, por exemplo quando $K = R_{m_1}$ e $L = R_{m_2}$.

3.3.2 Homomorfismos de anéis (embeddings)

Como visto em [1], ao tomar $K_m \cong \bigotimes_l K_{m_l}$ decorre diretamente das definições que σ é o produto tensorial dos *cannonical embeddings* $\sigma^{(l)}$ de K_{m_l}, i.e.,

$$\sigma(\otimes_l a_l) = \bigotimes_l \sigma^{(l)}(a_l) \tag{3.5}$$

Isto pode ser melhor explicado pelo seguinte raciocínio: o resultado de realizar os homomorfismos canônicos dos aneis e então fazer o produto tensorial deve ser o mesmo que fazer primeiro o produto tensorial e depois o homomorfismo de anel.

3.3.3 Cálculo do traço

Seja E uma extensão de corpo de K, e $a \in E$. Adicionalmente, $\forall i, 1 \leq i \leq [E : K]$, sejam σ_i os automorfismos de E que deixam os elementos em K fixos. A definição de traço usada é

$$Tr_{E/K}(a) = \sum_i \sigma_i(a) \tag{3.6}$$

Em outras palavras, o traço é a soma de todos os conjugados de a.

Agora, o objetivo será calcular o traço de um elemento em K_m para $\mathbb{Q}$. Usando o produto tensorial, temos:

$$Tr_{K_m/\mathbb{Q}}(\otimes_l a_l) = \sum_i \sigma_i(\otimes_l a_l) \tag{3.7}$$

Agora, aplicando a propriedade anterior [3.5],

$$\sum_i \sigma_i(\otimes_l a_l) = \sum_i \bigotimes_l \sigma_i^{(l)}(a_l) \tag{3.8}$$

Usando as propriedades da operação tensorial

$$\sum_i \bigotimes_l \sigma_i^{(l)}(a_l) = \prod_l (\sum_i \sigma_i^{(l)}(a_l)) \tag{3.9}$$

como $\sigma_i^{(l)}(a_l)$ representa o *cannonical embedding* do elemento a_l em K_{m_l} para $\mathbb{Q}$, a soma na equação representa o traço, ou seja,

$$\prod_l (\sum_i \sigma_i^{(l)}(a_l)) = \prod_l Tr_{K_{m_l}/\mathbb{Q}}(a_l) \tag{3.10}$$

Assim, finalmente alcançamos a relação desejada:

$$Tr_{K_m/\mathbb{Q}}(\otimes_l a_l) = \prod_l Tr_{K_{m_l}/\mathbb{Q}}(a_l) \tag{3.11}$$

3.4 Implementação e Testes

Segue a seguir, uma implementação da composição de anéis com o produto tensorial usando SageMath

```
R = PolynomialRing(ZZ, 'x')

# Compose the polynomial in the original polynomial ring,
# performing the isomorfism and the multiplication
def compose_poly(factors_m:list, poly_m:list, m:int):
'''
Computes the polynomial in the original field
factors_m: A tuple list representing the prime power that divides m
poly_m[i]: the polynomial in the field K_{p_i ^m_i} that decomposes
the original poly
'''
    poly = 1
    for i in range(len(poly_m)):
        poly = poly * poly_m[i](x^(m/(factors_m[i][0] ** factors_m[i][1])))

    return R(poly) % R(cyclotomic_polynomial(m))
```

Agora, usando a equação [3.11], aqui está uma implementação de como calcular o traço aproveitando a decomposição no SageMath

```
R = PolynomialRing(ZZ, 'x')
# Compute the trace using the decomposed polynomials
def compose_trace(factors_m:list, poly_m:list):
'''
Computes Tr_{K_m/Q}(a)
factors_m: A tuple list representing the prime power that divides m
poly_m[i]: the polynomial in the field K_{p_i ^m_i} that decomposes
the original poly
'''
trace = 1
  for i in range(len(factors_m)):
    trace = trace *
         fast_trace_pm_to_1(factors_m[i][0], factors_m[i][1], poly_m[i])
    return trace
```

Neste github há uma classe que testa as propriedades em [3.4] e a propriedade (3.11), usando SageMath. Todos os testes consistem em computar o lado esquerdo e o lado direito das equações e verificar que são iguais.

4 Duais

4.1 Introdução

No contexto da álgebra linear se temos um espaço vetorial V com uma base $B = \{b_i\}$ o espaço dual V^v é definido como o conjunto de todos os funcionais lineares em V. Um funcional linear é uma aplicação linear $f : V \to \mathbb{F}$, onde $\mathbb{F}$ é o corpo sobre o qual V está definido. O próprio espaço dual é um espaço vetorial e a base dual é definida de tal forma que $b_i^v(b_j) = \delta_{ij}$, o delta de Kronecker.

Agora, considerando o cenário em [2], seja $K = \mathbb{Q}(\zeta_m)$ onde $m \in \mathbb{Z}^+$. Para qualquer base $B = \{b_j\}$ de K sobre $\mathbb{Q}$, denotamos sua base dual por $B^v = \{b_j^v\}$, a qual é definida por:

$$Tr(b_i b_j^v) = \delta_{ij} \tag{4.1}$$

Assim, descrevendo $a \in K$ como $a = \sum_i a_i b_i$, tem-se o fato de que:

$$ab_j^v = \sum_i a_i b_i b_j^v \tag{4.2}$$

Aplicando o traço em ambos os lados,

$$Tr(ab_j^v) = Tr(\sum_i a_i b_i b_j^v) = \sum_i a_i Tr(b_i b_j^v)) \tag{4.3}$$

Agora, usando a definição $Tr(b_i b_j^v) = \delta_{ij}$,

$$a_j = Tr(ab_j^v) \tag{4.4}$$

Nos artigos [2] e [3], utiliza-se desses elementos da base dual de K para empacotar textos simples e criptogramas, urge então a necessidade de implementar o cálculo de uma base dual dado uma base bem definida.

4.2 Solução Teórica

Neste trabalho vamos utilizar a base de potências canônica, então o foco torna-se encontrar a base dual canônica de $\frac{\mathbb{Q}[X]}{<\Phi_p^n(x)>}$ onde p é primo e $n \in \mathbb{Z}^+$. Como a base canônica vai ser $[x^0, x^1, \ldots x^{\varphi(p^n)-1}]$, basta encontrar $\forall i, 0 < i < \varphi(p^n)$ um b_i^v que satisfaça (4.1).

Inicialmente note a seguinte propriedade sobre traços até os inteiros presente em [1]

$$\mathrm{Tr}(x^j) = \begin{cases} \varphi(p^n) & \text{se } j \equiv 0 \pmod{p^n} \text{ (I)} \\ -p^{n-1} & \text{se } j \equiv 0 \pmod{p^{n-1}} \text{ e } j \not\equiv 0 \pmod{p^n} \text{ (II)} \\ 0 & \text{caso contrário. (III)} \end{cases} \tag{4.5}$$

Seja então, um $k_i = x^{p^n - i} - x^{p^{n-1} - i}$, perceba que:

$$\mathrm{Tr}(x^i * k_i) = \mathrm{Tr}(x^{p^n} - x^{p^{n-1}}) = p^n \tag{4.6}$$

Logo, $\mathrm{Tr}(x^i * k_i)/p^n = 1$. Porém, também é necessário avaliar se nas outras potências o traço encontrado é 0, então

$$\mathrm{Tr}(x^j * k_i) = \mathrm{Tr}(x^{p^n - i + j}) - \mathrm{Tr}(x^{p^{n-1} - i + j}) \tag{4.7}$$

Vamos analisar como os 3 casos postos em (4.5) sobre $j - i$ impactam no resultado:

1. $j - i \equiv 0 \pmod{p^n}$, note que esse caso só ocorre quando $j = i$, que já foi analisado e tem o resultado desejado.

2. $j - i \equiv 0 \pmod{p^{n-1}}$ e $j - i \not\equiv 0 \pmod{p^n}$, pode ser dividido em dois casos

   2.1 $j - i + p^{n-1} \neq 0$
   Isso implica que ambos os traços satisfazem a condição (II) como são iguais, se cancelam resultando no resultado desejado, zero.

2.2 $j - i + p^{n-1} = 0$
Único caso problemático, visto que o segundo caso vai passar a satisfazer a condição (I) enquanto o primeira vai satisfazer a condição (II), então o valor do traço calculado ser $-p^n$.

3. caso contrário, ambos os traço vão ser 0 e a condição será satisfeita.

Note que o único caso problemático onde k_i não funciona, atinge os pares da forma $(i, i+p^{n-1})$. Uma solução trivial então é substituir $k_{i+p^{n-1}}$ por $k_{i+p^{n-1}} := k_{i+p^{n-1}} + k_i$. Vamos analisar o impacto dessa troca em todos os elementos x^j da base:

- $j \notin (i, i+p^{n-1})$ a corretude permanece já que, $\mathrm{Tr}(x^j(k_i + k_{i+p^{n-1}})) = 0$
- $j = i$, o traço entre $\mathrm{Tr}(x^i k_{i+p^{n-1}})$ se torna 0 como desejado
  $\mathrm{Tr}(x^i(k_i + k_{i+p^{n-1}})) = \mathrm{Tr}(x^i k_i) + \mathrm{Tr}(x^i k_{i+p^{n-1}}) = p^n - p^n = 0$
- $j = i + p^{n-1}$, o traço mantém o valor desejado já que:
  $\mathrm{Tr}(x^{i+p^{n-1}}(k_i + k_{i+p^{n-1}})) = \mathrm{Tr}(x^{i+p^{n-1}}(k_{i+p^{n-1}})) = 1$

Finalmente, basta utilizar o seguinte pseudocódigo para encontrar a base dual canônica:

Algorithm 1: `canon_dbasis(p, n)`

Input: Primo p, inteiro $n \geq 1$
Output: Base dual canônica dividida de p^n

1 $f \leftarrow$ polinômio ciclotômico $\Phi_{p^n}(x)$ em $\mathbb{Z}[x]$
2 $l \leftarrow$ vetor de tamanho $\varphi(p^n)$ em $\mathbb{Q}[x]$, entradas inicialmente 0
3 **for** $i \leftarrow 0$ **to** $p^{n-1} - 1$ **do**
4 $\quad k_i \leftarrow (x^{p^n - i} - x^{p^{n-1} - i}) \bmod f$ no anel $\mathbb{Z}[x]/(f)$
5 $\quad l[i] \leftarrow k_i$
6 **for** $i \leftarrow p^{n-1}$ **to** $\varphi(p^n) - 1$ **do**
7 $\quad l[i] \leftarrow (l[i] + l[i - p^{n-1}])/p^n$
8 **return** l

Pela linha 7 é possível inferir uma propriedade da base que será necessária para o futuro, as normas de cada termo $l[i], l[i - p^{n-1}]$ são $||p-1||$, portanto a norma infinita da base calculada é $2(p-1)/p^n$.

4.3 Implementação

Traduzindo o pseudocódigo para SageMath com algumas modificações, obtêm-se:

```
def canon_dbasis(p, m):
    f = Zx(cyclotomic_polynomial(p**m))
    l = vector(Qx, [0]*euler_phi(p**m))
    for i in range(p**(m-1)):
        p_magic = (p**(m-1) - i) % (p**m)
        fj = (Zx(x**(p**m - i) - x**p_magic) % f)
        l[i] = fj

    for i in range(p**(m-1), euler_phi(p**m)):
        p_magic = (p**(m-1) - i) % (p**m)
        fj = (Zx(x**(p**m - i) - x**p_magic) % f)
        fj = fj + l[i - p**(m-1)]
        l[i] = fj

    return l/(p**m)
```

Para ver mais da implementação e testes, verifique aqui.

5 RLWE e RGSW

Nesta seção os parâmetros para os esquemas serão apresentados tais como descritos em [2] com os detalhes específicos utilizados na implementação.

5.1 LWE

Uma breve introdução ao problema Learning With Errors(LWE) a base da segurança dos esquemas utilizados:

Definição 5.1. Considere inteiros $n, q \in \mathbb{N}^*$ e um parâmetro real $\sigma > 0$. Seja $\mathbf{s} \in \mathbb{Z}_q^n$ um vetor secreto fixo. A distribuição $\mathcal{A}_{\mathbf{s},\chi_\sigma}$ é definida pelo seguinte procedimento:

- amostrar $\mathbf{a} \leftarrow \mathbb{Z}_q^n$ uniformemente;
- amostrar o erro $e \leftarrow \chi_\sigma$, onde χ_σ é uma distribuição sub-Gaussiana centrada com parâmetro σ;
- computar $b := \langle \mathbf{a}, \mathbf{s} \rangle + e \mod q$;
- devolver o par $(\mathbf{a}, b)$.

Definição 5.2. (**LWE**) Dado um vetor secreto $\mathbf{s} \in \mathbb{Z}_q^n$, o problema *Learning With Errors* (LWE) com parâmetros (n, q, σ) consiste em, dado acesso a um número polinomial de amostras independentes extraídas da distribuição $\mathcal{A}_{\mathbf{s},\chi_\sigma}$, recuperar o vetor $\mathbf{s}$.

Mais precisamente, o problema decisório do LWE consiste em distinguir, com vantagem não negligenciável, entre amostras da distribuição uniforme $\mathcal{U}(\mathbb{Z}_q^n \times \mathbb{Z}_q)$ e amostras da distribuição $\mathcal{A}_{\mathbf{s},\chi_\sigma}$. É sabido que as duas versão são equivalentes (uma reduz para outra em tempo polinomial).

Ainda acredita-se que este problema é díficil [4], portanto faz sentido basear esquemas nisso. Porém, provas e discussões sobre sua dificuldade e definições de parâmetros se estendem juntamente a padronização do NIST (por exemplo [5]).

5.2 Funções Auxiliares

Para amostrar o ruído utilizado nos esquemas propostos utiliza-se uma distribuição gaussiana inteira centrada no 0, porém isso acarreta em um problema na implementação, -1 módulo Q é $Q-1$, o que estouraria o limite de corretude da decifração. Portanto, basta implementar uma função que lide com esse problema modular:

```
def sym_mod(a, q):
    a = ZZ(a) % q
    if 2*a > q:
        return a - q
    return ZZ(a)
```

Neste ponto se torna necessário a introduzir a operações de decomposição. Tome o vetor $g^\top = (1, 2, \ldots, 2^{\ell-1})$ e a matriz definida por $G = g^\top \otimes I_2$.

Lema 5.3 (lema 2.3 [2])**.** *Dado um inteiro q, seja $\ell = \lceil \log q \rceil$, $\mathbf{g}^\top = (1, 2, \ldots, 2^{\ell-1})$ e uma base fixa $\mathbb{Z}_q$ de $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ de $\mathcal{O}_K/q\mathcal{O}_K$. Então, existe uma função aleatória e eficientemente computável*

$$\mathbf{g}^{-1} : \mathcal{O}_K/q\mathcal{O}_K \to \mathcal{O}_K^\ell$$

tal que a saída da função, $\mathbf{x} \leftarrow \mathbf{g}^{-1}(a)$, sempre satisfaz $\langle \mathbf{g}, \mathbf{x} \rangle = a \bmod q$.

Mais especificamente, se $a = a_1\mathbf{b}_1 + \cdots + a_n\mathbf{b}_n$ onde $a_i \in \mathbb{Z}_q$ e $\mathbf{x}_i \leftarrow \mathbf{g}^{-1}(a_i)$ (onde a função $\mathbf{g}^{-1}(\cdot)$ é definida no Lema 2.1), então:

$$\mathbf{x} = \mathbf{x}_1\mathbf{b}_1 + \cdots + \mathbf{x}_n\mathbf{b}_n$$

e cada vetor $\mathbf{x}_i \in \mathbb{Z}_q^\ell$ é sub-Gaussiano com parâmetro $O(1)$.

Basicamente, se $x \in \mathcal{R}_Q$ então $g^{-1}(x)$ representa uma decomposição desse polinômio na base 2, na implementação feita é bem simples:

```
def inv_g_ZZ(x, B, Q):
    x = sym_mod(ZZ(x), Q)
    l = ceil(log(Q,B))
    return vector(x.digits(base=B, padto=l))

def inv_g_poly(x, B, Q, n):
    l = ceil(log(Q,B))
    result = vector(Zx, [0]*l)
    pow_x = Zx(1)
    for i in range(euler_phi(n)):
        result += pow_x * inv_g_ZZ(x[i], B, Q)
        pow_x *= Zx(x)
    return result
```

Perceba que é possível definir a mesma função para G, G^{-1}. A mesma se baseia na aplicação da anterior nas devidas linhas tal que $\langle \mathbf{G}, \mathbf{G^{-1}}(\mathbf{x}) \rangle = x$, sua implementação está no repositório.

5.3 Esquemas

Abaixo estão presentes os parâmetros utilizados nos esquemas:

- λ: o parâmetro de segurança
- $\mathcal{R}$: o anel ciclotômico $\frac{\mathbb{Z}[x]}{<\Phi_m(x)>}$
- Q: o módulo inteiro onde as cifras operam
- $\mathcal{R}_Q$: o anel quociente $\mathcal{R}/Q\mathcal{R}$
- $\mathcal{D}_\sigma$: Distribuição gaussiana discreta de ruído sobre $\mathcal{R}$ com desvio padrão σ.
- $l = \log_t(Q)$

5.3.1 Esquema RLWE

Considere também como parâmetros da cifra como o módulo do texto puro o inteiro t, o módulo inteiro do criptograma Q, o desvio padrão σ da distribuição gaussiana e o inteiro m representando o m-ésimo polinômio ciclotômico.

- **KeyGen**(1^λ): Amostre uma chave secreta $s \leftarrow \mathcal{R}_Q$
- **Enc**(sk, $\mu \in \mathcal{R}_t$): Amostre um elemeto uniforme do anel $a \leftarrow \mathcal{R}_q$ e um erro $e \leftarrow \mathcal{D}_\sigma$. Retorne um criptograma definido por
$$c := (a, sa + \left\lfloor \frac{Q}{t} \right\rceil \mu + e)$$
- **Dec**(c, sk): O algoritmo retorna um elemento $\mu \in \mathcal{R}_t$ definido por:
$$\mu = \lfloor t \langle (1, -s), c \rangle / Q \rceil \pmod t$$

O ruído de um criptograma c que ecriptma μ sobre a chave secreta s pode ser definido por

$$Err_\mu(c) = \langle (-s, 1), c \rangle - \left\lfloor \frac{Q}{t} \right\rceil \mu$$

5.3.2 Esquema RGSW

- **KeyGen**(1^λ): Amostre uma chave secreta $s \leftarrow \mathcal{R}_Q$.
- **Enc**(sk, $\mu \in \mathcal{R}$): Amostre um vetor uniformemente $\mathbf{a} \leftarrow \mathcal{R}_Q^{2\ell}$ e um vetor de ruído $e \leftarrow D_\sigma^{2\ell}$. Retorne um criptograma definido por $C := \begin{pmatrix} s\mathbf{a}^\top + \mathbf{e}^\top \\ \mathbf{a}^\top \end{pmatrix} + \mu\mathbf{G} \in R_Q^{2\times 2\ell}$
- **Dec**(c, sk): O algortimo retorna um elemento $\mu \in \mathcal{R}/2\mathcal{R}$:
$$\mu = \left\lfloor \left\langle (1, -s)_{,(2\ell-1)}\, c \right\rangle / Q \right\rceil \pmod 2$$

O ruído de um criptograma C que encripta a mensagem μ sobre a chave secreta $sk = (1, -s)^T$ pode ser definido por $Err_\mu(C) = e^T = sk^T \cdot C - \mu \cdot sk^T \cdot G$

5.4 Operações

- **Soma Homomórfica** $C_1 \boxplus C_2$: Recebe dois criptogramas RGSW C_1, C_2 cifrados sobre a mesma chave e retorna $C_1 \boxplus C_2 := C_1 + C_2$. O novo criptograma obtido cifra a soma das mensagens (μ_1, μ_2):

$$C_1 + C_2 = \begin{pmatrix} s(\mathbf{a_1}^\top + \mathbf{a_2}^\top) + \mathbf{E}^\top \\ \mathbf{a_1}^\top + \mathbf{a_2}^\top \end{pmatrix} + (\mu_1 + \mu_2)\mathbf{G}$$

- **Multiplicação Homomórfica** $C_1 \boxdot C_2$: Recebe dois criptogramas RGSW C_1, C_2 cifrados sobre a mesma chave e retorna $C_1 \boxdot C_2 := C_1\mathbf{G^{-1}}(C_2)$. O novo criptograma obtido cifra a multiplicação das mensagens (μ_1, μ_2):

$$C_1\mathbf{G^{-1}}(C_2) = \begin{pmatrix} s\mathbf{a_1}^\top + \mathbf{e_1}^\top \\ \mathbf{a_1}^\top \end{pmatrix} \mathbf{G^{-1}}(C_2) + (\mu_1) \begin{pmatrix} s\mathbf{a_2}^\top + \mathbf{e_2}^\top \\ \mathbf{a_2}^\top \end{pmatrix} + \mu_1\mu_2\mathbf{G}$$

$$= \begin{pmatrix} s(\mathbf{a_1}^\top\mathbf{G^{-1}}(C_2) + \mu_1\mathbf{a_2}^\top) + \mathbf{e_1}^\top\mathbf{G^{-1}}(C_2) + \mu_1\mathbf{e_2}^\top \\ \mathbf{a_1}^\top + \mu_1\mathbf{a_2}^\top \end{pmatrix} + \mu_1\mu_2\mathbf{G}$$

- **Produto Externo** $C_1 \boxtimes c_2$: Recebe um criptogramas RGSW e um RLWE C_1, C_2 respectivamente cifrados sobre a mesma chave e retorna $C_1 \boxtimes C_2 := C_1\mathbf{g^{-1}}(c_2)$. O novo criptograma obtido cifra a multiplicação das mensagens (μ_1, μ_2) pelo processo análogo ao anterior, suprimido para conscisão.

Para checar a implementação dos esquemas e cada uma das operações descritas acesse o github.

6 Empacotamento

Nesta seção, a 4 seção do artigo [2] é exposta resumidamente com comentários sobre a implementação.

Definições Seja $K = K_1 \otimes K_2 \otimes K_3$ um corpo tensorial de três corpos linearmente disjuntos, e $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$ sejam seus anéis de inteiros, respectivamente. Conclui-se que o anel de inteiros de K (denotado como $\mathcal{R}$) é isomorfo a $\mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$. Além disso,

- K_{12} e K_{13} denotam $K_1 \otimes K_2$ e $K_1 \otimes K_3$, respectivamente.
- $\mathcal{R}$, $\mathcal{R}_{12}$ e $\mathcal{R}_{13}$ denotam os anéis de inteiros de K, K_{12}, e K_{13}, respectivamente. Sabe-se que $\mathcal{R} \equiv \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$, $\mathcal{R}_{12} \equiv \mathcal{R}_1 \otimes \mathcal{R}_2$, e $\mathcal{R}_{13} \equiv \mathcal{R}_1 \otimes \mathcal{R}_3$.
- Sejam $(v_1, v_2, \ldots, v_\rho)$ e $(w_1, w_2, \ldots, w_\tau)$ algumas $\mathbb{Z}$-bases de $\mathcal{R}_2$ e $\mathcal{R}_3$, respectivamente, onde ρ e τ são os graus dos anéis $\mathcal{R}_2$ e $\mathcal{R}_3$.
- Denotamos $(v_1^\vee, v_2^\vee, \ldots, v_\rho^\vee)$ e $(w_1^\vee, w_2^\vee, \ldots, w_\tau^\vee)$ como as correspondentes $\mathbb{Z}$-bases dos espaços duais $\mathcal{R}_2^\vee$ e $\mathcal{R}_3^\vee$, respectivamente.
- Seja $r = \min(\rho, \tau)$, o número máximo de slots que nosso método pode empacotar.
- Denotamos as funções de traço (com respeito a diferentes subcorpos subjacentes) como

$$\mathrm{Tr}_{K/K_{12}} : K \to K_{12} \text{ e } \mathrm{Tr}_{K/K_{13}} : K \to K_{13}$$

6.1 Empacotamento de *plaintext*

O algoritmo recebe r mensagens $(x_1, \ldots, x_r) \in \mathcal{R}_1$ e um índice para um dos quatro modos: (1) $\mathcal{R}_{12}$, (2) $\mathcal{R}_{13}$, (3) $\mathcal{R}_{12} \to \mathcal{R}_{13}$, e (4) $\mathcal{R}_{13} \to \mathcal{R}_{12}$, ,modos esses mantidos exatamente na implementação. O resultado da função é:

- Modo 1: saída $\sum_{i=1}^{r} x_i \cdot v_i \in \mathcal{R}_{12}$.
- Modo 2: saída $\sum_{i=1}^{r} x_i \cdot w_i \in \mathcal{R}_{13}$.
- Modo 3: saída $\sum_{i=1}^{r} x_i \cdot v_i^\vee w_i \in \mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3$.
- Modo 4: saída $\sum_{i=1}^{r} x_i \cdot w_i^\vee v_i \in \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3^\vee$.

O algoritmo de empacotamento anexa um índice ao seu modo.

– **Adição.** O algoritmo recebe duas codificações, nomeadamente (x, modo_1), (y, modo_2), e produz $(x + y, \mathrm{modo}_1)$ se $\mathrm{modo}_1 = \mathrm{modo}_2$, caso contrário $\perp$.

– **Multiplicação.** O algoritmo recebe duas codificações, nomeadamente (x, modo_1) e (y, modo_2), e faz uma das seguintes operações, selecionada pelos modos.

- $\mathrm{modo}_1 = 1$ e $\mathrm{modo}_2 = 3$: saída $\mathrm{Tr}_{K/K_{13}}(xy) \in \mathcal{R}_{13}$.
- $\mathrm{modo}_1 = 3$ e $\mathrm{modo}_2 = 1$: saída $\mathrm{Tr}_{K/K_{12}}(xy) \in \mathcal{R}_{12}$.
- $\mathrm{modo}_1 = 2$ e $\mathrm{modo}_2 = 4$: saída $\mathrm{Tr}_{K/K_{13}}(xy) \in \mathcal{R}_{13}$.
- $\mathrm{modo}_1 = 4$ e $\mathrm{modo}_2 = 2$: saída $\mathrm{Tr}_{K/K_{12}}(xy) \in \mathcal{R}_{12}$.
- Caso contrário $\perp$.

Essas operações estão implementadas no github no directório plaintext_operations.

Nota: Para o plaintexts e ciphertexts, seus elementos vão ser armazenados em $\mathcal{R}$ ao invés do dual. Por exemplo, para o criptograma $a \in \mathcal{R}_1 \otimes \mathcal{R}_2^\vee \otimes \mathcal{R}_3$, vamos armazenar o criptograma $aP \in \mathcal{R}$ onde $P = \rho$. Esta multiplicação garante que o elemento tem representação única em $\mathcal{R}$ e facilita a implementação.

6.2 Empacotamento de *ciphertexts*

Para esta seção utilizamos uma função chamada de Eval-Trace, que recebe um criptograma cifrado em RLWE ou RGSW e devolve outro cifrando o traço da mensagem, ou seja:

$$\text{Eval-Tr}_{K/K_{13}} : \text{RLWE}_s(\mu) \to \text{RLWE}_s(\text{Tr}_{K/K_{13}}(\mu))$$

Tome-a como *black-box* na próxima seção o seu funcionamento vai ser detalhado, então vamos descrever os empacotamentos.

RGSW-Pack. O algoritmo recebe como entrada r RGSW ciphertexts $C_1, C_2, \ldots, C_r \in \mathcal{R}^{2\times 2^{2\ell}}$, onde cada C_i é RGSW(μ_i), para $\mu_i \in \mathcal{R}_1$ - no caso da implementação, recebe o elemento correspondente em $\mathcal{R}$, $\mu_i' = \mu_i \otimes 1$ mas seu funcionamento teórico permanece - e um índice para um dos quatro modos: (1) $\mathcal{R}_{12}$, (2) $\mathcal{R}_{13}$, (3) $\mathcal{R}_{12} \to \mathcal{R}_{13}$ e (4) $\mathcal{R}_{13} \to \mathcal{R}_{12}$. O algoritmo produz um RGSW ciphertext empacotado, de modo descrito a seguir

- Modo 1: saída $\sum_{i=1}^{r} C_i \cdot v_i \in \mathcal{R}^{2\times 2^{\ell}}$.
- Modo 2: saída $\sum_{i=1}^{r} C_i \cdot w_i \in \mathcal{R}^{2\times 2^{\ell}}$.
- Modo 3: saída $\sum_{i=1}^{r} C_i \cdot v_i^{\vee} w_i \in (\mathcal{R}_1 \otimes \mathcal{R}_2^{\vee} \otimes \mathcal{R}_3)^{2\times 2^{\ell}}$.
- Modo 4: saída $\sum_{i=1}^{r} C_i \cdot w_i^{\vee} v_i \in (\mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3^{\vee})^{2\times 2^{\ell}}$.

O algoritmo de empacotamento anexa um índice ao seu modo na clara.

– RLWE-Pack. O algoritmo recebe r RLWE ciphertexts $e_1, e_2, \ldots, e_r$, onde cada $e_i \in \text{RLWE}(\mu_i)$, para $\mu_i \in \mathcal{R}_1$ e um índice para um dos quatro modos iguais aos de RGSW-packing. O algoritmo produz uma codificação dos RLWE ciphertexts.

- Modo 1: saída $\sum_{i=1}^{r} c_i \cdot v_i \in \mathcal{R}_{12}$.
- Modo 2: saída $\sum_{i=1}^{r} c_i \cdot w_i \in \mathcal{R}_{13}$.
- Modo 3: saída $\sum_{i=1}^{r} c_i \cdot v_i^{\vee} w_i \in (\mathcal{R}_1 \otimes \mathcal{R}_2^{\vee} \otimes \mathcal{R}_3)$.
- Modo 4: saída $\sum_{i=1}^{r} c_i \cdot w_i^{\vee} v_i \in (\mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3^{\vee})$.

O modo é incluído no criptograma às claras e os elementos no dual são passados para o anel principal $\mathcal{R}$ como descrito na sessão anterior.

– Adição para RGSW-encodings: O algoritmo recebe duas codificações RGSW, nomeadamente (C_1, modo_1), (C_2, modo_2), e produz $(C_1 + C_2, \text{modo}_1)$ se $\text{modo}_1 = \text{modo}_2$, caso contrário $\perp$.

– Adição para RLWE-encodings: O algoritmo recebe duas codificações RLWE, nomeadamente (e_1, modo_1), (e_2, modo_2), e produz $(e_1 + e_2, \text{modo}_1)$ se $\text{modo}_1 = \text{modo}_2$, caso contrário $\perp$.

– Produto Homomórfico para RGSW-RGSW: O algoritmo recebe dois criptogramas RGSW empacotados, nomeadamente (C_1, modo_1) e (C_2, modo_2), e então calcula $C = C_1 \cdot G^{-1}(C_2)$. Então ele retorna:

- $\text{modo}_1 = 1$ e $\text{modo}_2 = 3$: saída Eval-$\text{Tr}_{K/K_{13}}(C)$.
- $\text{modo}_1 = 2$ e $\text{modo}_2 = 4$: saída Eval-$\text{Tr}_{K/K_{12}}(C)$.
- $\text{modo}_1 = 3$ e $\text{modo}_2 = 1$: saída Eval-$\text{Tr}_{K/K_{13}}(C)$.
- $\text{modo}_1 = 4$ e $\text{modo}_2 = 2$: saída Eval-$\text{Tr}_{K/K_{12}}(C)$.
- Caso contrário $\perp$.

– Eval-Mult (Produto Externo para RGSW-RLWE). O algoritmo recebe uma codificação RGSW (C_1, modo_1) e uma codificação RLWE (e_2, modo_2), e então calcula $C = C_1 \cdot g^{-1}(c_2)$. Então, retorna para cada caso:

- $\text{modo}_1 = 1$ e $\text{modo}_2 = 3$: saída Eval-$\text{Tr}_{K/K_{13}}(C)$.
- $\text{modo}_1 = 2$ e $\text{modo}_2 = 4$: saída Eval-$\text{Tr}_{K/K_{12}}(C)$.
- $\text{modo}_1 = 3$ e $\text{modo}_2 = 1$: saída Eval-$\text{Tr}_{K/K_{13}}(C)$.
- $\text{modo}_1 = 4$ e $\text{modo}_2 = 2$: saída Eval-$\text{Tr}_{K/K_{12}}(C)$.
- Caso contrário $\perp$.

Por que não empacotar as mensagens em um criptograma? O ruído acaba crescendo exponencialmente ao realizar múltiplos produtos externos consecutivamente. Para mais detalhes, vamos utilizar resultados das seções seguintes. Se decidirmos empacotar mensagens a expressão de um produto externo será:

$$Tr_{K/K_{13}}(e_1 g^{-1}(c_2) + \mu_1 e_2) + e'$$

onde e_1, e_2 são os ruídos do criptograma RGSW e RLWE respectivamente. Como o ruído não possui nenhuma propriedade especial em relação as bases v_i, w_i é válido afirmar que a sua norma vai crescer proporcional a um fator $\phi(p_2^{n_2})$. Portanto, após k aplicações de produto externo, a norma final será limitada com um fator de $\approx \phi(p_2^{n_2})^{k/2}\phi(p_3^{n_3})^{k/2}$ que é exponencial. Tal crescimento pode ser verificado via o gráfico a seguir:

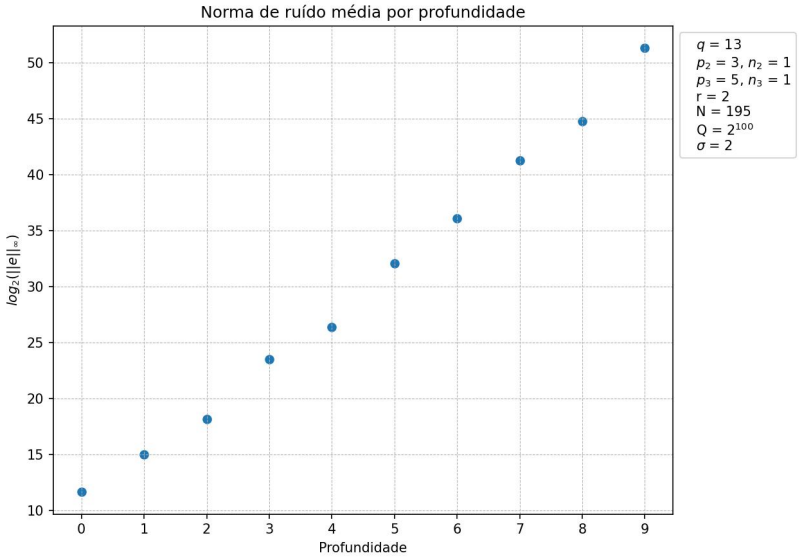

Figura 1: Crescimento do ruído pela quantidade de produtos externos com empacotamento errado.

7 Traço homomórfico sobre RLWE

O objetivo é encontrar uma maneira de executar o Eval-Trace, isto é, dado um criptograma $d \in \mathrm{RLWE}_s(\mu)$ computar um criptograma $d' \in \mathrm{RLWE}_s(Tr_{K/K_{13}}(\mu))$ homomorficamente, ou seja, sem ter acesso a mensagem original.

7.1 Algoritmo de Key Switch

Primeiramente, é necessário introduzir o conceito de troca de chave, *key-switch*. Como está no nome, esse algoritmo tem por objetivo trocar a chave na qual um criptograma está crifrado.

7.1.1 Descrição matemática do Algoritmo

1. **Criptograma Original:** Um criptograma RLWE $c = (a, b) \in \mathcal{R}^2$ cifra uma mensagem μ sob a chave s' se
$$b = a \cdot s' + e + \mu \cdot \Delta$$
onde e é o ruído e $\Delta = Q/B$.

2. **Chave de Troca de Chave ($K_{s' \to s}$):** É um conjunto de criptogramas $\{k_i\}$ que cifram múltiplos da antiga chave s' sob a nova chave s. Cada $k_i \in K_{s' \to s}$ é um criptograma $\mathrm{RLWE}_s(s' \cdot g[i])$.

3. **Procedimento:**

   - Calcula-se um criptograma de $a \cdot s'$ sob a nova chave s via
   $$(a_{\text{new}}, b_{\text{new}}) = \sum_{i=0}^{\ell-1} k_i \cdot g^{-1}(a)[i]$$
   - Agora vamos calcular o criptograma procurado,
   $$c' = (-a_{\text{new}}, b - b_{\text{new}}).$$

4. **Verificação:** Decifrando $c' = (a', b')$ com a chave s, temos:
$$b' + a' \cdot s = (b - b_{\text{new}}) - a_{\text{new}} \cdot s \approx b - (b_{\text{new}} + a_{\text{new}} \cdot s) \approx b - a \cdot s'.$$
Como $b = a \cdot s' + e + \mu \cdot \Delta$, resulta que:
$$b' + a' \cdot s \approx \mu \cdot \Delta + e.$$
Ou seja, a mensagem μ (com ruído e) agora está cifrada sob a nova chave s.

7.1.2 Pseudocódigo e implementação do key switch

Algorithm 2: Algoritmo de Troca de Chave (Key-Switching)

Input: Criptograma $c = (a, b) \in \mathcal{R}_Q^2$, uma cifra $\mathrm{RLWE}_{s'}(\mu)$.
Chave de troca de chave $K_{s' \to s} = \{k_i = \mathrm{RLWE}_s(s' \cdot g_i)\}_{i=0}^{\ell-1}$.
Output: Criptograma $c' \in \mathcal{R}_Q^2$, uma cifra $\mathrm{RLWE}_s(\mu)$.

$(a_0, a_1, \ldots, a_{\ell-1}) \leftarrow g^{-1}(a)$
$(a_{\text{new}}, b_{\text{new}}) \leftarrow (0, 0) \in \mathcal{R}_Q^2$.
for $i \leftarrow 0$ **to** $\ell - 1$ **do**
    Seja $k_i = (c_i, d_i)$.
    $(a_{\text{new}}, b_{\text{new}}) \leftarrow (a_{\text{new}} + a_i \cdot c_i, b_{\text{new}} + a_i \cdot d_i)$.
$a' \leftarrow -a_{\text{new}}$.
$b' \leftarrow b - b_{\text{new}}$
return $c' = (a', b')$

É claro que a implementação deste algoritmo é bem mais simples na prática. A seguir um snippet de código que contém não só a implementação da troca de chave no RLWE como RGSW, com a única diferença é que temos que aplicar a diferença para as linhas relevantes.

```
def key_switch_rlwe(ciphertext, K, base, B, q, N):
    f = Zx(cyclotomic_polynomial(N))
    Zqx = ZZ.quotient(q)['x']
    Rq = Zqx.quotient(f)

    a, b = ciphertext[0], ciphertext[1]
    ev_aK = vector(inv_g_poly(a, base, q, N)) * K

    ev_aK = [Rq(ev_aK[0]), Rq(-ev_aK[1])]
    return vector([-ev_aK[0], Rq(b) + Rq(ev_aK[1])])

def key_switch_gsw(gsw, K, base, C):
    l, B, q, n = gsw.l, gsw.B, gsw.q, gsw.n
    new_C = Matrix(gsw.Rq, 2* l, 2)

    for i in range(l):
        new_C[l+i] = key_switch_rlwe(C[l+i], K, base, B, q, n)
        new_C[i] = gsw.inv_g_row_ciphertext(new_C[l+i]) * gsw.Ks

    return new_C
```

7.2 Pseudocódigo do algoritmo

Basta apresentar o algoritmo utilizado para o Eval-trace, no pseudocódigo a seguir foi escolhido um traço em específico mas note que basta apenas trocá-lo pelo outro cenário.

Algorithm 3: (RLWE)-Eval-Tr$_{K/K_{13}}$ com a estrutura de torre

Entrada:

- $c = (b, a) \in (\mathcal{R})^2$ que encripta uma mensagem $\mu \in \mathcal{R}$ sob um segredo $s \in \mathcal{R}$.
- $\{\text{evk}^{(\sigma)}\}_{\sigma \in \bigcup_{i=1}^{t-1} \text{Gal}(E_i/E_{i+1})}$, e evk $\in \text{RGSW}_s(P^{-1} \cdot s) \in \mathcal{R}^2$.

Saída : $c_{out} \in \text{RLWE}_s(\text{Tr}_{K/K_{13}}(\mu))$.

1 Inicialize $c = (b, a)$
2 Calcule $d = \text{evk} \boxtimes (0, a)$
3 **for** $i = 1$ *até* $t - 1$ **do**
4 Tome $(d_1, d_2) \leftarrow d$
5 Calcule $d' = \sum_{\sigma \in \text{Gal}(E_i/E_{i+1})} \text{KS}(\sigma(d_1), \sigma(d_2), \text{evk}^{(\sigma^{-1})})$
6 Atualize $d = d'$ e $d' = (0, 0)$
7 Retorne $(\text{Tr}_{K/K_{13}}(b), 0) - d$

Perceba que como armazenamos os criptogramas já em $\mathcal{R}$ a multiplicação por tal fator aqui se torna desnecessária. Para computar as chaves necessárias evk e as relativas ao key-switch, a classe que implementa o esquema RGSW calcula tais valores assim que instanciada.

Para mostrar sua corretude, observe que antes do loop $d \in \text{RLWE}_s as$, durante o loop calculamos o traço encriptado as com ajudar do key-switch. Logo, o algortimo retorna

$$(\text{Tr}_{K/K_{13}}(b - as) - a's - e, -a') = (-a's + \text{Tr}_{K/K_{13}}(\mu)\frac{Q}{B} + e', -a')$$

ou seja, um criptograma válido de $\text{RLWE}_s(\text{Tr}_{K/K_{13}}(\mu))$.

8 Análise de Ruído das operações

8.1 Operações RGSW

Nesta seção estão as análises do crescimento de ruído das operações entre criptogramas RGSW.

8.1.1 Soma Homomórfica $C_1 \boxplus C_2$

O ruído dessa operação pode ser calculado por:

$$Err(C_1 \boxplus C_2) = sk^T(C_1 + C_2) - (\mu_1 + \mu_2)sk^T G$$

Organizando,

$$Err(C_1 \boxplus C_2) = sk^T(C_1) - (\mu_1)sk^T G + sk^T(C_2) - (\mu_2)sk^T G = e_1^T + e_2^T$$

8.1.2 Multiplicação Homomórfica $C_1 \boxdot C_2$

O ruído desta operação pode ser calculado por

$$Err(C_1 \boxdot C_2) = sk^\top(C_1 G^{-1}(C_2)) - (\mu_1\mu_2)sk^\top \mathbf{G}$$

Expandindo a expressão encontra-se:

$$Err(C_1 \boxdot C_2) = (1, -s)\begin{pmatrix} s\mathbf{a}_1^\top + \mathbf{e}_1^\top \\ \mathbf{a}_1^\top \end{pmatrix} G^{-1}(C_2) + \mu_1(1, -s)\begin{pmatrix} s\mathbf{a}_2^\top + \mathbf{e}_2^\top \\ \mathbf{a}_2^\top \end{pmatrix}$$

Simplificando, finalmente:

$$Err(C_1 \boxdot C_2) = \mathbf{e}_1^\top G^{-1}(C_2) + \mu_1 \mathbf{e}_2^\top$$

8.2 Ext-Prod

O produto externo entre RGSW e RLWE pode ser calculado por

$$C_1 \boxtimes c_2 = \begin{pmatrix} s\mathbf{a}_1^\top + \mathbf{e}_1^\top \\ \mathbf{a}_1^\top \end{pmatrix} g^{-1}(c_2) + \mu_1 \mathbf{G} g^{-1}(c_2)$$

$$= \begin{pmatrix} s\mathbf{a}_1^\top g^{-1}(c_2) + \mathbf{e}_1^\top g^{-1}(c_2) \\ \mathbf{a}_1^\top g^{-1}(c_2) \end{pmatrix} + \begin{pmatrix} sa_2\mu_1 + \mu_1\mu_2\frac{Q}{B} + \mu_1 e_2 \\ a_2\mu_1 \end{pmatrix}$$

$$= \begin{pmatrix} s(\mathbf{a}_1^\top g^{-1}(c_2) + a_2\mu_1) + \mu_1\mu_2\frac{Q}{B} + \mathbf{e}_1^\top g^{-1}(c_2) + \mu_1 e_2 \\ \mathbf{a}_1^\top g^{-1}(c_2) + a_2\mu_1 \end{pmatrix}$$

Finalmente, o ruído é calculado por:

$$Err(C_1 \boxtimes c_2) = \mathbf{e}_1^\top g^{-1}(c_2) + \mu_1 e_2$$

Dessa forma, a sua norma infinita deve ser

$$||Err(C_1 \boxtimes c_2)||_\infty < 2\ell N||e_1||_\infty + \mu_1||e_2||_\infty$$

8.3 Subgaussiana

Para a análise de ruído 'justa' é preciso utilizar que parte das variáveis aleatórias trabalhadas são limitadas por distribuições subgaussianas. A intuição é de que podemos encontrar um limite superior probabilístico utilizando a desigualdade de Markov se soubermos a distribuição que rege a variável aleatória.

Definição 8.1. Para $s > 0$, define-se a função Gaussiana $\rho_s : H \to (0, 1]$ por

$$\rho_s(\mathbf{x}) = \exp\left(-\pi\|\mathbf{x}\|_2^2/s^2\right).$$

A distribuição Gaussiana contínua D_s é obtida pela normalização de ρ_s, com densidade $s^{-n} \cdot \rho_s(\mathbf{x})$.

Define-se também que uma variável aleatória $X \in \mathbb{R}$ é δ-subgaussiana com parâmetro $s > 0$ se, para todo $t \in \mathbb{R}$,

$$\mathbb{E}[\exp(2\pi tX)] \leq \exp(\delta + \pi s^2 t^2).$$

Agora tome as seguintes propriedades presentes em [1] e [2].

Lema 8.2 (Lema 3.2 [2])**.** *Para quaisquer cifras RGSW* $\mathbf{C}_1, \mathbf{C}_2$ *que cifram* μ_1, μ_2 *com termos de erro* $\mathbf{e}_1, \mathbf{e}_2$ *respectivamente, temos o seguinte:*

$$Err(\mathbf{C}_1 \boxplus \mathbf{C}_2) = \mathbf{e}_1^\top + \mathbf{e}_2^\top.$$

$$Err(\mathbf{C}_1 \boxtimes \mathbf{C}_2) = \mathbf{e}_1^\top \cdot \mathbf{G}^{-1}(\mathbf{C}_2) + \mu_1 \cdot \mathbf{e}_2^\top.$$

Além disso, suponha que $\mathbf{G}^{-1}$ *é amostrado com respeito a alguma base* $\mathbb{Z}$ *de* $\mathcal{R}$*, isto é,* $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$*, tal que* $\max_{i\in[n]}\{\|\sigma(\mathbf{b}_i)\|_\infty\} \leq 1$ *como no Lema 2.3. Então os seguintes fatos valem:*

- *Denote* $\mathbf{e}_1^\top \cdot \mathbf{G}^{-1}(\mathbf{C}_2)$ *como* $\mathbf{e}^\top = (e_1, \ldots, e_{2\ell})$*. Então cada entrada de* $\mathbf{e}$ *é uma variável aleatória independente.*
- $\|\sigma(\mathbf{e})\|_\infty$ *é limitada superiormente por uma variável sub-Gaussiana com parâmetro* $O(r)$*, para algum* $r > 0$ *tal que* $r \leq \sqrt{N \cdot \log Q} \cdot \|\sigma(\mathbf{e}_1)\|_\infty$*.*

Lema 8.3. *Seja uma variável aleatória* X *de distribuição gaussiana de parâmetros* $(\mu = 0, \sigma)$*, esta variável é limitada superiormente por um subgaussiana de parâmetro mínimo* $\sqrt{2\pi}\sigma$

Demonstração. Basta computar $\mathbb{E}[2\pi tX]$

$$\mathbb{E}[2\pi tX] = \int_{-\infty}^{\infty} Ae^{-\frac{1}{2\sigma^2}X^2} e^{2\pi tX} dX$$

$$\int_{-\infty}^{\infty} Ae^{-\frac{1}{2\sigma^2}X^2} e^{2\pi tX} dX = \int_{-\infty}^{\infty} Ae^{-\frac{1}{2\sigma^2}X^2} e^{2\pi tX} e^{2\pi\sigma^2t^2} e^{-2\pi\sigma^2t^2} dX$$

$$\mathbb{E}[2\pi tX] = \int_{-\infty}^{\infty} Ae^{-\frac{1}{2\sigma^2}(X-2\pi\sigma^2t)^2} e^{2\pi^2\sigma^2t^2} dX$$

Como a segunda exponencial não depende de X,

$$\mathbb{E}[2\pi tX] = e^{2\pi^2\sigma^2t^2} \int_{-\infty}^{\infty} Ae^{-\frac{1}{2\sigma^2}(X-2\pi\sigma^2t)^2} dX$$

Note porém que a integral agora toma a forma, $\int_{-\infty}^{\infty} Ae^{-\frac{1}{2\sigma^2}(X-\mu)^2} dX$, ou seja, é apenas a função de probabilidade de uma gaussiana deslocada da origem integrada em sua completude, logo resulta em 1. Perceba que o processo anterior também funciona para uma gaussiana discreta, afinal se baseia apenas no processo de se aproveitar da normalização e do comportamento exponencial da gaussiana. Finalmente, para que seja uma distribuição subgaussiana de parâmetro s

$$\mathbb{E}[2\pi tX] = e^{2\pi^2\sigma^2t^2} \leq e^{\delta} e^{\pi s^2 t^2}$$

Como o intuito é que a desigualdade seja válida para qualquer t, basta que $2\pi^2\sigma^2t^2 \leq \pi s^2t^2 \Rightarrow 2\pi\sigma^2 \leq s^2 \Rightarrow s \geq \sigma\sqrt{2\pi}$ Chegando assim no resultado desejado.

□

Agora, podemos passar para uma análise dos ruídos acumulados pelas operações.

8.4 Key Switch

O algoritmo de key-switch, como presente no nome, tem como função a troca de um criptograma cifrado em uma chave antiga s', para uma chave nova s, o que aumenta o ruído gerado na cifra, vamos verificar o quanto. Para que o algortimo seja realizado corretamente, é necessário que seja passado como parâmetro a *lower half* de um criptograma $RGSW_s(s')$, ou seja, é necessário assumir que o RGSW tem segurança circular.

Seja então $d \in \text{RLWE}_{s'}(\mu)$ e $d = (a, b)$, e queremos então um criptograma de μ cifrado em RLWE_s. Considere então K a *lower half* de um criptograma $RGSW_s(s')$. O objetivo então é encontrar o ruído,

$$[0, b] - K \times g^{-1}(a)$$

Como K é a parte inferior de $RGSW_s(s')$ considere $K = (A, As + gs' + \mathbf{e}_{KS}) \in \mathcal{R}_Q^{2\times\ell}$. Aplicando na expressão anterior:

$$(g^{-1}(a) \times A), (g^{-1}(a) \times A)s + s'gg^{-1}(a) + \mathbf{e}_{KS}g^{-1}(a)$$

Utilizando a definição de g, $gg^{-1}(a) = a$, e subtraindo pelo vetor $[0, b]$, sabendo que $b = s'a + \mu \left\lfloor \frac{Q}{B} \right\rceil + E$:

$$-(g^{-1}(a) \times A), -(g^{-1}(a) \times A)s + \mu\frac{Q}{B} + E - \mathbf{e}_{KS}g^{-1}(a)$$

Note que o vetor obtido é um criptograma valido $\in \mathrm{RLWE}_s(\mu)$ com ruído

$$E - \mathbf{e}_{KS}g^{-1}(a)$$

onde E é o ruído inicial do criptograma e $\mathbf{e}_{KS}$ é o vetor de ruído utilizado para cifrar s. Então, a norma infinita do ruído key-switch é:

$$||Err(KS(d))||_\infty < ||Err(d)||_\infty + N\ell||\mathbf{e}_{KS}||_\infty$$

No teorema 4.6 do [2], assume-se que $Err(d)$ e $\mathbf{e}_{KS}g^{-1}(a)$ são subgaussianas com um mesmo parametro B, o que faz sentido visto que a magnitude do ruído do key-switch deve ser muito menor do que a da mensagem d.

8.5 Eval-Trace

Para a análise de ruído do algoritmo 4.1 de [2]. Para facilitar a análise, considere o mesmo algoritmo simplificado, para o efeito da acumulação do ruído ser mais visível, a variável d terá um índice que representa seu degrau na torre de extensão.

Algorithm 4: (RLWE)-Eval-$\mathrm{Tr}_{K/K_{13}}$ com a estrutura de torre

Entrada:

- $c = (b, a) \in (\mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3)^2$ que encripta uma mensagem $\mu \in \mathcal{R}_1 \otimes \mathcal{R}_2 \otimes \mathcal{R}_3$ sob um segredo $s \in \mathcal{R}$.
- $\{\mathrm{evk}^{(\sigma)}\}_{\sigma\in\bigcup_{i=1}^{t-1}\mathrm{Gal}(E_i/E_{i+1})}$, e $\mathrm{evk} \in \mathrm{RGSW}_s(P^{-1} \cdot s) \in \mathcal{R}^2$.

Saída : $c_{out} \in \mathrm{RLWE}_s(\mathrm{Tr}_{K/K_{13}}(\mu))$.

1 Inicialize $c = (b, a)$, defina $\bar{a} = P \cdot a$
2 Calcule $d_0 = \mathrm{evk} \boxtimes (0, \bar{a})$
3 **for** $i = 1$ *até* $t - 1$ **do**
4 Calcule $d_{i+1} = \sum_{\sigma\in\mathrm{Gal}(E_i/E_{i+1})} \mathrm{KS}(\sigma(d_i), \mathrm{evk}^{(\sigma^{-1})})$
5 Retorne $(\mathrm{Tr}_{K/K_{13}}(b), 0) - d_n$

Assuma sem perda de generalidade que o traço realizado é $K \to K_{13}$. Começaremos analisando a recurssão do loop na linha 4:

$$d_{i+1} = \sum_{\sigma\in\mathrm{Gal}(E_i/E_{i+1})} \mathrm{KS}(\sigma(d_i), \mathrm{evk}^{(\sigma^{-1})})$$

Então, utilizando a expressão do erro do key-switch:

$$Err(d_{i+1}) = \sum_{\sigma\in\mathrm{Gal}(E_i/E_{i+1})} Err(\sigma(d_i)) - e_{KS}^{\sigma}g^{-1}(d_i[0])$$

Perceba que $Err(\sigma(d_i)) = \sigma(Err(d_i))$ e que o segundo termo no somatório tem sua norma limitada por um valor pequeno, em uma análise inicial $N\ell||e_{KS}^{\sigma}||$, utilizando o lema 3.2 é uma variável subgaussiana com parâmetro limitado por $\sqrt{N\ell}||e_{KS}^{\sigma}||$. Visto que sua norma independe do seu estágio na torre, troque-o por e' onde $||e'|| \leq N\ell||e_{KS}^{\sigma}||$.

$$Err(d_{i+1}) < p_2e' + \sum_{\sigma\in\mathrm{Gal}(E_i/E_{i+1})} \sigma(Err(d_i))$$

Observe que o termo $\sum_{\sigma \in \mathrm{Gal}(E_i/E_{i+1})} \sigma(Err(d_i)) = Tr_{E_i/E_{i+1}}(Err(d_i))$

$$Err(d_{i+1}) < p_2 e' + Tr_{E_i/E_{i+1}}(Err(d_i))$$

A partir dessa recorrência, é possível chegar na relação:

$$Err(d_n) < Tr_{K/K_{13}}(Err(d_0)) + e'(p_2 - 1)\sum_{i=0}^{n_2-1} p_2^i < Tr_{K/K_{13}}(Err(d_0)) + \rho' e'$$

Logo,

$$Err(d_n) < Tr_{K/K_{13}}(Err(d_0)) + \rho' e'$$

Sabendo que $d_0 = evk \boxtimes [0, \bar{a}]$, $Err(d_0) = Err(evk)g^{-1}([0, \bar{a}])$. Tome f como o criptograma resultante, $f = (\mathrm{Tr}_{K/K_{13}}(b), 0) - d_n$. Então, expressão do ruído toma a forma:

$$Err(f) < Tr_{K/K_{13}}(e) + Tr_{K/K_{13}}(Err(d_0)) + \rho' e'$$

Ao considerar que as normas de ruído de $\mathbf{e}_{KS}$ e evk são limitadas pelo mesmo valor E, temos que $||Tr_{K/K_{13}}(Err(d_0))|| < 2N\ell E$, logo aplicando as normas:

$$||Err(f)|| < ||Tr_{K/K_{13}}(e)|| + 3\rho' N\ell E$$

Lembrando que $||e||$ é a norma do ruído do criptograma inicial. Note que no teorema 4.6 de [2] é proposta uma análise mais justa do erro, a única diferença é que os produtos da forma $\mathbf{e}g^{-1}(x)$, onde $||\mathbf{e}|| < E$ são subgaussianos com parametros limitados.

8.6 Ext-Prod + Trace

Lembrando algumas definições necessárias,

- $\rho = \phi(p_2^{n_2}), \rho' = p_2^{n_2}$ sua base v_i e seu dual v_i^v
- $\tau = \phi(p_3^{n_3}), \tau' = p_3^{n_3}$ sua base w_i e seu dual w_i^v
- $\delta_i^{(j)}$ componente i do ruído da mensagem $\mu_i^{(j)}$ empacotada cifrada em RGSW
- $e_i^{(j)}$ componente i do ruído da mensagem d_j resultante empacotada em RLWE

8.6.1 Aplicação única

Com o resultado da seção anterior e com o erro do produto externo, temos:

$$e^{(1)} < Tr_{K/K_{13}}(\sum_i \delta_i^{(1)} v_i^v w_i g^{-1}(d_0) + \mu^{(0)} e_i^{(0)} v_i) + \Delta$$

Onde $||\Delta|| < 3\rho' N\ell E$. Se aplicarmos a norma infinita da expressão, conseguimos obter:

$$||e^{(1)}|| < \frac{2\rho(p_2 - 1)N\ell}{\rho'} \sum ||\delta_i^{(1)}|| + \rho||\mu^{(1)}|| \sum ||e_i^{(0)}|| + 3\rho' N\ell$$

Perceba que essa expressão é análoga a expressão obtida no artigo ao efetuarmos uma análise utilizando variáveis subgaussianas.

8.6.2 k Aplicações

Finalmente, um vislumbre do fim. Calcular o ruído após k operações onde as mensagens $\mu_i^{(j)} = \xi_q^{t_i^{(j)}}$ simulando o que ocorre durante o batch bootstraping proposto. Suponha que o k é par.

$$e^{(k)} < Tr_{K/K_{13}}(\sum_i \delta_i^{(k)} v_i^v w_i g^{-1}(d_{k-1}) + \sum_s e_i^{(k-1)} v_i \mu_s^{(k)} v_s^v w_s) + \Delta$$

$$e^{(k)} < \sum_i e_i^{(k-1)} \xi_q^{t_i^{(k)}} w_i + Tr_{K/K_{13}}(\sum_i \delta_i^{(k)} v_i^v w_i g^{-1}(d_{k-1})) + \Delta$$

Vamos replicar a mesma coisa para $k-1$ lembrando que agora o traço efetuado será relativo ao corpo K_{12}:

$$e^{(k-1)} < \sum_i e_i^{(k-2)} \xi_q^{t_i^{(k-1)}} v_i + Tr_{K/K_{12}}(\sum_i \delta_i^{(k)} w_i^v v_i g^{-1}(d_{k-2})) + \Delta'$$

onde $||\Delta'|| \leq 3\tau' N\ell E$. Agora temos que isolar as componentes de $e^{(k-1)}$ utilizando o traço, por definição do dual temos:

$$e_i^{(k-1)} = Tr_{K/K_{13}}(e^{(k-1)} v_i^v)$$

Para facilitar essa conta tome $F = \sum_i e_i^{(k-1)} \xi_q^{t_i^{(k)}} w_i = \sum_i Tr_{K/K_{13}}(e^{(k-1)} v_i^v) \xi_q^{t_i^{(k)}} w_i$, expandindo:

$$F < \sum_i Tr_{K/K_{13}}\big(\big[\sum_j e_j^{(k-2)} \xi_q^{t_j^{(k-1)}} v_j + Tr_{K/K_{12}}(\sum_j \delta_j^{(k)} w_j^v v_j g^{-1}(d_{k-2})) + \Delta'\big] v_i^v\big) \xi_q^{t_i^{(k)}} w_i$$

Sabemos que o traço e somatórios são lineares, então denote cada termo onde o traço mas externo é aplicado de 1 até 3, $F < F_1 + F_2 + F_3$.

$$F_1 = \sum_i Tr_{K/K_{13}}\big(\sum_j e_j^{(k-2)} \xi_q^{t_j^{(k-1)}} v_j v_i^v\big) \xi_q^{t_i^{(k)}} w_i = \sum_i e_i^{(k-2)} \xi_q^{t_i^{(k)} + t_i^{(k-1)}} w_i$$

Agora para o segundo termo:

$$F_2 = \sum_i Tr_{K/K_{13}}\big(Tr_{K/K_{12}}\big(\sum_j \delta_j^{(k)} w_j^v v_j g^{-1}(d_{k-2})\big) v_i^v\big) \xi_q^{t_i^{(k)}} w_i$$

É importante calcularmos uma norma infinita para F_2. Estão sendo somados r traços logo,

$$||F_2|| \leq r||Tr_{K/K_{13}}\big(Tr_{K/K_{12}}\big(\sum_j \delta_j^{(k)} w_j^v v_j g^{-1}(d_{k-2})\big) v_i^v\big) \xi_q^{t_i^{(k)}} w_i||$$

$$||F_2|| \leq r \cdot \rho\tau||\big(\sum_j \delta_j^{(k)} w_j^v v_j g^{-1}(d_{k-2})\big)|| \cdot ||v_i^v|| \cdot ||\xi_q^{t_i^{(k)}}|| \cdot ||w_i||$$

$$||F_2|| \leq r \cdot \rho\tau r E||w_j^v|| \cdot ||v_i^v||$$

$$||F_2|| \leq 4p_2p_3r^2 N\ell E$$

Para o último termo,

$$F_3 = \sum_i Tr_{K/K_{13}}\big(\Delta' v_i^v\big) \xi_q^{t_i^{(k)}} w_i$$

Também segue uma norma,

$$||F_3|| = 2p_1 r||\Delta'||$$

Concluímos que as normas de F_2, F_3 **não** tem dependência com termos de ruído de k!
Substituindo de volta na equação do $e^{(k)}$,

$$e^{(k)} < \sum_i e_i^{(k-2)} \xi_q^{t_i^{(k)} + t_i^{(k-1)}} w_i + F_2 + F_3 + Tr_{K/K_{13}}(\sum_i \delta_i^{(k)} v_i^v w_i g^{-1}(d_{k-1})) + \Delta$$

Finalmente, concatemos em um resultado de norma, considere $||e^{(k-2)}|| < ||\sum_i e_i^{(k-2)} \xi_q^{t_i^{(k)} + t_i^{(k-1)}} w_i||$

$$||e^{(k)}|| < ||e^{(k-2)}|| + ||F_2|| + ||F_3|| + ||2rp_1 N\ell E|| + ||\Delta||$$

$$||e^{(k)}|| < ||e^{(k-2)}|| + 4p_2p_3r^2 N\ell E + 2p_2 r||\Delta'|| + 2rp_2 N\ell E + ||\Delta||$$

$$||e^{(k)}|| < ||e^{(0)}|| + \frac{k}{2}(4p_2p_3r^2 N\ell E + 6p_2 r\tau' N\ell E + 2rp_2 N\ell E + 3\rho' N\ell E)$$

Simplificando a expressão para:

$$||e^{(k)}|| < ||e^{(0)}|| + \frac{k}{2}(4p_2p_3r^2 + 6p_2r\tau' + 2rp_2 + 3\rho')N(\log Q)E$$

Sobre a análise 'justa', utilizando o lema 8.3, basicamente o que mudam são as normas das operações envolvendo os vetores g, artigo ajustasse o parâmetro da subgaussiana para $r\sqrt{N \log Q}E$ tendo assim como resultado final:

$$||e^{(k)}|| < ||e^{(0)}|| + \frac{k}{2}(4p_2p_3r^2 + 6p_2r\tau' + 2rp_2 + 3\rho')r\sqrt{N \log Q}E$$

Veja a figura a seguir comparando o crescimento a norma:

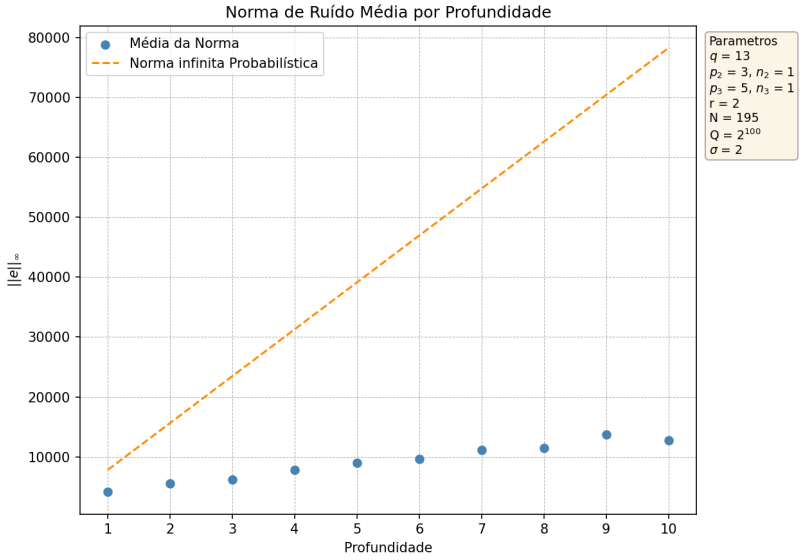

Figura 2: Crescimento do ruído pela quantidade de produtos externos.

9 Complexidades e Definições de parâmetro

9.1 Complexidade de Memória

Trivialmente, um inteiro com tamanho máximo Q pode ser representado por $\log Q$ bits. Logo um polinômio de grau N com seus coeficientes de tamanho máximo a $N \log Q$

- RGSW: $2N \log^2 Q$(evk)$+[\phi(m_2)+\phi(m_3)] \log Q$(Automorfismos)$+[\phi(m_2)+\phi(m_3)]2N \log^2 Q(KS)+$ $6 \log Q$(máximo superestimado) $\approx O(rN \log^2 Q)$
- RLWE $6 \log Q$(máximo superestimado)
- chaves $4N \log Q$(máximo superestimado)
- Total parece ser proporcional a $\approx O(rN \log^2 Q)$

Criptogramas RGSW $4N \log^2 Q$ / Criptogramas RLWE $2N \log Q$

9.2 Complexidade de Tempo

Produto externo + Traço

Produto Externo:

- Decompor dois polinomios, decompor $\phi(N)$ vezes a complexidade de decompor um inteiro na base necessária (base normalmente é 2)
- Multiplicar um vetor de $(1 \times 2\ell)$ por uma matrix $(2\ell \times 2)$ onde cada indíce é um polinômio representado por $N \log Q$ bits, basicamente 4ℓ multiplicações e $4\ell - 2$ adições.

Traço Homomórfico:

- Produto externo
- Se o corpo reduzido tem como potência de primo m, vamos ter $\log d$ chamadas de key-switch, cada key switch faz 2ℓ multiplicacoes e $2\ell - 2$ somas.

  No total seria algo como $\log d \times (2\ell$ multiplicações $+ 2\ell$ adições)

Referências

[1] Lyubashevsky, V., Peikert, C., Regev, O.: *A toolkit for ring-LWE cryptography.* In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 35–54. Springer, Heidelberg (2013). Available at: `https://eprint.iacr.org/2013/293.pdf`

[2] Liu, F.-H., Wang, H.: *Batch Bootstrapping I: A New Framework for SIMD Bootstrapping in Polynomial Modulus.* In: Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23–27, 2023, Proceedings, Part III, pp. 321–352. Springer, Berlin, Heidelberg (2023). Available at: `https://doi.org/10.1007/978-3-031-30620-4_11`

[3] Liu, F.-H., Wang, H.: *Batch Bootstrapping II: Bootstrapping in Polynomial Modulus only Requires $\tilde{O}(1)$ FHE Multiplications in Amortization.* In: Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23–27, 2023, Proceedings, Part III, pp. 353–384. Springer, Berlin, Heidelberg (2023). Available at: `https://doi.org/10.1007/978-3-031-30620-4_12`

[4] Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC, pp. 575–584. ACM Press (2013)

[5] Balbás, D.: The Hardness of LWE and Ring-LWE: A Survey. Cryptology ePrint Archive, Paper 2021/1358 (2021)