Implementação em Tempo Constante de Amostragem de Gaussianas Discretas

Jheyne N. Ortiz[1], **Diego F. Aranha**[1], **Ricardo Dahab**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Campinas – SP – Brasil

Abstract. *Algorithms for discrete Gaussian sampling over lattices require discrete methods for sampling over integers. Considering that sampling from $\mathcal{D}_{\Lambda,\sigma,\mathbf{c}}$ is a step when encrypting in certain modern lattice-based cryptosystems, side-channel resistant implementations of discrete Gaussian sampling over integers are desirable. This work presents a constant-time implementation of the Knuth-Yao and Ziggurat methods comparing them with their variable-time counterparts. The Knuth-Yao constant-time implementation is applied to a Gaussian sampler over lattices.*

Resumo. *Algoritmos de amostragem Gaussiana discreta sobre reticulados demandam métodos discretos de amostragem sobre inteiros. Considerando que a amostragem de $\mathcal{D}_{\Lambda,\sigma,\mathbf{c}}$ é um dos passos na encriptação em certos criptosistemas modernos baseados em reticulados, surge a preocupação por implementações resistentes a ataques por canais laterais. Este trabalho apresenta e discute implementações dos métodos Knuth-Yao e Ziggurat em tempo constante comparando-as com suas versões de tempo variável. A implementação em tempo constante do Knuth-Yao é aplicada na amostragem sobre reticulados.*

1. Introdução

Criptografia Baseada em Reticulados se apresenta como uma alternativa promissora a sistemas criptográficos assimétricos, no que tange a resistência contra poder computacional quântico. Desde o trabalho seminal de Ajtai [Ajtai 1996], já foram propostos na literatura acadêmica esquemas para encriptação [Stehlé et al. 2009, Lyubashevsky et al. 2010] e variantes [Agrawal et al. 2010, Mochetti and Dahab 2014], assinaturas digitais [Hoffstein et al. 2010] e acordo de chaves. Mesmo que sejam conhecidos esquemas cuja segurança depende da dificuldade de certos problemas em reticulados, sua viabilidade prática depende ainda da disponibilidade de implementações seguras e eficientes, tanto em *software* quanto em *hardware*.

Este trabalho tem como objetivo colaborar com o aprimoramento de implementações em *software* de Criptografia Baseada em Reticulados, ao estudar algoritmos e implementações para o problema de amostragem de elementos discretos segundo uma distribuição Gaussiana. Esta operação de amostragem é utilizada, por exemplo, para se realizar a amostragem de elementos de um reticulado, necessária durante o processo de derivação de chaves e de encriptação. Por esse motivo, é também crítica do ponto de vista de segurança, pois a exploração de canais laterais para atacar esquemas de encriptação baseada em reticulados já foi demonstrada [Roy et al. 2014].

A principal contribuição desse trabalho é propor e comparar estratégias de implementação em tempo constante para amostragem de uma distribuição Gaussiana discreta sobre inteiros. Apesar de alguns trabalhos realizarem essa tarefa para implementações em *hardware*, este parece ser o primeiro tratamento rigoroso em *software*. Versões adaptadas dos algoritmos Ziggurat [Buchmann et al. 2014] e Knuth-Yao [Knuth and Yao 1976, Dwarakanath and Galbraith 2014] são projetadas para execução em tempo invariante com a entrada. Nesse contexto, o algoritmo Knuth-Yao se apresenta como substancialmente mais amigável a implementações seguras, observando-se uma penalidade de desempenho de quinze vezes sobre a implementação do algoritmo Knuth-Yao convencional. A operação de amostragem de elementos de um reticulado é então implementada utilizando a amostragem Gaussiana sobre inteiros, a título de ilustração e completude.

O trabalho está organizado da seguinte forma: na Seção 2 são discutidas abordagens para amostragem Gaussiana em *hardware*. Os fundamentos teóricos são descritos na Seção 3, bem como algoritmos para amostragem Gaussiana sobre inteiros e em reticulados. A Seção 4 apresenta as estratégias de implementação, enfatizando as adaptações para resistência contra canais laterais de tempo. Resultados experimentais são analisados na Seção 5, seguida pela última seção dedicada a apresentar conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Nesta seção são apresentados alguns trabalhos que tratam do problema de amostragem segura de uma distribuição Gaussiana discreta sobre inteiros em *hardware*. Um tratamento mais aprofundado do problema da amostragem Gaussiana pode ser encontrado em [Folláth 2014].

Em [Roy et al. 2014], os autores apresentam uma implementação em *hardware* em tempo constante do método de amostragem Knuth-Yao, aplicável a esquemas de encriptação tais como o de Lyubashevsky, Peikert e Regev [Lyubashevsky et al. 2010], baseado no problema LWE (*Learning with Errors*) sobre anéis. Nesse esquema, a amostragem de uma distribuição discreta ocorre na geração de chaves e na encriptação. A proteção contra ataques laterais é feita de duas formas: as colunas da matriz de probabilidade são utilizadas para indexar as possíveis amostras da distribuição e armazená-las em memória ROM, na forma de tabelas de *lookup* compactas com acesso constante; e um embaralhador é utilizado para misturar as amostras na etapa final do algoritmo.

[Pöppelmann and Güneysu 2014] apresentam uma implementação em FPGA do algoritmo RING-LWEENCRYPT, que inclui um amostrador Gaussiano. Este amostrador segue o método da transformação inversa e é executado em tempo constante. O módulo como um todo, responsável pela encriptação, decriptação e amostragem de valores discretos, é resistente a ataques por análise de tempo, porém é vulnerável a injeção de falha e outros ataques de canal lateral.

Finalmente, [de Clercq et al. 2015] apresentam uma implementação eficiente de encriptação baseada em reticulados em microcontroladores ARM Cortex-M4F. Apesar do amostrador ser capaz de calcular uma amostra no tempo médio de apenas 28,5 ciclos, não há qualquer proteção contra ataques de canal lateral.

3. Fundamentação Teórica

Para um valor inteiro $q \geq 2$, $\mathbb{Z}_q$ denota o anel de inteiros módulo q, e $\mathbb{Z}_q[x]$ denota o anel de polinômios na variável x e coeficientes em $\mathbb{Z}_q$. $\mathbb{Z}_q^{n\times m}$ denota uma matriz de $n \times m$ elementos do anel $\mathbb{Z}_q$. Matrizes são denotadas por letras maiúsculas em negrito, por exemplo $\mathbf{A}$, enquanto que vetores são denotados por letras minúsculas em negrito, por exemplo $\hat{\mathbf{a}}$. Matrizes às vezes serão denotadas também pelo seu conjunto (ordenado) de colunas, por exemplo, $\mathbf{A} = \{\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_n}\}$. $\mathbf{A}\|\mathbf{B}$ denota a matriz resultante da concatenação das colunas das matrizes $\mathbf{A}$ e $\mathbf{B}$, ambas tendo o mesmo número de linhas. Para dois vetores $\mathbf{a}, \mathbf{b}$ de dimensão n, as operações de produto interno e norma são definidas e denotadas por:

$$\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} (\mathbf{a}_i \cdot \mathbf{b}_i) \text{ e } \|\mathbf{b}\| = \sqrt{\langle \mathbf{b}, \mathbf{b} \rangle}.$$

Estendemos a definição de norma para a matriz $\mathbf{A} = \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n\}$, onde $\|\mathbf{A}\| = \mathbf{max}_{1\leq i\leq n}\|\mathbf{a}_i\|$. O anel de polinômios $\mathbb{Z}_q[x]$ módulo um polinômio ciclotômico é denotado por $\mathcal{R}$, e, como de costume, $\mathcal{R}^m$ representa o conjunto dos vetores de dimensão m em $\mathcal{R}$.

3.1. Reticulados

Um reticulado Λ de dimensão m é um subgrupo discreto com posto completo do conjunto $\mathbb{R}^m$. Uma base $\mathbf{A}$ de Λ é um conjunto de vetores linearmente independentes maximal de Λ. A subclasse de reticulados ideais corresponde a ideais no anel de polinômios $\mathcal{R}^0 = \mathbb{Z}[x]/f(x)$. Quando $f(x) = x^n + 1$, o reticulado é dito cíclico e a base $\mathbf{A}$ é igual a $Rot_f(\hat{\mathbf{a}}) \in \mathbb{Z}_q^{n\times mn}$, com $\hat{\mathbf{a}} \in \mathcal{R}^m$, e $\mathcal{R} = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$. A operação $Rot_f(\hat{\mathbf{a}})$ é a concatenação $[rot_f(\hat{\mathbf{a}}_1)\|...\|rot_f(\hat{\mathbf{a}}_m)]$, onde $rot_f(\mathbf{a})$, $\mathbf{a} \in \mathcal{R}$, é a expansão de $\mathbf{a}$ na matriz $\mathbb{Z}_q^{n\times n}$, definida por

$$rot_f(\mathbf{a}) = [x^0\mathbf{a}, x^1\mathbf{a}, ..., x^{n-1}\mathbf{a}],$$

e as multiplicações são tomadas módulo f.

3.2. Distribuição Gaussiana Discreta $\mathcal{D}_{\mathbf{\Lambda},\sigma,\mathbf{c}}$

A distribuição Gaussiana discreta, descrita pelo centro $c \in \mathbb{R}$ e desvio padrão $\sigma \in \mathbb{R}^+$, é uma função que determina a probabilidade de um valor inteiro x ser amostrado dentre os demais:

$$\mathcal{D}_{\sigma,c}(x) = \frac{\rho_{\sigma,c}(x)}{\sum_{y=-\infty}^{\infty} \rho_{\sigma,c}(y)}, \tag{1}$$

sendo que ρ denota a função Gaussiana $\rho_{\sigma,c}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-c)^2}{2\sigma^2}}$.

Tendo em vista que o intervalo de amostragem definido na Equação 1 é infinito, algoritmos em máquinas finitas são capazes de amostrar somente elementos de distribuições Gaussianas estatisticamente próximas da ideal. Neste sentido, as amostragens se dão no intervalo $[c - t\sigma, c + t\sigma]$, com t o comprimento da cauda da distribuição.

Num esquema criptográfico, tanto os parâmetros da distribuição como a distância estatística são determinados por cálculos oriundos da demonstração de segurança do esquema, a qual é singular para cada esquema. Para satisfazer a distância estatística inferior

a $2^{-\lambda}$, a implementação deve considerar precisão mínima de λ bits para as operações de ponto flutuante. Saídas com múltipla precisão podem ser simuladas via software pelo uso de bibliotecas tais como a NTL [Shoup 2015] para a linguagem C++.

O método mais trivial de amostragem é o método por rejeição em que um valor inteiro é obtido aleatoriamente no intervalo $[c-t\sigma, c+t\sigma]$ e é aceito se a sua probabilidade for menor ou igual a um valor aleatório no intervalo $[0, 1)$. Outro método tradicional é o método da inversão. Neste método, a amostra é o maior valor inteiro cuja probabilidade é inferior a uma probabilidade obtida aleatoriamente. Além destes, existem outros métodos como o Knuth-Yao [Knuth and Yao 1976, Folláth 2014] e a versão discreta do método de Ziggurat [Buchmann et al. 2014].

Knuth-Yao. O algoritmo Knuth-Yao [Knuth and Yao 1976, Folláth 2014] utiliza o conceito de árvore DDG (*Discrete Distribution Generating*) para gerar uma amostra que será o rótulo de um nó desta árvore. A árvore é descrita pela matriz de probabilidades $\mathbf{P} \in \mathbb{Z}_2^{2t\sigma \times \lambda}$, que consiste na expansão binária das probabilidades dos elementos no intervalo $[c-t\sigma, c+t\sigma]$. Sendo um método finito, as probabilidades são calculadas com λ bits de precisão podem ser armazenadas em uma fase preliminar para uso na amostragem.

Algoritmo 1: KNUTH-YAO($\mathbf{P} \in \mathbb{Z}_2^{2t\sigma \times \lambda}$)

```
1  d ← 0;
2  hit ← 0;
3  col ← 0;
4  while hit = 0 do
5      r ←$ Z_2;
6      d ← 2d + 1 − r;
7      for row ← 2tσ to 0 do
8          d ← d − P[row][col];
9          if d = −1 then
10             S ← row;
11             hit ← 1;
12             break;
13     col ← col + 1;
14 return S;
```

No Algoritmo 1, a linha 5 descreve a amostragem de um elemento de $\mathbb{Z}_2$, 0 ou 1, com igual probabilidade. Em geral, $a \leftarrow^{\$} X$ denota a amostragem de um dentre os elementos de X distribuidos uniformemente. O caminho na árvore DDG é construído à medida em que a distância d entre o nó que está sendo visitado e o nó interno mais à direita é computada. Assim sendo, quando a distância d é igual a -1, um nó terminal foi atingido e seu rótulo é retornado como resultado da amostra. Outra abordagem sugere que existe um nó terminal com o rótulo row no nível col da árvore, se, e somente se, $P[row][col] = 1$. O método Knuth-Yao supõe a existência de um gerador de bits aleatórios que determina, a cada passo, a direção a ser tomada na árvore.

Ziggurat Discreto. A versão discreta do método Ziggurat, originalmente contínuo [Marsaglia and Tsang 2000], foi proposta recentemente por [Buchmann et al. 2014]. O método Ziggurat consiste em duas fases: particio-

namento e amostragem. Na fase de amostragem, a região abaixo da função densidade de probabilidade é dividida em m retângulos de mesma área. O algoritmo de particionamento computa, iterativamente, os valores de x e y que delimitam os limites de cada retângulo, iniciando com $x_m = t\sigma$ e $y_m = 0$.

Algoritmo 2: ZIGGURAT-DISCRETO$(m, \sigma, \omega, \{\lfloor x_1 \rfloor, ..., \lfloor x_m \rfloor\}, \{\bar{y}_0, ..., \bar{y}_m\})$

```
1  while true do
2      i ←$ {1, ..., m};
3      s ←$ {−1, 1};
4      x ←$ {0, ..., ⌊x_i⌋};
5      if 0 < x ≤ ⌊x_{i−1}⌋ then
6          return sx;
7      else
8          if x = 0 then
9              b ←$ {0, 1};
10             if b = 0 then
11                 return sx;
12         else
13             y′ ←$ {0, ..., 2^ω − 1};
14             ȳ ← y′ · (ȳ_{i−1} − ȳ_i);
15             if ȳ ≤ 2^ω · (ρ_σ(x) − ȳ_i) then
16                 return sx;
```

Na fase de amostragem, descrita no Algoritmo 2, um retângulo, assim como um valor inteiro x neste retângulo, e um sinal s, são obtidos aleatoriamente. A seguir, utilizando um procedimento de rejeição, a amostra candidata x é avaliada e então aceita, se enquadrada em alguma das seguintes situações. No primeiro caso, a amostra está contida mais à esquerda na área abaixo da função densidade e é, com certeza, uma amostra válida. No segundo teste, o valor amostrado é nulo e é aceito com 50% de probabilidade. Em último caso, x é um valor próximo da curva e, o que se sucede, é um cálculo que determina se esse valor está abaixo ou acima da curva. Se estiver abaixo, o valor de x é retornado como sendo a amostra.

Em [Buchmann et al. 2014], os autores propõem uma versão otimizada do Algoritmo 2, que contém casos mais específicos que consideram a curvatura da curva. Nesta versão, o cálculo da probabilidade $\rho_\sigma(x)$ é evitada e realizada somente em última instância. A computação de ρ inclui o cálculo de exponenciais, operação que consome a maior parte do tempo de processamento do algoritmo de amostragem [Buchmann et al. 2014].

3.3. Ortogonalização de Gram-Schmidt

O processo de ortogonalização de Gram-Schmidt consiste em produzir, a partir de um conjunto de vetores linearmente independentes $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$, um conjunto ortogonal $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n\}$ que gera o mesmo subespaço de $\mathbf{B}$. A ortogonalização de Gram-Schmidt para $\mathbf{B}$ é uma base única tal que uma dessas propriedades deve ser satisfeita:

- $\forall k \in [1, ..., n], \tilde{\mathbf{b}}_k = \mathbf{b}_k - \mathbf{Proj}(\mathbf{b}_k, \mathbf{B}_{k-1})$;

- $\forall k \in [1, ..., n], \tilde{\mathbf{b}}_k = \mathbf{b}_k - \sum_{j=1}^{k-1} \frac{\langle \mathbf{b}_k, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|^2} \tilde{\mathbf{b}}_j$;
- $\forall k \in [1, ..., n], \tilde{\mathbf{b}}_k \perp \mathbf{B}_{k-1}$ e $(\mathbf{b}_k, \tilde{\mathbf{b}}_k) \in \mathbf{B}_{k-1}$.

A redução de Gram-Schmidt do vetor $\mathbf{b}_k$ é dada por $\tilde{\mathbf{b}}_k$. O processo usual de Gram-Schmidt é apresentado no Algoritmo 3.

Algoritmo 3: GRAMSCHMIDT-PROCESS($\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$)

for $i = 1$ **to** n **do**
    $\tilde{\mathbf{b}}_i \leftarrow \mathbf{b}_i$;
    for $j = 1$ **to** $(i - 1)$ **do**
        $\mu_{i,j} = \frac{\langle \mathbf{b}_i, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|^2}$;
        $\tilde{\mathbf{b}}_i \leftarrow \tilde{\mathbf{b}}_i - \mu_{i,j} \tilde{\mathbf{b}}_j$;
return $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n\}$;

[Lyubashevsky and Prest 2015] apresentam algoritmos otimizados para ortogonalização de bases isométricas e bases compostas por várias bases isométricas. Uma base é dita isométrica se, para uma base ordenada $\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$ de um reticulado, que é subconjunto de um espaço de produto inteiro $\mathcal{H}$, existe uma isometria r tal que

$$\forall k \in [2, ..., n], \mathbf{b}_k = r(\mathbf{b}_{k-1}) = x\mathbf{b}_{k-1}.$$

Em relação ao procedimento usual de Gram-Schmidt, a primeira versão otimizada é $O(n)$ mais rápida e efetua $5n^2$ multiplicações e adições. O Algoritmo 4, uma segunda otimização, realiza $3n^2$ multiplicações e adições [Lyubashevsky and Prest 2015].

Algoritmo 4: FASTER-ISOMETRIC-GSO($\mathbf{B} = \{\mathbf{b}_1, ..., \mathbf{b}_n\}$)

$\tilde{\mathbf{b}}_1 \leftarrow \mathbf{b}_1$;
$\mathbf{v}_1 \leftarrow \mathbf{b}_1$;
$C_1 \leftarrow \langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_1) \rangle$;
$D_1 \leftarrow \|\mathbf{b}_1\|^2$;
for $k = 1$ **to** $(n - 1)$ **do**
    $\tilde{\mathbf{b}}_{k+1} \leftarrow r(\tilde{\mathbf{b}}_k) - \frac{C_k}{D_k} \mathbf{v}_k$;
    $\mathbf{v}_{k+1} \leftarrow \mathbf{v}_k - \frac{C_k}{D_k} r(\tilde{\mathbf{b}}_k)$;
    $C_{k+1} \leftarrow \langle \mathbf{v}_1, r(\tilde{\mathbf{b}}_{k+1}) \rangle$;
    $D_{k+1} \leftarrow D_k - \frac{{C_k}^2}{D_k}$;
return $\tilde{\mathbf{B}} = \{\tilde{\mathbf{b}}_1, ..., \tilde{\mathbf{b}}_n\}$;

Além disso, o Algoritmo 4 pode ser utilizado como passo intermediário na geração da base ortogonal de um conjunto de vetores ordenados, que é a união de várias bases isométricas $\mathbf{B}^i$.

[Lyubashevsky and Prest 2015] propõem o Algoritmo 5 especificamente para bases que são um conjunto de bases isométricas. A entrada consiste em um conjunto de m bases isométricas $\mathbf{B}^{(i)}$ e a saída é uma base ortogonal com a mesma dimensão de $\mathbf{B}$, tal que $\tilde{\mathbf{B}} \in \mathbb{R}^{n \times nm}$ ou $\tilde{\mathbf{B}} \in \mathbb{Q}^{n \times nm}$. Neste procedimento, o primeiro vetor de cada bloco $\mathbf{B}^{(i)}$

Algoritmo 5: BLOCK-GSO($\mathbf{B} = \{\mathbf{B}^{(1)}, ..., \mathbf{B}^{(m)}\} = \{\mathbf{b}_1, ..., \mathbf{b}_{mn}\}$)

for $i = 0$ **to** $(m-1)$ **do**
  $\tilde{\mathbf{b}}_{ni+1} \leftarrow \mathbf{b}_{ni+1} - \sum_{j=1}^{ni} \frac{\langle \mathbf{b}_{ni+1}, \tilde{\mathbf{b}}_j \rangle}{\|\tilde{\mathbf{b}}_j\|^2} \tilde{\mathbf{b}}_j$;
  $\tilde{\mathbf{B}}^{(i+1)} \leftarrow \{\tilde{\mathbf{b}}_{ni+1}, r(\tilde{\mathbf{b}}_{ni+1}), ..., r^{n-1}(\tilde{\mathbf{b}}_{ni+1})\}$;
  $\tilde{\mathbf{B}}^{(i+1)} \leftarrow$ FASTER-ISOMETRIC-GSO($\tilde{\mathbf{B}}^{(i+1)}$);
return $\tilde{\mathbf{B}} = \{\tilde{\mathbf{B}}^1, ..., \tilde{\mathbf{B}}^m\}$;

é tornado ortogonal aos vetores dos blocos 1 a $i-1$. A seguir, o vetor $\tilde{\mathbf{b}}_{ni+1}$ é expandido por meio da isometria r, e sua expansão é, então, fornecida como entrada para o algoritmo FASTER-ISOMETRIC-GSO. A saída do algoritmo BLOCK-GSO é a concatenação das m bases ortogonais.

3.4. Distribuição Gaussiana Discreta $\mathcal{D}_{\Lambda,\sigma,\mathbf{c}}$

Uma Distribuição Gaussiana Discreta $\mathcal{D}_{\Lambda,\sigma,\mathbf{c}}$ sobre um reticulado Λ é definida pelos parâmetros $\mathbf{c} \in \mathbb{R}^n$, $\sigma > 0$, seu centro e o desvio padrão, e por uma base $\mathbf{B}$ do reticulado Λ. Amostragens Gaussianas sobre Λ produzem vetores com probabilidades que obedecem a $\mathcal{D}_{\Lambda,\sigma,\mathbf{c}}$, contanto que o valor do desvio padrão σ da amostra seja maior ou igual à norma do vetor mais longo da base $\tilde{\mathbf{B}}$, multiplicada por um pequeno fator.

Os algoritmos para amostragem Gaussiana constroem iterativamente os coeficientes que irão compor o vetor-amostra resultante. A cada repetição, um valor inteiro é obtido de uma distribuição $\mathcal{D}_{\mathbb{Z},\sigma,c}$, cujos parâmetros são re-computados a cada chamada, que pode ser implementada com algum dos métodos apresentados na Seção 3.2.

Dentre os algoritmos para amostragem Gaussiana, SAMPLED é capaz de obter vetores de qualquer reticulado e tem complexidade $O(n^2)$ mais o custo de n chamadas para amostragem de inteiros de $\mathcal{D}_{\mathbb{Z},\sigma,c}$ [Gentry et al. 2008]. O método SAMPLED é similar ao Algoritmo 6, incluindo apenas a computação do vetor $\mathbf{v}$, inicialmente nulo, e a cada passo um novo valor é obtido do anterior por meio da expressão $\mathbf{v}_{i-1} \leftarrow \mathbf{v}_i + z_i \mathbf{b}_i$.

Algoritmo 6: GAUSSIAN-SAMPLER($\sigma, \mathbf{c}, \mathbf{B}, \tilde{\mathbf{B}}$)

$\mathbf{c}_n \leftarrow \mathbf{c}$;
for $i = n$ **to** 1 **do**
  $c'_i = \langle \mathbf{c}_i, \tilde{\mathbf{b}}_i \rangle / \langle \tilde{\mathbf{b}}_i, \tilde{\mathbf{b}}_i \rangle$;
  $\sigma'_i = \sigma / \|\tilde{\mathbf{b}}_i\|$;
  $z_i \leftarrow D_{\mathbb{Z},\sigma'_i,c'_i}$;
  $\mathbf{c}_{i-1} \leftarrow \mathbf{c}_i - z_i \mathbf{b}_i$;
return $(\mathbf{c} - \mathbf{c}_0) \in \mathcal{D}_{\Lambda(\mathbf{B}),\sigma,\mathbf{c}}$;

[Lyubashevsky and Prest 2015] consideram como definição padrão de algoritmo para amostragem sobre reticulados uma versão simplificada do SAMPLED, o Algoritmo 6 - GAUSSIAN-SAMPLER. Esses dois algoritmos requerem que a base ortogonalizada $\tilde{\mathbf{B}}$ esteja disponível em memória física durante toda a execução da amostragem de $\mathcal{D}_{\Lambda(\mathbf{B}),\sigma,\mathbf{c}}$. Para vetores grandes, e armazenados com precisão de λ bits, o consumo de memória

torna-se demasiado. Neste sentido, Lyubashevsky e Prest propõem uma versão compacta do Algoritmo 6, a saber, o algoritmo COMPACT-GAUSSIAN-SAMPLER.

Algoritmo 7: COMPACT-GAUSSIAN-SAMPLER$(\sigma, \mathbf{c}, \mathbf{v}_n, \tilde{\mathbf{b}}_n, \mathbf{B}, \mathbf{H}, \mathbf{I})$

$\mathbf{c}_n \leftarrow \mathbf{c}$;
for $i = n$ **to** 1 **do**
  $c'_i = \langle \mathbf{c}_i, \tilde{\mathbf{b}}_i \rangle / \langle \tilde{\mathbf{b}}_i, \tilde{\mathbf{b}}_i \rangle$;
  $\sigma'_i = \sigma / \|\tilde{\mathbf{b}}_i\|$;
  $z_i \leftarrow D_{\mathbb{Z},\sigma'_i,c'_i}$;
  $\mathbf{c}_{i-1} \leftarrow \mathbf{c}_i - z_i \mathbf{b}_i$;
  $\tilde{\mathbf{b}}_{i-1} \leftarrow r^{-1}(\mathbf{H}_{i-1}\tilde{\mathbf{b}}_i + \mathbf{I}_{i-1}\mathbf{v}_i)$;
  $\mathbf{v}_{i-1} \leftarrow \mathbf{I}_{i-1}\tilde{\mathbf{b}}_i + \mathbf{H}_{i-1}\mathbf{v}_i$;
return $(\mathbf{c} - \mathbf{c}_0) \in \mathcal{D}_{\Lambda(\mathbf{B}),\sigma,\mathbf{c}}$;

Adicionalmente, o Algoritmo 7 requer os valores pré-computados de $\mathbf{H}$, $\mathbf{I}$, $\mathbf{v}_n$ e $\tilde{\mathbf{b}}_n$, conforme definição em [Lyubashevsky and Prest 2015]. Assim sendo, em um processo reverso, os valores de $\mathbf{v}_{i-1}$ e $\tilde{\mathbf{b}}_{i-1}$ são obtidos conforme a execução da amostragem no i-ésimo passo. Um amostrador compacto torna viável a amostragem para reticulados com bases que são blocos de bases isométricas sobre parâmetros grandes, tais como a dimensão dos vetores e a quantidade de blocos. Para os algoritmos anteriores, fazia-se necessário armazenar $mn \times n$ valores reais da base $\tilde{\mathbf{B}}$ com precisão de λ bits, sendo m a quantidade de blocos e n a dimensão dos vetores. Neste algoritmo, somente mn valores reais com tal precisão precisam estar disponíveis a cada iteração do algoritmo. Em contrapartida, a versão compacta é, no máximo, 3 vezes mais lenta [Lyubashevsky and Prest 2015].

4. Implementação de Amostragem de Gaussianas Discretas

Diversos esquemas criptográficos baseados em reticulados, tais como NTRU [Hoffstein et al. 2010], HIBE (*Hierarchical Identity-Based Encryption*) [Mochetti and Dahab 2014] e PKE-PEKS [Chen and Lin 2014], requerem amostras de distribuições Gaussianas discretas sobre inteiros e sobre reticulados. Tomando como exemplo o esquema HIBE proposto em [Mochetti and Dahab 2014], propósito geral das implementações descritas nesta seção, o sistema requer amostragens com distribuições Gaussianas discretas $\mathcal{D}_{\Lambda,\sigma,\mathbf{c}}$ e $\mathcal{D}_{\mathbb{Z},\sigma,c'}$ para a derivação de uma chave privada para cada usuário de uma estrutura hierárquica.

Considerando que as saídas do amostrador farão parte da chave secreta derivada de uma identidade pública ou de uma encriptação, é de interesse que a implementação da amostragem tenha tempo constante além de ser eficiente. Em uma implementação com tempo de execução constante, a latência independe da entrada e, portanto, não revela informações sensíveis ao adversário sobre a saída. Por outro lado, implementações com tempo de execução variável tornam-se suscetíveis a ataques por canais laterais.

Assim, esta seção apresenta implementações em C++ utilizando a biblioteca NTL [Shoup 2015] tanto em tempo constante como em tempo variável para os métodos Ziggurat e Knuth-Yao para amostragens com distribuições Gaussianas discretas sobre inteiros que, posteriormente, foram aplicados para amostragem com distribuições sobre

reticulados. As implementações Knuth-Yao e ZigguratO têm tempo de execução variável enquanto que Knuth-YaoC e ZigguratC apresentam tempo constante.

ZigguratO. O algoritmo ZigguratO, proposto em [Buchmann et al. 2014] tem o mesmo embasamento teórico que o algoritmo Ziggurat [Folláth 2014] porém inclui testes adicionais que avaliam a concavidade da curva. O objetivo da otimização de Buchmann et al. é evitar a computação da função ρ, que utiliza operações de ponto flutuante de alta precisão. O ZigguratO é implementado usando desvios condicionais aninhados e, em último caso, computa a probabilidade da amostra. Assim sendo, a amostra é retornada no momento de aceitação por algum dos testes, implicando em diferentes tempos de execução. Apesar de ser suscetível a ataques por análise de tempo, esta implementação é capaz de obter cerca de 500.000 amostras por segundo.

Knuth-Yao. Com a finalidade de avaliar a degradação no tempo de execução da versão em tempo constante do algoritmo Knuth-Yao, a versão convencional deste método foi implementada seguindo o pseudo-código de [Folláth 2014]. O algoritmo termina quando a distância entre o nó visitado e o nó-folha mais à direita é igual a -1 e, assim como o algoritmo ZigguratO, é suscetível a ataques por análise de tempo.

ZigguratC. A fim de evitar os desvios condicionais inerentes ao Algoritmo 2, um seletor foi utilizado para atribuir uma amostra à variável de retorno `S`. Além disso, os Testes 1 e 2 verificam as relações de "igual a" e "menor que", respectivamente, entre inteiros em termos de operações binárias.

Teste 1. Teste de igualdade em tempo constante.

```
1 unsigned isEqual(int x, int y) {
2     unsigned equal = (x-y);
3     return (1 ^ ((equal | -equal) >> 31) & 1);
4 }
```

Teste 2. Teste de "menor que" em tempo constante.

```
1 unsigned lessThan(int x, int y) {
2     unsigned less = (x-y);
3     return (less >> sizeof(int)*8-1);
4 }
```

O laço, que antes dependia da aceitação da amostra sx, passou a executar uma quantidade fixa de iterações. O número de iterações é definido empiricamente conforme o conjunto de parâmetros. Considerando que nesta versão todos os testes de avaliação de uma amostra são executados, o emprego da versão otimizada do algoritmo, denotada por ZigguratO [Buchmann et al. 2014], somente aumenta o tempo de execução do algoritmo. Considerando ainda que a computação da probabilidade de uma amostra candidata x é sempre realizada, esta implementação tem como referência o Algoritmo 2 que contém somente os três testes básicos (Seção 3).

Knuth-YaoC. Originalmente, os acessos à matriz de probabilidades seguem a ordem das colunas, que são percorridas de baixo para cima. Visando o acesso contínuo à memória, o algoritmo foi adaptado para acessar a matriz de probabilidades na ordem da disposição de suas linhas da esquerda para a direita. Uma segunda adaptação do algoritmo permite a amostragem de distribuições Gaussianas discretas centradas em qualquer ponto $c' \in \mathbb{R}$.

Por definição, o algoritmo é apto somente para amostragens com distribuições centradas em zero e, assim como o método Ziggurat, é inadequado para aplicação em amostragem sobre reticulados (veja os parâmetros c'_i e σ'_i nos Algoritmos 6 e 7).

Para que a implementação tenha tempo de execução independente da entrada, uma quantia fixa de iterações devem ser concluídas. Porém, tendo em vista o algoritmo usual, continuar a execução do laço resulta na sobrescrita da amostra obtida anteriormente. A solução empregada para o algoritmo Knuth-Yao combina uma somatória com operações binárias. As operações binárias testam se a distância é igual a -1 e definem o valor da variável de controle `enable` como 1 em caso afirmativo. Caso contrário, `enable` é zero. Uma outra variável, `hit`, contém a informação de que uma amostra já foi aceita. Com esses dois valores inteiros, um seletor determina se ou o valor inválido `invalidSample` – fora do intervalo $[c'-t\sigma, c'+t\sigma]$ – ou se o valor da coluna `col`, que é o rótulo da amostra, será somado à variável de saída `S`. No término do algoritmo, o valor de `S` é da forma $k \cdot invalidSample + col$, com $k \in \mathbb{Z}$. Assim sendo, a redução módulo `invalidSample` remove o erro e a operação $S = S - (t + \sigma) + c'$ obtém o valor correto da amostra.

A implementação como um todo, que é uma aplicação das implementações de amostragem de $\mathcal{D}_{\mathbb{Z},\sigma,c'}$, inicialmente gera as chaves mestres do sistema HIBE sobre reticulados ideais proposto em [Mochetti and Dahab 2014] que consiste nos quatro algoritmos: SETUP, KEYDERIVE, ENC e DEC. O algoritmo de geração de chaves mestres SETUP utiliza o algoritmo IDEALTRAPGEN de Stehlé et al. [Stehlé et al. 2009] para determinar o vetor $\hat{\mathbf{a}} \in \mathcal{R}^m$ e uma matriz $\mathbf{T_A} \in \mathcal{R}_0^{m \times m}$. A base curta $\mathbf{S} \in \mathbb{Z}^{mn \times mn}$ resulta da aplicação do operador rot_f a cada componente de $\mathbf{T_A}$ sendo que $\mathbf{S}$ gera o mesmo subgrupo que $\mathbf{A} = Rot_f(\hat{\mathbf{a}})$.

Dada a base curta $\mathbf{S}$, a base ortogonal $\tilde{\mathbf{S}}$ é obtida da chamada ao algoritmo GRAMSCHMIDT-PROCESS. Com a base ortogonal, define-se o desvio padrão como um valor real tal que $\sigma \geq \|\tilde{\mathbf{S}}\| \cdot \omega(\sqrt{log\ n})$. Por fim, o algoritmo de amostragem GAUSSIAN-SAMPLER, utilizando as bases $\tilde{\mathbf{S}}$ e $\mathbf{S}$, obtém uma amostra da distribuição $\mathcal{D}_{\Lambda(\mathbf{A}),\sigma,\mathbf{c}}$ utilizando a implementação em tempo constante do algoritmo Knuth-Yao para obter valores inteiros amostrados conforme $\mathcal{D}_{\mathbb{Z},\sigma,c'}$.

5. Análise e Resultados

A Figura 1 apresenta o tempo de execução dos algoritmos ZigguratO, Knuth-Yao, ZigguratC e Knuth-YaoC para amostragem de coeficientes de polinômios de dimensões variadas. As dimensões consideradas foram 1024, 2048, 4096 e 8192, e os tempos de execução são tomados como média de mil execuções. Para os algoritmos ZigguratC e ZigguratO, o tempo de execução para cada dimensão considera o tempo de particionamento da função de densidade mais o tempo de amostragem de coeficientes para o polinômio. Nos algoritmos Knuth-Yao e Knuth-YaoC, o tempo de execução é a soma do tempo necessário para criação da matriz de probabilidades e do tempo de amostragem dos coeficientes.

Da Figura 1, existe uma degradação no tempo de execução do método Ziggurat quando implementado em tempo constante. O algoritmo ZigguratO é capaz de gerar cerca de 500.000 amostras por segundo. O algoritmo ZigguratC, por sua vez, amostra somente 25.000 no mesmo espaço de tempo. Essa discrepância é resultado, principalmente, das operações de ponto flutuante realizadas na computação da probabilidade da amostra,

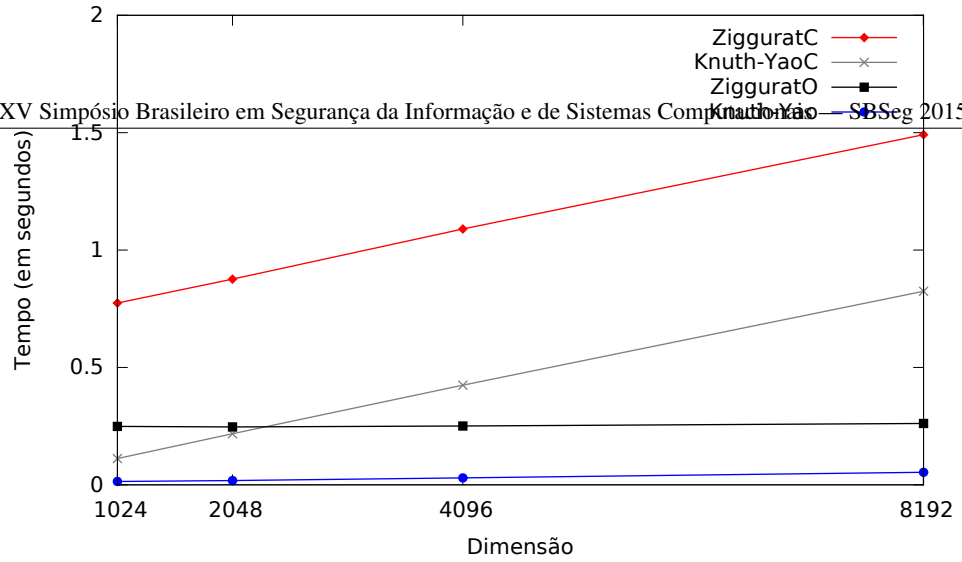

Figura 1. Tempo de execução dos algoritmos ZigguratO, ZigguratC, Knuth-Yao e Knuth-YaoC variando a dimensão dos polinômios. Os resultados foram obtidos de uma máquina com processador Intel Core i7-3632QM @ 2.20 GHz e 8 GB de memória física.

computação evitada no método ZigguratO devido à inserção de novos testes de rejeição.

Na versão em tempo constante do método Knuth-Yao, a execução prossegue mesmo que uma amostra tenha sido aceita e toda a matriz de probabilidade é visitada. No algoritmo, a computação sobre os valores nulos em $\mathbf{P}$ são evitados, tendo em vista que não surte efeito no cálculo de distância no laço mais inteiro. Para tanto, define-se para cada linha o primeiro e o último valor não nulo.

Apesar de o algoritmo ZigguratO se apresentar menos eficiente do que o algoritmo Knuth-Yao para polinômios de grandes dimensões, como, por exemplo, quinhentos mil ou um milhão de coeficientes, o algoritmo ZigguratO torna-se consideravelmente mais eficiente, podendo atingir tempo de execução três vezes menor.

As execuções de ambos os métodos considera o seguinte conjunto de parâmetros: desvio padrão $\sigma = 3,195$, precisão de $\lambda = 107$ bits e comprimento de cauda $t = 13$. Para o método Ziggurat, especificamente, o número de retângulos é $m = 63$ e o parâmetro $\omega = 107$, seguindo a precisão em bits. O conjunto de parâmetros é definido para cada esquema e depende da demonstração de segurança. O conjunto adotado nessa análise não é específico para um sistema e é uma compilação de valores adotados em [Roy et al. 2014] e [Buchmann et al. 2014].

A Tabela 1 sumariza o desempenho do algoritmo SETUP, que tem como subrotina o algoritmo IDEALTRAPGEN de [Stehlé et al. 2009]; do algoritmo GRAMSCHMIDT-PROCESS, que cria a base ortogonalizada $\tilde{\mathbf{S}}$; e do algoritmo GAUSSIAN-SAMPLER, que efetua amostragem da distribuição Gaussiana discreta sobre o reticulado Λ. Os resultados de menor magnitude são tomados como média de dez execuções enquanto que para os algoritmos GRAMSCHMIDT-PROCESS e GAUSSIAN-SAMPLER os tempos são média de somente duas execuções.

O algoritmo SETUP gera um par de chaves $(msk, mpk) = (\mathbf{T_A}, \{\hat{\mathbf{a}}, \hat{\mathbf{a}}'_j, \hat{\mathbf{b}}, \mathbf{u}\})$, sendo que $\hat{\mathbf{a}} \in \mathcal{R}^m$ e $\mathbf{T_A} \in \mathcal{R}_0^{m \times m}$ são gerados pelo algoritmo IDEALTRAPGEN e os demais vetores são obtidos aleatoriamente. O algoritmo IDEALTRAPGEN requer quatro parâmetros: a dimensão dos vetores dada por n, o número de bases isométricas m em $\mathbf{A}$, o módulo q dos coeficientes nos vetores e o polinômio ciclotômico $f(x) = x^n + 1$. Assim sendo, os resultados resumidos na Tabela 1 consideram os conjunto de parâmetros

Algoritmo	$(64, 133, 1019, x^{64}+1)$	$(16, 111, 499, x^{16}+1)$
SETUP	0,67	0,10
GRAMSCHMIDT-PROCESS	61036,90	565,43
GAUSSIAN-SAMPLER	31627,90	2060,04

Tabela 1. Tempo de execução, em segundos, para os algoritmos SETUP, GRAMSCHMIDT-PROCESS e GAUSSIAN-SAMPLER para dois conjuntos de parâmetros. Os resultados foram obtidos de uma máquina com processador Intel Core i7-3632QM @ 2.20 GHz e 8 GB de memória física.

$(n, m, q, f) = (64, 133, 1019, x^{64}+1)$ e $(n, m, q, f) = (16, 111, 499, x^{16}+1)$ para uma hierarquia de $h = 10$ níveis. Vale ressaltar que esses conjuntos de parâmetros são a título de ilustração e visam somente a aplicação da implementação segura do Knuth-Yao. Um conjunto de parâmetros real com nível de segurança de 128 bits se assemelha a $(n, m, q, f) = (128, 183, 2083, x^{128} + 1$. Além disso, para o sistema HIBE [Mochetti and Dahab 2014] especificamente, tal parametrização requer mais espaço em memória do que o provido por computadores convencionais.

Apesar dos altos tempos de execução para a geração da base ortogonalizada e para a amostragem sobre um reticulado, vale ressaltar a frequência com que esses algoritmos são requisitados no esquema HIBE. A base $\mathbf{S}$ é gerada na fase de configuração e se mantém enquanto a instância do sistema estiver ativa. O processo de ortogonalização de $\mathbf{S}$, que resulta na base $\tilde{\mathbf{S}}$, é executado somente quando a chave pública do sistema for modificada e no momento da primeira configuração. A amostragem da distribuição Gaussiana discreta, por sua vez, é executa uma única vez para cada derivação de chave. Supondo uma hierarquia com k elementos, seriam necessárias k amostras da distribuição $\mathcal{D}_{\Lambda(\mathbf{A}),\sigma,\mathbf{c}}$, uma para cada identidade pública.

6. Considerações Finais

Neste artigo foram apresentadas implementações em *software* com tempo de execução constante para os métodos de amostragem de Gaussiana discreta Knuth-Yao [Knuth and Yao 1976, Folláth 2014] e Ziggurat Discreto [Buchmann et al. 2014]. As implementações apresentaram detrimento no desempenho em prol de serem resistentes a ataques por análise de tempo. O desempenho das duas implementações foram avaliadas a fim de construir vetores de dimensões 1024, 2048, 4096 e 8192 com coeficientes amostrados da distribuição $\mathcal{D}_{\mathbb{Z},\sigma,c'}$.

No contexto da Criptografia Baseada em Reticulados, tais implementações foram aplicadas para amostragem de vetores de um reticulado $\Lambda(\mathbf{A})$ para a derivação de chaves de um sistema HIBE (*Hierarchical Identity-Based Encryption*) baseado em reticulados ideais [Mochetti and Dahab 2014]. Por se tratarem de reticulados ideais, a base pública $\mathbf{A}$ pode ser armazenada em sua versão reduzida, resumindo-se a um vetor de m polinômios de dimensão n.

O processo KEYDERIVE do sistema HIBE [Mochetti and Dahab 2014], responsável pela derivação de chaves privadas para usuários da hierarquia, define a nova chave como sendo um vetor amostrado da distribuição $\mathcal{D}_{\Lambda(\mathbf{A}),\sigma,\mathbf{c}}$. A amostragem Gaussiana discreta sobre reticulados requer a ortogonalização de $\mathbf{S}$, a expansão da matriz $\mathbf{T_A} \in \mathcal{R}_0^{m\times m}$ através do operador rot_f aplicado a cada componente.

Apesar de as bases $\hat{\mathbf{a}}$ e $\mathbf{T_A}$ serem compactas sobre os respectivos anéis, os algoritmos de ortogonalização e amostragem sobre o reticulado requerem sua versão expandida além do fator acrescentado devido à quantidade de bits necessários nas computações em ponto flutuante em alta precisão. Independentemente da especificidade, a implementação dos algoritmos GRAMSCHMIDT-PROCESS e GAUSSIAN-SAMPLER apresentou latências elevadas, consequência da dimensão da base do reticulado em sua forma rotacionada e das operações em ponto flutuante com alta precisão. Em contrapartida, os algoritmos não são invocados com alta frequência no sistema HIBE tal como as operações de encriptação e decriptação.

Considerando que o conjunto de parâmetros $(n, m, q, f) = (64, 133, 1019, x^{64} + 1)$ requer cerca de 7 GB de memória RAM durante a execução dos métodos de ortogonalização e amostragem sobre o reticulado, o algoritmo COMPACT-GAUSSIAN-SAMPLER de [Lyubashevsky and Prest 2015] pode mostrar-se mais adequado apesar de apresentar perda de desempenho. Alternativamente, o método híbrido de [Ducas and Prest 2015] para reticulados ideais pode resultar em latências reduzidas em relação às relatadas na Seção 5. A análise de viabilidade e a possível implementação desses métodos ficam como trabalhos futuros.

Referências

[Agrawal et al. 2010] Agrawal, S., Boneh, D., and Boyen, X. (2010). Efficient Lattice (H)IBE in the Standard Model. In Gilbert, H., editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 553–572. Springer Berlin Heidelberg.

[Ajtai 1996] Ajtai, M. (1996). Generating Hard Instances of Lattice Problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(7).

[Buchmann et al. 2014] Buchmann, J., Cabarcas, D., Göpfert, F., Hülsing, A., and Weiden, P. (2014). Discrete Ziggurat: A Time-Memory Trade-Off for Sampling from a Gaussian Distribution over the Integers. In Lange, T., Lauter, K., and Lisoněk, P., editors, *Selected Areas in Cryptography – SAC 2013*, volume 8282 of *Lecture Notes in Computer Science*, pages 402–417. Springer Berlin Heidelberg.

[Chen and Lin 2014] Chen, Y. and Lin, D. (2014). Generic Constructions of Integrated PKE and PEKS. (2013):1–34.

[de Clercq et al. 2015] de Clercq, R., Roy, S. S., Vercauteren, F., and Verbauwhede, I. (2015). Efficient software implementation of ring-LWE encryption. In Nebel, W. and Atienza, D., editors, *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, DATE 2015*, pages 339–344. ACM.

[Ducas and Prest 2015] Ducas, L. and Prest, T. (2015). A Hybrid Gaussian Sampler for Lattices over Rings. Cryptology ePrint Archive, Report 2015/660. `http://eprint.iacr.org/`.

[Dwarakanath and Galbraith 2014] Dwarakanath, N. and Galbraith, S. (2014). Sampling from discrete Gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing*, 25(3):159–180.

[Folláth 2014] Folláth, J. (2014). Gaussian Sampling in Lattice Based Cryptography. *Tatra Mountains Mathematical Publications*, 60(1):1–23.

[Gentry et al. 2008] Gentry, C., Peikert, C., and Vaikuntanathan, V. (2008). Trapdoors for Hard Lattices and New Cryptographic Constructions. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 197–206, New York, NY, USA. ACM.

[Hoffstein et al. 2010] Hoffstein, J., Howgrave-Graham, N., Pipher, J., and Whyte, W. (2010). Practical Lattice-Based Cryptography: NTRUEncrypt and NTRUSign. In Nguyen, P. Q. and Vallée, B., editors, *The LLL Algorithm*, Information Security and Cryptography, pages 349–390. Springer Berlin Heidelberg.

[Knuth and Yao 1976] Knuth, D. and Yao, A. (1976). *Algorithms and Complexity: New Directions and Recent Results*, chapter The Complexity of Nonuniform Random Number Generation. Academic Press, New York, j. f. traub edition.

[Lyubashevsky et al. 2010] Lyubashevsky, V., Peikert, C., and Regev, O. (2010). On Ideal Lattices and Learning with Errors over Rings. In Gilbert, H., editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer Berlin Heidelberg.

[Lyubashevsky and Prest 2015] Lyubashevsky, V. and Prest, T. (2015). Quadratic Time, Linear Space Algorithms for Gram-Schmidt Orthogonalization and Gaussian Sampling in Structured Lattices. Cryptology ePrint Archive, Report 2015/257. `http://eprint.iacr.org/`.

[Marsaglia and Tsang 2000] Marsaglia, G. and Tsang, W. W. (2000). The Ziggurat Method for Generating Random Variables. *Journal of Statistical Software*, 5(8):1–7.

[Mochetti and Dahab 2014] Mochetti, K. and Dahab, R. (2014). Ideal Lattice-based (H)IBE Scheme. Technical Report IC-14-18, Institute of Computing, University of Campinas.

[Pöppelmann and Güneysu 2014] Pöppelmann, T. and Güneysu, T. (2014). Towards Practical Lattice-Based Public-Key Encryption on Reconfigurable Hardware. In Lange, T., Lauter, K., and Lisoněk, P., editors, *Selected Areas in Cryptography – SAC 2013*, volume 8282 of *Lecture Notes in Computer Science*, pages 68–85. Springer Berlin Heidelberg.

[Roy et al. 2014] Roy, S. S., Reparaz, O., Vercauteren, F., and Verbauwhede, I. (2014). Compact and Side Channel Secure Discrete Gaussian Sampling. Cryptology ePrint Archive, Report 2014/591. `http://eprint.iacr.org/`.

[Shoup 2015] Shoup, V. (2015). Number Theory Library (NTL). Website. `http://www.shoup.net/ntl/` - Último acesso em 24 de junho de 2015.

[Stehlé et al. 2009] Stehlé, D., Steinfeld, R., Tanaka, K., and Xagawa, K. (2009). Efficient Public Key Encryption Based on Ideal Lattices. In Matsui, M., editor, *Advances in Cryptology - ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 617–635. Springer.

