Universidade do Minho
Escola de Ciências

Maria Zita Fiqueli de Abreu

Esquemas de assinatura digital *Lattice-based* e experimentação de certificados híbridos com criptografia pós-quântica

setembro de 2020

Esquemas de assinatura digital *Lattice-based* e experimentação de certificados híbridos com criptografia pós-quântica

Maria Zita Fiqueli de Abreu

UMinho | 2020

Universidade do Minho
Escola de Ciências

Maria Zita Fiqueli de Abreu

Esquemas de assinatura digital *Lattice-based* e experimentação de certificados híbridos com criptografia pós-quântica

Dissertação de Mestrado
Mestrado em Matemática e Computação

Trabalho efetuado sob a orientação de:
Orientador: **Professor Doutor José Pedro Miranda Mourão Patrício**
Orientador: **Professor Doutor José Carlos Bacelar Almeida**
Tutora: **Engenheira Carla Coutinho**

setembro de 2020

AGRADECIMENTOS

A realização da presente dissertação só foi possível graças a um leque de factores propícios e a um apoio gigantesco concedido por muitos e que merece o meu profundo agradecimento.

Primeiramente gostaria de agradecer ao Professor Pedro Patrício e ao Professor José Carlos Bacelar pela permanente disponibilidade, paciência, colaboração e por todos os conselhos valiosos que me facultaram. Foi um privilégio fazer este percurso tão bem orientada e guardarei com muito carinho toda esta experiência.

Gostaria de agradecer também à engenheira Carla Coutinho e ao engenheiro Nuno Ponte pela disponibilidade, simpatia e por participarem nesta etapa tão importante do meu percurso académico.

Para finalizar agradeço com todo o meu coração aos de sempre, aos que acreditam em mim sem hesitar. Esta é uma conquista fruto de todo o apoio imensurável da minha família e dos meus amigos mais próximos.

Zita Abreu

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

RESUMO

Os algoritmos criptográficos pós-quânticos têm como premissa de segurança a dificuldade na resolução de problemas matemáticos que se conjecturam difíceis na computação quântica. O interesse em implantar esses algoritmos tem vindo a crescer por forma a que a informação esteja protegida contra ataques quânticos no futuro. O *National Institute of Standards and Technology* (NIST) tem, atualmente, aberto um concurso para a seleção de algoritmos criptográficos pós-quânticos [1]. Nesta dissertação estuda-se dois desses algoritmos, mais precisamente, os esquemas de assinatura digital qTESLA e Crystals-Dilithium, tendo como ferramenta principal, no que concerne à implementação não otimizada dos mesmos, o *software SageMath*.

Embora o concurso da NIST seja um passo importante, é relevante que exista uma transição dos protocolos atuais para um novo modelo, integrando soluções híbridas. Nesse sentido, e tendo em vista uma melhor transição dos algoritmos clássicos, analisa-se a adaptação dos certificados à criptografia pós-quântica e faz-se a experimentação de certificados híbridos com os esquemas de assinatura já mencionados.

Este trabalho foi desenvolvido em parceria com a Universidade do Minho e a Multicert.

Palavras chave: Criptografia pós-quântica, Criptografia *lattice-based*, LWE, SIS, qTESLA, Crystals-Dilithium, Certificados híbridos

ABSTRACT

The post-quantum cryptographic algorithms have as security premise the difficulty in solving mathematical problems that are conjectured difficult in quantum computing. The interest in implementing these algorithms has been growing so that the information is protected against quantum attacks in the future. The National Institute of Standards and Technology (NIST) has currently opened a call for selection of post-quantum cryptographic algorithms [1]. In this dissertation, two of these algorithms are studied, more precisely, the digital signature schemes qTESLA and Crystals-Dilithium, having as its main tool, regarding their non-optimized implementation, the SageMath software.

Although the NIST contest is an important step, it is important that there is a transition from the current protocols to a new model, integrating hybrid solutions. In this sense, and with a view to a better transition from classical algorithms, the adaptation of certificates to post-quantum cryptography is analyzed and hybrid certificates are experimented with the signature schemes already mentioned.

This work was developed in partnership with the University of Minho and Multicert.

Palavras chave: Post-quantum cryptography, Lattice-based cryptography, LWE, SIS, qTESLA, Crystals-Dilithium, Hybrid certificates

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

SIGLAS

CA *Certificate Authority*. 63
CMA *Chosen-Message Attack*. 55
CVP *Closest Vector Problem*. 21

DIP Domínio de Ideias Principais. 5

EUF *Existential Unforgeability*. 55
EUF-CMA *Existential Unforgeability under Chosen Message Attack*. 56

IoT *Internet of Things*. 64

LWE *Learning With Errors*. 2

NIST *National Institute of Standards and Technology*. iv, v

OQS *Open Quantum Safe*. 3

PKI *Public Key Infrastructure*. 1, 63
PQC *Post-Quantum Cryptography*. 1
PQCS *Post-Quantum Cryptography Standardization*. 2

SIS *Short Integer Problem*. 2
SIVP *Shortest Independent Vector Problem*. 22
SUF *Strong Unforgeability*. 55
SUF-CMA *Strong Existential Unforgeability under Chosen Message Attack*. 56
SVP *Shortest Vector Problem*. 21

TCR Teorema Chinês dos Restos. 18
TLS *Transport Layer Security*. 63
TTN Transformação teórica numérica. 41

XOF *Extendable Output Function*. 47

1

INTRODUÇÃO

Os sistemas de criptografia de chave pública e as assinaturas digitais são usados para proteger quase todos os canais de comunicação seguros na Internet e autenticar a informação digital desses, respetivamente. Atualmente, tais sistemas criptográficos dependem de algo chamado de "*computational hardness assumption*": a conjetura de que um problema numérico teórico (como o problema da factorização de inteiros ou o problema do logaritmo discreto) não tem solução eficiente. Esse pressuposto é baseado na potência do processador dos computadores clássicos, que, por exemplo, para quebrarem um sistema criptográfico de 2084 *bits* necessitam de bilhões de anos.

Em 1997, Peter Williston Shor demonstrou que alguns algoritmos dos esquemas de chave assimétrica poderiam ser quebrados muito facilmente com um computador quântico suficientemente poderoso e capaz de resolver tanto o problema da factorização de inteiros, como o problema do logaritmo discreto.

Um computador quântico de grande porte, com suficientes *qubits* e profundidade de circuitos, poderá quebrar algoritmos de esquemas de chave assimétrica em tempo útil. Nomeadamente, a tal quebra de segurança que demoraria bilhões de anos a ser efetuada, recorrendo a um computador quântico, demorará apenas alguns meses [2].

Consequentemente, a computação quântica vem representar uma grande ameaça à criptografia e são já várias as empresas que estão a investir na segurança *Post-Quantum Cryptography* (PQC), de modo a tentar assumir o controlo sobre a situação.

Ainda não se sabe quando é que existirá um computador quântico estável capaz de derrubar as primitivas clássicas atuais. Inicialmente, um relatório interno da NIST de abril de 2016 ([3]) previa que, nos 20 anos seguintes, nenhuma das primitivas clássicas de chave pública (RSA, Diffie-Hellman, Curvas Elípticas, etc) poderiam ser consideradas seguras. No entanto, actualmente a expectativa já está nos sete a dez anos [4]. E, dadas as ameaças em potencial e aos rápidos desenvolvimentos nessa área, é bem possível que essa mudança ocorra ainda mais cedo que o previsto.

É assim urgente assimilar que se trata apenas de uma questão de tempo e não esquecer que a indústria PKI actual levou quase duas décadas para se desenvolver. Portanto, independentemente da estimativa para o tempo exato da chegada da era da computação quântica, deve-se começar agora a preparar os sistemas de segurança da informação para serem capazes de resistir a ataques quânticos. Segundo a NIST, "é improvável que haja uma substituição simples para os algoritmos criptográficos de chave pública atuais. Um esforço

significativo será necessário para desenvolver, padronizar e implantar novos criptossistemas pós-quânticos"[5].

Os avanços na construção de computadores quânticos e o facto de vários *standards* atuais em criptografia serem inseguros, na presença de um eventual computador quântico de maior porte, levaram a que a NIST promovesse um concurso de padronização de algoritmos pós-quânticos. O concurso encontra-se ainda em andamento, mais concretamente já se encontra na terceira ronda.

Das candidaturas que fazem parte do concurso PQCS da NIST, destacam-se nesta dissertação, os esquemas de assinatura digital que procuram ser resistentes a ataques quânticos e assim substituir os algoritmos utilizados até à data. Mais precisamente, estuda-se dois desses esquemas de assinatura digital pós-quânticos, o **qTESLA** e o **Crystals-Dilithium**.

O objetivo essencial deste trabalho é, de modo geral, a aprendizagem e consolidação dos fundamentos desses dois esquemas de assinatura digital, dando especial importância à componente matemática dos mesmos. Tenciona-se também, numa vertente mais prática, estimular as aptidões computacionais necessárias para desenvolver uma implementação não otimizada de ambos os esquemas de assinatura digital, no *software SageMath*. Adicionalmente, tendo em vista a transição dos protocolos atuais para um novo modelo, integrando soluções híbridas, pretende-se efectuar o estudo da conexão entre certificados e criptografia pós-quântica e fazer, além disso, a experimentação de certificados híbridos com os esquemas de assinatura já mencionados.

Na primeira parte desta dissertação, faz-se introdução dos fundamentos, conceitos e resultados pertinentes à contextualização matemática do Capítulo 5. Nesse capítulo faz-se a descrição com algum detalhe dos problemas que fundamentam os esquemas de assinaturas já referidos, o problema LWE e o problema SIS.

A segunda parte é subdividida em três capítulos: no primeiro faz-se a descrição do esquema de assinatura digital qTESLA, no segundo faz-se o mesmo mas para o Crystals-Dilithium e, no último, faz-se a comparação direta entre estes dois esquemas, tendo como critérios de comparação essencialmente a segurança, o desempenho e o tamanho da chave pública e da assinatura.

Na terceira e última parte estuda-se, brevemente, a adaptação dos certificados à criptografia pós-quântica e, como aplicação, efetua-se a experimentação de certificados híbridos com os esquemas de assinatura pós-quânticos já mencionados.

Com esta dissertação, mais precisamente, com a comparação feita entre o qTESLA e o Crystals-Dilithium, constatou-se que o Crystals-Dilithium tinha, de modo geral, um melhor desempenho, o que está em concordância com o facto de o mesmo ter prevalecido, ao invés do qTESLA, na terceira ronda do concurso da NIST.

Quanto à bibliografia consultada, a primeira parte desta dissertação, é baseada em diversas fontes bibliográficas. O capítulo 11 do livro de Lawrence C. Washington [10] juntamente com os apontamentos sobre corpos e teoria de Galois de James Milne [11] fundamentam o capítulo 2. O capítulo "Lattice-based Cryptography" do livro de Daniel Bernstein, Johannes Buchmann e Erik Dahmen [1], conduziu, de modo geral, os capítulos 3 e 4. Mais, o livro [8] e o artigo [9] de Chris Peikert fundamenta o capítulo 5 inerente aos

problemas LWE e SIS. Quanto às implementações em *SageMath*, que se encontram em anexo, foram baseadas nas documentações oficiais disponibilizadas pelo NIST, referentes à segunda ronda do concurso, do qTESLA e do Crystals-Dilithium (ver [6] e [7]). A comparação do desempenho feita entres estes esquemas de assinatura digital, no capítulo 6, tem como base as implementações oficiais destes, disponíveis em linguagem **C**, nas seguintes páginas de *github*: `https://github.com/qtesla/qTesla` e `https://github.com/pq-crystals/dilithium`. Para finalizar, a secção inerente à experimentação dos certificados híbridos com os esquemas de assinatura estudados, no capítulo 7, foi fundamentada no projeto *Open Quantum Safe* (OQS), mais precisamente na página do *github*: `https://github.com/open-quantum-safe/openssl`.

2

PRELIMINARES

O objetivo deste capítulo é fornecer uma apresentação resumida de alguns dos fundamentos matemáticos sobre os quais esta dissertação se baseia. Aqui será, por exemplo, apresentado o conceito de 'polinómio ciclotómico', tendo em consideração as aplicações criptográficas do mesmo.

Consequentemente este capítulo acaba por ser uma breve introdução matemática útil para a compreensão dos próximos capítulos.

Assume-se ainda que o leitor possui alguns conhecimentos essenciais de Álgebra, para além dos aqui apresentados.

Adicionalmente aproveita-se a oportunidade para introduzir alguma notação e terminologia padrão.

2.1 ANÉIS DE POLINÓMIOS

Os vectores são provavelmente a forma mais habitual de agrupar informação, tendo como exemplo mais simples de agregação as *strings* de *bits* de tamanho fixo n.

Note-se que $GF(2), \mathbb{F}_2, \mathbb{Z}_2$ são três formas equivalentes de designar o mesmo corpo, o corpo de *Galois* com 2 elementos, e que uma *string* de n *bits*, ou *array* de n *bits*, é um elemento da estrutura $\mathbb{F}_2^n$, ou $GF(2)^n$, ou $(\mathbb{Z}_2)^n$. Recorde que $\mathbb{F}^n = \mathbb{F} \times \mathbb{F} \times \cdots \times \mathbb{F}$ (n vezes). Tal estrutura é um espaço vectorial com as operações definidas da forma usual, e, considerando a simplificação da notação $(a_1, a_2, \cdots, a_n)$ por $a_1 a_2 \cdots a_n$, é possível efectuar, por exemplo, a soma de vectores componente a componente:

$$10110 + 11000 = 01110.$$

Por simplificação, pode-se associar às componentes de cada vector o significado de serem coeficientes de um polinómio. Por exemplo, considerando a estrutura algébrica $\mathbb{Z}_2^5$ como espaço vetorial, pode-se associar 10110 ao polinómio $1 + 0 \cdot x + x^2 + x^3 + 0 \cdot x^4$, de variável x. E, assim, usando esta interpretação, os dois vectores acima representam os polinómios $1 + x^2 + x^3$ e $1 + x$, respectivamente.

Sendo R um anel, representa-se por

$$R[x] = \{p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n | a_i \in R\}$$

o **anel dos polinómios de coeficientes em R e variável** x, cujas operações são a soma e a multiplicação de polinómios na sua forma usual.

Seguem-se agora algumas observações relevantes:

- Quando o anel R é ele próprio um anel de polinómios sobre outra variável y e coeficientes num outro anel R' (i.e. se $R = R'[y]$) então $R[x] = R'[y][x]$ escreve-se também como $R'[y, x]$.

- Surgem propriedades interessantes quando R é o anel dos inteiros e constrói-se $\mathbb{Z}[x]$ ou quando R é um corpo finito $R = GF(q^d)$, sendo q um primo (a característica) e $d > 0$ a dimensão. Note-se que d diz-se a dimensão porque $GF(q^d)$ é um espaço vectorial sobre $GF(q)$ de dimensão d e com base $\{1, q, q^2, \cdots, q^{d-1}\}$.

  Sabe-se ainda que quando R é um corpo, $R[x]$ é um Domínio de Ideais Principais (DIP) e que num DIP está definida a noção de maior divisor comum (gcd) e o algoritmo de Euclides funciona para o cálculo do gcd e das inversas multiplicativas.
  Note-se também que para polinómios a duas variáveis não é possível definir gcd nem calcular inversas dessa forma.

Nas estruturas atrás definidas, se R for finito ou contável, a estrutura $R[x]$ é sempre contável. Por exemplo, sabendo que $\mathbb{Z}$ é um conjunto infinitamente contável temos que $\mathbb{Z}[x]$ é também contável. Mais, em aplicações criptográficas exige-se, por norma, estruturas finitas, o que requer restringir o anel dos coeficientes dos polinómios.

Observe que num corpo os elementos são identificados por polinómios de grau inferior à dimensão d do corpo onde a soma e a multiplicação são realizadas módulo um polinómio irredutível de grau d, designado por **polinómio característico** do corpo.

Nos anéis de polinómios, caso o polinómio característico seja irredutível e os coeficientes do polinómio pertencerem a um corpo, então esse tal anel de polinómios é um corpo, isto é, todo o polinómio, excepto o polinómio nulo, é invertível.

Num anel $R[x]$, onde R é um DIP, um polinómio $p(x)$ é invertível módulo um polinómio característico $m(x)$ se e só se:

$$\gcd(p(x), m(x)) = 1.$$

Uma estrutura de polinómios que usa $m(x)$ de grau n e realiza as operações usuais módulo $m(x)$ é efectivamente limitada ao grau n. Com tais operações define-se então o chamado **anel quociente** que se representa por

$$R[x]/\langle m(x)\rangle.$$

Um anel quociente não é obrigatoriamente finito. Sendo R um DIP, o anel $R[x]/\langle m(x)\rangle$ é finito quando R é um corpo finito. Neste caso, o anel quociente $R[x]/\langle m(x)\rangle$ será um corpo se e só se $m(x) \in R[x]$ for irredutível.

Observações:

1. Uma vez que $\mathbb{Z}$ não é finito, o facto do polinómio $m(x) = x^8 + 1$ ser irredutível em $\mathbb{Z}$, não implica necessariamente que o anel quociente $\mathbb{Z}[x]/\langle x^8 + 1\rangle$ seja um corpo.

2. Algumas das técnicas criptográficas sobre polinómios usam uma estrutura deste tipo com a restrição de os coeficientes só poderem tomar valores $\{-1, 0, 1\}$.
   O conjunto dos vectores $\{-1, 0, 1\}^n$ pode ser associado a um subconjunto de um anel $\mathbb{Z}[x]/\langle m(x)\rangle$, sendo n o grau de $m(x)$. No entanto, esta estrutura não sendo fechada nem para a soma nem para multiplicação, não constitui um anel.

3. Ao definir o anel quociente como $\mathbb{Z}_p[x]/\langle m(x)\rangle$ obtém-se um anel finito que é um corpo (com p^n elementos) se e só se $m(x)$ for irredutível.

4. Quando não é necessário obter um corpo é usual usar dois tipos de polinómios característicos:

   - Polinómios da forma $m(x) = x^n - 1$, quando n é primo.
   - Polinómios da forma $m(x) = x^n + 1$, quando $n = 2^d$ para um inteiro d aleatório.

   A escolha destes polinómios tem como principal objetivo tornar eficiente a multiplicação de polinómios, uma vez que é nessas operações onde se concentra grande parte da carga computacional que as técnicas criptográficas desta família têm de suportar.

2.2 OPERAÇÕES EM ANÉIS DE POLINÓMIOS

Sendo a multiplicação a operação computacional mais pesada em técnicas criptográficas que recorrem a polinómios, é importante procurar soluções o mais eficazes possível.

Repare que o algoritmo directo da multiplicação provém das propriedades distributivas da multiplicação em relação à soma. Considere agora o anel dos polinómios de coeficientes inteiros, $\mathbb{Z}[x]$, sem qualquer limite relativamente ao grau dos polinómios.

Sejam

$$a = \sum_{i=0}^{deg(a)} a_i x^i \quad \text{e} \quad b = \sum_{j=0}^{deg(b)} b_j x^j,$$

não nulos e $deg(a)$ e $deg(b)$ os graus dos polinómios a e b, respectivamente.

A multiplicação $c = a \cdot b$ tem grau $deg(c) = deg(a) + deg(b)$ e o coeficiente c_k do monómio $c_k x^k$ calcula-se facilmente:

$$c_k = \sum_{i+j=k} a_i b_j,$$

com $k \leq deg(a) + deg(b)$.

É essencial ter em consideração que em muitas aplicações criptográficas o grau dos polinómios é da ordem de muitas centenas, eventualmente até milhares. Deste modo é

usual associar à sequência de coeficientes de um polinómio a analogia de um sinal discreto. Com tal comparação, a multiplicação adoptada é precisamente a mesma que se usa para calcular a chamada convolução de dois sinais, designa por convolução de polinómios.

Considere-se agora que a multiplicação é efetuada módulo o polinómio característico $m(x) = x^n - 1$, e que

$$a = \sum_{i=0}^{n-1} a_i x^i \quad \text{e} \quad b = \sum_{j=0}^{n-1} b_j x^j.$$

O produto:

$$c = a \cdot b = a(x) \cdot b(x) \bmod (x^n - 1)$$

será então determinado por uma sequência de coeficientes c_k, com $k = 0, \ldots, n-1$, tal que

$$c_k = \sum_{i+j=k \bmod n} a_i b_j.$$

Note-se que $x^n = x^0 = 1 \bmod (x^n - 1)$ e $x^k = x^{(k \bmod n)} \bmod (x^n - 1)$. Portanto, com o módulo $x^n - 1$, a convolução de polinómios é do mesmo modo simples de calcular e a relação anterior pode-se reescrever da seguinte forma:

$$c_k = \sum_{i=0}^{n-1} a_i b_{k-i} \bmod n.$$

Contudo, o número de operações de multiplicação de coeficientes é da ordem de $O(n^2)$ operações.

Consequentemente, quando n é grande, da ordem dos milhares, compensa considerar um algoritmo mais complexo mas com melhor eficiência computacional, por exemplo, um algoritmo baseado na chamada "*Fast Fourier Transform*" e de complexidade $O(n\ log\ n)$.

Note-se que quando o módulo é $m(x) = x^n - 1$ existe uma relação importante entre a convolução de polinómios e a multiplicação de matrizes.

Considere o seguinte polinómio:

$$a(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

e sejam os coeficientes deste os elementos da primeira linha de uma matriz $n \times n$.

Tendo em mente a composição da matriz gerada pelos coeficientes do polinómio $a(x)$, na segunda linha coloca-se os coeficientes do polinómio $x \cdot a(x)\ mod\ (x^n - 1)$. Repare-se que $x \cdot a(x)$ é o polinómio

$$a_0 x + a_1 x^2 + \cdots + a_{n-2} x^{n-1} + a_{n-1} x^n$$

e, como $x^n = 1$, o coeficiente a_{n-1} passa a ser o coeficiente da potência x^0, sendo agora a primeira entrada da segunda linha. Mais, multiplicar por x é desviar todos os coeficientes uma posição para a direita e fazer com que o último "rode" para ocupar a primeira posição.

Analogamente, com esta operação de "*shift-rotate*", constrói-se as restantes linhas da matriz que correspondem aos coeficientes dos polinómios $x^2a(x), x^3a(x)$ e $x^4a(x)$.

Denota-se por $[a(x)]$ o vetor linha cujos elementos são os coeficientes, por ordem crescente de grau, do polinómio $a(x)$. Assim, constrói-se a matriz A da seguinte forma:

$$A = \begin{bmatrix} [a(x)] \\ [x \cdot a(x)] \\ [x^2 \cdot a(x)] \\ \cdots \\ [x^{n-1} \cdot a(x)] \end{bmatrix}$$

A título de exemplo admita agora que $n = 5$ e que $a(x) = 3 - x + x^3 + 3x^4$. A matriz gerada pelos coeficientes deste polinómio $a(x)$ é:

$$\begin{bmatrix} 3 & -1 & 0 & 1 & 3 \\ 3 & 3 & -1 & 0 & 1 \\ 1 & 3 & 3 & -1 & 0 \\ 0 & 1 & 3 & 3 & -1 \\ -1 & 0 & 1 & 3 & 3 \end{bmatrix}$$

Deste modo, para efectuar a convolução $b(x) \cdot a(x) \bmod (x^n - 1)$, comecemos por considerar a seguinte multiplicação modular:

$$b(x) \cdot a(x) = \left(\sum_{i=0}^{n-1} b_i x^i \right) \cdot a(x) = \sum_{i=0}^{n-1} b_i \cdot (a(x) \cdot x^i).$$

Como $[a(x) \cdot x^i]$ são as linhas da matriz A, o vector de coeficientes da convolução é uma combinação linear, com coeficientes b_i, das linhas da matriz A. Isto é, $[b(x) \cdot a(x)]$ obtém-se por multiplicação de matrizes:

$$[b(x) \cdot a(x)] = [b(x)] \cdot A.$$

Deste modo, reduz-se o cálculo da convolução à multiplicação entre um vector linha e uma matriz quadrada A, que é construída de forma muito eficiente.

Exemplo: Sejam:

$$a(x) = 3 - x + x^3 + 3x^4 \text{ e } b(x) = x + 2x^2 - x^4.$$

Observe-se que a matriz A gerada por $a(x)$ já foi calculada atrás e que o polinómio $b(x)$ é determinado pelo vector $[0, 1, 2, 0, -1]$. Portanto a convolução é calculada como se segue:

$$[0 \;\; 1 \;\; 2 \;\; 0 \;\; -1] \cdot \begin{bmatrix} 3 & -1 & 0 & 1 & 3 \\ 3 & 3 & -1 & 0 & 1 \\ 1 & 3 & 3 & -1 & 0 \\ 0 & 1 & 3 & 3 & -1 \\ -1 & 0 & 1 & 3 & 3 \end{bmatrix} = [6 \;\; 9 \;\; 4 \;\; -5 \;\; -2].$$

Assim $b(x) \cdot a(x)\ mod\ (x^5 - 1) = 6 + 9x + 4x^2 - 5x^3 - 2x^4$.

Note-se que a convolução de polinómios também é possível com o módulo $(x^N + 1)$, sendo que a regra "*right-shift-rotate*" usada no módulo $(x^N - 1)$ passa a ser "*right-shift-change.sign-rotate*" e consiste em trocar o sinal do último elemento de cada linha antes de este vir a ocupar a primeira posição da linha seguinte.

É claro que este mecanismo aqui apresentado para polinómios com coeficientes em $\mathbb{Z}$, pode ser estendido para qualquer outro anel finito $\mathbb{Z}_q$.

Em criptografia são muitas as técnicas que usam polinómios e recorrem às propriedades daquilo que se pode designar por "duplo módulo", em que um $a(x) \in \mathbb{Z}[x]$ é visto como uma construção da forma

$$(a(x) \bmod q) \bmod p,$$

em que q e p são inteiros primos entre si, i.e.

$$mdc(q, p) = 1.$$

Nesses casos não é benéfico representar tais polinómios como elementos de $\mathbb{Z}_q[x]$ ou $\mathbb{Z}_p[x]$, pois o mesmo polinómio pode ser visto tanto num anel como no outro. Deste modo é útil considerá-lo apenas como elemento de $\mathbb{Z}[x]$ e admitir que operações de módulo são operações aplicadas a cada um dos seus coeficientes.

É essencial ainda ter em conta o seguinte anel de referência:

$$\mathbb{Z}[x]/\langle x^N - 1\rangle$$

que, por simplificação de notação, será aqui denotado somente por R_N.

Do mesmo modo que é vantajoso trabalhar com polinómios de coeficientes em $\mathbb{Z}$, o mesmo se aplica quando se considera o módulo:

$$m(x) = x^{2^n} + 1,$$

sendo $n > 1$, porque estes polinómios são sempre irredutíveis em $\mathbb{Z}$.

Provemos que de facto o polinómio $m(x) = x^{2^n} + 1$, com $n \geq 1$, é irredutível em $\mathbb{Z}$.

Começa-se então por recordar o lema que se segue:

Lema de Gauss:

Seja $f \in \mathbb{Z}[x]$ um polinómio de grau ≥ 1. Então f é irredutível em $\mathbb{Z}[x]$ se e só se f for primitivo sobre $\mathbb{Z}$ e irredutível em $\mathbb{Q}[x]$.[11]

Antes de mais, recorde-se que um polinómio primitivo sobre um anel é um polinómio com coeficientes coprimos.

Segundo este lema, para provar então que $m(x) = x^{2^n} + 1$ é irredutível em $\mathbb{Z}[x]$, basta provar que:

1. m é um polinómio primitivo sobre $\mathbb{Z}$
2. $m(x)$ é irredutível em $\mathbb{Q}[x]$.

Prova de 1.

Temos que

$$m(x) = a_{2^n} \cdot x^{2^n} + a_0,$$

com $a_0 = 1$ e $a_{2^n} = 1$.

Como $\gcd(a_0, a_{2^n}) = \gcd(1, 1) = 1$, conclui-se então que $m(x)$ é polinómio primitivo sobre $\mathbb{Z}$.

Prova de 2.

Comece por recordar o seguinte critério:

Critério de Eisenstein:
Considere o seguinte polinómio com coeficientes inteiros:

$$g(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0.$$

Se existir um número primo p que satisfaça as três condições que se seguem:

- p não divide a_n
- p^2 não divide a_0
- p divide cada a_i, para $i \neq n$

então, o polinómio $g(x)$ é irredutível sobre os números racionais.[11]

Tendo em mente o critério acima, considere a seguinte substituição: $y = x - 1$, tal que:

$$x^{2^n} + 1 = (y + 1)^{2^n} + 1.$$

E, pelo Binómio de Newton,

$$\begin{aligned}(y+1)^{2^n}+1 &= \sum_{k=0}^{2^n}\binom{2^n}{k}y^{2^n-k}+1\\ &= \binom{2^n}{0}y^{2^n}+\binom{2^n}{1}y^{2^n-1}+\cdots+\binom{2^n}{2^n-1}y+\binom{2^n}{2^n}y^0+1\\ &= y^{2^n}+\binom{2^n}{1}y^{2^n-1}+\cdots+\binom{2^n}{2^n-1}y+2\end{aligned}$$

Observe-se então que para $p=2$ temos que:

- p não divide $a_{2^n}=1$
- $p^2=4$ não divide $a_0=2$

Falta então apenas mostrar que p divide cada a_i, com $i\neq 2^n$, ou seja, que $\binom{2^n}{j}$, para $1\leq j\leq 2^n-1$, é divisível por 2. Noutras palavras, falta apenas mostrar que $\binom{2^n}{j}$ é par, para $j=1,2,\ldots,2^n-1$. Por conseguinte, tal é facilmente comprovado, pois:

$$(y+1)^2\equiv y^2+1 \pmod 2.$$

E, por indução em n, é trivial que:

$$(y+1)^{2^n}\equiv y^{2^n}+1 \pmod 2.$$

E tal, por sua vez, permite concluir que a última condição do critério de Eisenstein é satisfeita e finalizar assim a prova de que $m(x)=x^{2^n}+1$, com $n\geq 1$, é irredutível em $\mathbb{Z}$.

Na verdade estes polinómios em estudo são um caso particular dos chamados **polinómios ciclotómicos**. Nomeadamente, o polinómio:

$$x^{2^{d-1}}+1,$$

com $d\geq 1$, é o polinómio ciclotómico de ordem 2^d e cuja propriedade fulcral é precisamente o facto de ser irredutível.

Tal irredutibilidade do polinómio x^N+1, com $N=2^{d-1}$ e $d\geq 1$, fornece as seguintes propriedades algébricas ao anel $C_N=\mathbb{Z}[x]/\langle x^N+1\rangle$:

- O anel C_N é um DIP [10][12] e, por conseguinte, qualquer ideal $I\subseteq C_N$ é formado por múltiplos de um determinado gerador $g\in I$, tal que:
$$I=\{rg|r\in C_N\}.$$
- Todo o ideal é um $\mathbb{Z}-$módulo de dimensão N. Noutras palavras, ao escolher um ideal $I\subseteq C_N$ este pode ser sempre gerado por combinações lineares inteiras de N elementos em C_N, ou seja, existe $b_1,b_2,\ldots,b_N\in C_N$, tal que:
$$I=\{n_1b_1+n_2b_2+\cdots+n_Nb_N|\ n_i\in\mathbb{Z}\}.$$

Estes $\mathbb{Z}-$módulos, também conhecidos como **Reticulados**, ou "***Lattices***" em inglês, são uma das estruturas algébricas mais relevantes na criptografia moderna e no Capítulo 4 serão estudados com algum detalhe.

3

CRIPTOSSISTEMAS EM ANÉIS CICLOTÓMICOS

Consideremos os anéis de polinómios da forma:

$$C_N = \mathbb{Z}[x]/\langle x^N + 1\rangle,$$

em que N é uma potência de 2. Se $N = 2^{d-1}$, com $d \geq 1$, então o polinómio $\alpha^N + 1$ diz-se um polinómio ciclotómico de ordem $2^d = 2 \cdot N$.

Como já mencionado no fim do capítulo anterior, este anel quociente possui as seguintes propriedades essenciais:

- Todo o ideal de C_N é principal.
  Portanto, sempre que se verificam as propriedades responsáveis pela definição de ideal numa determinada estrutura, essa mesma estrutura coincide com o conjunto dos múltiplos de um determinado gerador.
- Todo o ideal de C_N é um $\mathbb{Z}$-módulo (ou grupo abeliano aditivo) de dimensão N.
  Tal significa que todo o elemento b de um ideal $I \subseteq C_N$ é uma combinação linear inteira:
  $$b = n_1 b_1 + n_2 b_2 + \cdots + n_N b_N, n_i \in \mathbb{Z}$$
  de N elementos de I linearmente independentes. Ao conjunto $B = \{b_1, b_2, \ldots, b_N\}$ chamamos base do ideal I.

Estas duas propriedades acabam por ser duas representações alternativas do mesmo ideal: uma como estrutura das multiplicações de um gerador e outra como estrutura das combinações lineares inteiras de uma base com N elementos.

A partir de um gerador $g \in I$, sendo $I \subseteq C_N$ um ideal, é simples calcular uma base de I, bastando considerar os polinómios:

$$\{g, \alpha g, \ldots, \alpha^{N-1} g\}.$$

Na prática, qualquer combinação linear inteira desta base, com $r_i \in \mathbb{Z}$, tem a forma:

$$\begin{aligned} r_0 g + r_1 \alpha g + \cdots + r_{N-1} \alpha^{N-1} g, &= (r_0 + r_1 \alpha + \cdots + r_{N-1} \alpha^{N-1}) \cdot g \\ &= r \cdot g, \end{aligned}$$

sendo r o polinómio $r_0 + r_1\alpha + \cdots + r_{N-1}\alpha^{N-1}$.

Adicionalmente existe um algoritmo de complexidade polinomial que efectua o processo inverso, isto é, a partir de uma base $B \subseteq C_N$ de dimensão N, determina se a mesma gera um ideal em C_N e, se tal acontece, determina o gerador desse ideal. (Para mais detalhes sobre esse algoritmo, consultar o artigo [13]).

As aplicações na criptografia destas estruturas algébricas consistem numa abordagem assente no Problema "***Learning with errors***" (LWE), na variante designada por Ring-LWE, e que será apresentada em seguida.

3.1 LEARNING WITH ERRORS SOBRE ANÉIS

No Capítulo 4 será estudado, com algum pormenor, a forma geral do problema LWE e respetivas variantes. Nesta secção descreve-se a forma particular deste problema definido em anéis ciclotómicos

$$C_N = \mathbb{Z}[x]/\langle x^N + 1\rangle, \text{ com } N = 2^{d-1},$$

ou na sua forma modular

$$C_{q,N} = \mathbb{Z}_q[x]/\langle x^N + 1\rangle,$$

sendo q um primo suficientemente grande que verifica $q \equiv 1 \bmod 2N$. Observe-se que $C_{q,N}$ é um corpo se e só se $x^N + 1$ for irredutível em $\mathbb{Z}_q$.

Para um segredo $s \in C_{q,N}$, o problema LWE em tais anéis, designado por **R-LWE**, analisa a distribuição dos pares

$$(a, b) \in C_{q,N} \times C_{q,N}$$

em que:

- os valores dos coeficientes de a são gerados aleatoriamente através de uma distribuição uniforme;
- para cada par (a, b) a componente b verifica

  $$b \equiv s \cdot a + e \bmod x^N + 1$$

  sendo e um polinómio erro gerado aleatoriamente por uma distribuição não-uniforme. Note-se que sendo a geração destes erros não-uniforme esta é por conseguinte distinta da geração dos polinómios a.

Ao gerar vários valores de a aleatoriamente constrói-se uma sequência aleatória de vetores $a_i \in \mathbb{Z}_q^N$, interpretados como polinómios definidos pelo vector dos seus coeficientes.

Antes de mais, temos de estudar a formação dos erros e_i em pormenor. Tais erros são gerados por um algoritmo G_α, responsável por gerar inteiros num determinado intervalo I_q, sendo:

$$I_q = \left[-\left\lfloor \frac{q}{2} \right\rfloor, \left\lfloor \frac{q}{2} \right\rfloor\right] \cap \mathbb{Z},$$

e, assumindo que q é ímpar e que se segue uma distribuição $\mathcal{X}_\alpha$ que se aproxima da distribuição gaussiana de média $\mu = 0$ e desvio padrão $\sigma = \alpha q$.

Para valores de α muito inferiores a $1/2$, o algoritmo G_α gera com grande probabilidade números pertencentes ao intervalo $[-2\alpha q, 2\alpha q]$. Esta distribuição contrasta com a geração uniforme que distribui inteiros com igual probabilidade em todo o intervalo I_q.

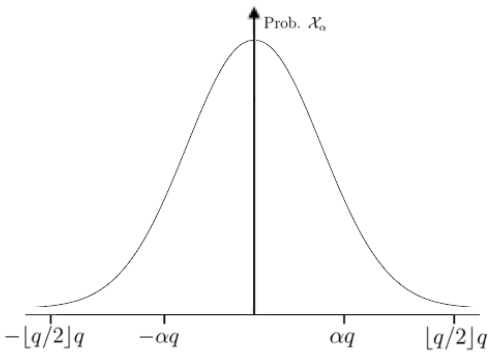

Figura 1: Função densidade da distribuição Gaussiana de média $\mu = 0$ e desvio padrão $\sigma = \alpha q$.

Tem-se então:

$$b_i = s \cdot a_i + e_i \bmod x^N + 1,$$

com $e_i \leftarrow G_\alpha^N$, i.e, sendo e_i o *output* do algoritmo G_α.

O oráculo $RLWE_\alpha(s)$ gera os pares $(a_i, b_i) \in \mathbb{Z}_q^N \times \mathbb{Z}_q^N$ e consultando este oráculo define-se dois problemas:

- LWE Decisional: Consultando o oráculo um número finito de vezes, conseguir diferenciar o seu '*output*' do *output* de um gerador uniforme em I_q^{2N}.
- LWE Computacional: Consultando o oráculo um número finito de vezes, conseguir computar s.

Sabe-se que estes problemas são difíceis, mesmo em modelos de computação quântica. De facto, a complexidade média de cada um destes problemas, para valores de s, gerados segundo uma distribuição uniforme, não é menor que a complexidade, no pior caso, do problema de encontrar vectores curtos no reticulado definido por s. (Ver Capítulo 4)

Para compreender melhor como é que estes problemas originam criptossistemas considere uma variante muito simples, quando $N = 1$. Neste caso os polinómios s, a, b, e reduzem-se a elementos de $\mathbb{Z}_q$.

Construindo uma sequência de inteiros $a_1, a_2, \ldots, a_k$ gerados uniformemente no intervalo I_q, os produtos:

$$s \cdot a_1, s \cdot a_2, \ldots, s \cdot a_k, \bmod x + 1$$

são uniformemente distribuídos no intervalo, excepto quando $s = 0$; nesse caso os produtos seriam todos constantes e nulos.

Efectuando a soma dos erros e_i, gerados pelo algoritmo G_α, e seguindo a distribuição $\mathcal{X}_\alpha$, as somas obtidas continuam a ser elementos de I_q uniformemente distribuídos.

Deste modo, com esta informação, constrói-se então uma cifra simétrica, que usa a chave s e cifra um único *bit* $\beta \in \{0, 1\}$. Segue-se a descrição da mesma quanto à cifragem, decifragem e segurança.

Cifragem:

1. Gerar uma sequência $a_1, a_2, \ldots, a_k$ de elementos de I_q uniformemente distribuídos.
2. Calcular
$$b_i = \begin{cases} s \cdot a_i + e_i, \ com\ e_i \leftarrow G_\alpha, \ se\ \beta = 0 \\ s \cdot a_i + \left\lfloor \frac{q}{2} \right\rfloor, \ se\ \beta = 1 \end{cases}$$
3. Enviar a sequência de pares (a_i, b_i).

Decifragem:

1. Com a chave s recuperar a diferença $d_i = (b_i - s \cdot a_i)$.
2. Verificar se os d_i são uniformemente iguais a $\lfloor q/2 \rfloor$ ou se se distribuem segundo a distribuição gaussiana. No primeiro caso produzir o *output* 1 e no segundo o *output* 0.

Segurança:

Suponhamos que o atacante não conhece s mas sim uma chave diferente s'. Então a diferença que ele calcula é $d'_i = (b_i - s' \cdot a_i) = d_i + (s - s') \cdot a_i$.

Como $s - s' \neq 0$ e os a_i são uniformemente distribuídos, qualquer que seja a distribuição d_i, o atacante obtém a distribuição d'_i, que é sempre uniforme.

3.2 CRIPTOSSISTEMA LWE DE CHAVE PÚBLICA

O criptossistema apresentado anteriormente não é muito prático pois apenas cifra mensagens com 1 *bit* e para tal usa uma sequência de k inteiros d_i e outra de inteiros b_i. Já um sistema de chave pública que cifre mensagens β com tamanho N seria certamente mais útil. A construção básica desse sistema é, contudo, semelhante à cifra anterior, ou seja, consiste em perturbar o erro com a mensagem β, multiplicada por um inteiro grande $\left\lfloor \frac{q}{2} \right\rfloor$.

3.2.1 *Geração das chaves*

1. Escolher $N = 2^d$ e um primo q suficientemente grande.
2. A chave privada é o polinómio:
$$s \in C_{q,N},$$
com coeficientes gerados aleatoriamente em $\{-1, 0, 1\}$.
3. A chave pública é o par $(a, b) \in C^2_{q,N}$, sendo a gerado aleatoriamente segundo uma distribuição uniforme e com coeficientes em I^N_q, e b definido como se segue:
$$b \equiv s \cdot a + e_0 \bmod \alpha^N + 1,$$
com $e_0 \leftarrow G^N_\alpha$.

3.2.2 *Cifragem*

Suponhamos que se pretende cifrar $\beta \in \{0, 1\}^N$.

1. Constrói-se um polinómio r de variável x e de grau N e cujos coeficientes são gerados aleatoriamente em $\{-1, 0, 1\}^N$.
2. Gera-se $e_1, e_2 \leftarrow G^N_\alpha$.
3. Calcula-se o par $(u, v) \in C^2_{q,N}$ tal que:
$$u \equiv a \cdot r + e_1 \bmod x^N + 1$$
$$v \equiv b \cdot r + e_2 + \beta \cdot \left\lfloor \frac{q}{2} \right\rfloor \bmod x^N + 1$$
4. O criptograma é o par (u, v).

3.2.3 *Decifragem*

Conhece-se a chave privada s e o criptograma (u, v).

1. Calcular $(v - u \cdot s)$, arredondando cada um dos coeficientes deste polinómio para 0 ou 1, consoante se o inteiro mais próximo for 0 ou $\left\lfloor \frac{q}{2} \right\rfloor$.

3.2.4 *Justificação*

A justificação é simples de efectuar e consiste em calcular o seguinte:

$$\begin{aligned} v - u \cdot s &= \left(b \cdot r + e_2 + \beta \cdot \left\lfloor \frac{q}{2} \right\rfloor\right) - (a \cdot r + e_1) \cdot s \bmod x^N + 1 \\ &= b \cdot r + e_2 + \beta \cdot \left\lfloor \frac{q}{2} \right\rfloor - a \cdot r \cdot s - e_1 \cdot s \bmod x^N + 1 \\ &= (r \cdot e_0 - s \cdot e_1 + e_2) + \left\lfloor \frac{q}{2} \right\rfloor \cdot \beta \bmod x^N + 1. \end{aligned}$$

Seja $\epsilon = r \cdot e_0 - s \cdot e_1 + e_2$ e observe-se que este é pequeno porque é a soma de múltiplos de polinómios gerados por G_α^N. Se α for suficientemente pequeno, então, com grande probabilidade, e_0, e_1, e_2 são também pequenos e, como r e s têm coeficientes em $\{-1, 0, 1\}$, o resultado será pequeno.

Portanto, no arredondamento para 0 ou $\left\lfloor \frac{q}{2} \right\rfloor$, o *bit* β_i dita o resultado final.

Note que recuperar (s, e) a partir da chave pública (a, b) é equivalente a resolver o problema computacional LWE.

3.3 MULTIPLICAÇÃO EFICIENTE

Considere nesta secção polinómios definidos na variável w.

No que concerne à eficiência computacional, a operação crítica neste tipo de criptossistemas é a multiplicação de polinómios. Em oposição ao que acontece nos anéis ciclotómicos, onde um dos operandos é fixo e pode ser substituído pela sua matriz circulante, aqui normalmente ambos os operandos são variáveis.

No entanto pode-se, por norma, aproveitar o facto de o polinómio $(w^N + 1)$ ser ciclotómico para representar um polinómio genérico:

$$f(w) \in \mathbb{F}_q[w]/\langle w^N + 1 \rangle.$$

Note-se que:

$$(w^{2N} - 1) = (w^N - 1) \cdot (w^N + 1).$$

Sabe-se que existe um valor α numa extensão do corpo $\mathbb{F}_q$ (um corpo $\mathbb{F}_{q^k}$, para algum k) tal que:

- todas as raízes do polinómio $(w^{2N} - 1)$ são da forma $\alpha^i, i \in \{0, 1, \ldots, 2N - 1\}$.
- ao selecionar os índices pares, $i = 2j$, constrói-se o conjunto de raízes de $w^N - 1$.
- selecionando os índices ímpares, $i = 2j + 1$, constrói-se o conjunto de raízes de $w^N + 1$.

Portanto, definindo $z_i = \alpha^{2i+1}, i \in \{0, 1, \ldots, 2N - 1\}$ constrói-se o conjunto das raízes de $(w^N + 1)$ em algum $\mathbb{F}_{q^k}$:

$$(w^N + 1) = (w - z_0) \cdot (w - z_1) \cdot \ \ldots \ \cdot (w - z_{N-1}).$$

Como α é o gerador de um grupo cíclico com $2N$ elementos, o corpo $\mathbb{F}_{q^k}$ tem $q^k - 1$ elementos invertíveis que formam um grupo cíclico e, por isso, $q^k - 1$ tem de ter $2N$ como divisor.

Por conseguinte, k é o menor inteiro tal que $q^k = 1 \bmod 2N$ e pode-se usar o TCR para, a partir dos valores z_i construir os polinómios $e_i(w)$ que formam a base gerada pelos monómios $(w - z_i)$.

Ou seja, os erros $e_i \in \mathbb{F}_{q^k}[w]/\langle w^N + 1\rangle$ verificam:

$$e_i(z_i) = 1 \;\text{ e }\; e_i(z_j) = 0, \;\text{ para }\; i \neq j.$$

Desta forma:

- Dado um polinómio $f(w) \in \mathbb{F}_q[w]/\langle w^N + 1\rangle$ constrói-se:

$$f(z_i) = f(w) \bmod (w - z_i), \quad i = 0, \ldots, N-1.$$

- Inversamente, conhecendo $f(z_i)$ reconstrói-se $f(w)$ fazendo:

$$f(w) = \sum_{i=0}^{N-1} f(z_i) e_i(w) \bmod (w^N + 1).$$

Com esta conversão de representações, a multiplicação torna-se simples. Para multiplicar polinómios $f(w), g(w)$ representados por vectores de coeficientes:

- Calcula-se os vários valores $f(z_i), g(z_i) \in \mathbb{F}_{q^k}$;
- Multiplica-se em $\mathbb{F}_{q^k}$ cada um destes pares de valores e obtém-se

$$(f \cdot g)(z_i) = f(z_i) \cdot g(z_i).$$

- Reconstrói-se $(f \cdot g)(w)$ usando o TCR como já foi referido.

4

RETICULADOS

Os reticulados permitem desenvolver uma diversidade de estruturas criptográficas que assumem muita importância numa classe de técnicas designada por **Criptografia Pós - Quântica**.

Normalmente as técnicas nesta classe caracterizam-se por fundamentar a sua segurança na complexidade computacional de certos problemas que são difíceis mesmo usando computação quântica. Adicionalmente, os criptossistemas desta classe recorrem a algoritmos muito simples e eficientes em relação ao tempo de execução, sendo muito ineficientes no que concerne ao espaço ocupado.

Deste modo esta classe apresenta importantes vantagens em relação às técnicas mais clássicas. O único e maior obstáculo à sua utilização é a enorme quantidade de memória exigida pelos criptossistemas.

Veremos que um reticulado é, em particular, um grupo abeliano finitamente gerado, ou seja, é um grupo abeliano no qual é possível identificar um subgrupo de elementos de modo a que qualquer outro elemento dessa estrutura possa ser construído através de um número finito de somas desses elementos.

Em mais detalhe e de acordo com a definição de Micciancio e Regev [1], num espaço vetorial real de dimensão n, um reticulado é um conjunto de pontos com uma estrutura periódica, como a ilustrada na Figura 2. Mais exatamente, dados n vetores linearmente independentes, $v_1, \ldots, v_n \in \mathbb{R}^n$, designados por base do reticulado, o reticulado gerado por eles é o conjunto de vetores:

$$\mathcal{L}(v_1, \ldots, v_n) := \left\{ \sum_{i=1}^{n} \alpha_i v_i \middle| \alpha_i \in \mathbb{Z} \right\}.$$

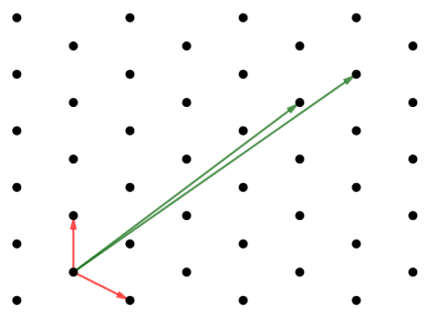

Figura 2: Reticulado em $\mathbb{R}^2$ e duas bases identificadas.

A forma como os reticulados são associados à criptografia não é muito óbvia e resultou de um artigo muito inovador publicado em 1996 por Ajtai [14]. Desde então o estudo de reticulados ganhou relevância não só como ferramenta na criptoanálise, tendo Don Coppersmith um papel fulcral nesta área [15], mas também na construção de primitivas criptográficas.

4.1 PROPRIEDADES DOS RETICULADOS

Os reticulados podem ser classificados em: **reticulados euclidianos**, **reticulados inteiros** ou **reticulados de ideais**. Sucintamente, os reticulados euclidianos são reticulados gerados por uma base finita de um espaço vectorial. Os reticulados inteiros são os subgrupos aditivos de $\mathbb{Z}^n$, para algum n. E, por fim, os reticulados de ideais têm a forma de um ideal I num anel quociente $\mathbb{Z}[x]/\langle f\rangle$, sendo f um polinómio mónico irredutível de grau n.

Note-se que a base de um reticulado pode ser representada por uma matriz $B = [b_1, \ldots, b_n] \in \mathbb{R}^{n\times n}$, tendo esta como colunas os vetores da base. Deste modo, o reticulado gerado pelas colunas de uma matriz B pode definir-se da seguinte forma:

$$\mathcal{L}(B) = \{B\mathrm{x} : \mathrm{x} \in \mathbb{Z}^n\}.$$

À matriz B denominamos por matriz base.

Seja U uma matriz unimodular, isto é, uma matriz quadrada inteira tal que:

$$|det(U)| = 1, U \in \mathbb{Z}^{n\times n}.$$

As colunas de B e BU geram o mesmo reticulado. Mais, $\mathcal{L}(B) = \mathcal{L}(B')$ se e só se $B' = BU$, onde U é matriz unimodular. Em particular, qualquer reticulado admite múltiplas bases e esse facto está na origem de muitas aplicações criptográficas e inclusive em mudanças de base para que dessas possam surgir algum benefício associado à capacidade de, através de um reticulado, ser possível resolver algum problema difícil (Na secção seguinte serão apresentados alguns problemas difíceis em reticulados).

As mudanças de base implicam que se classifique determinadas bases de reticulados como "más bases" (aquelas que dificultam a resolução de problemas) e "boas bases" (as que facilitam essa resolução). Contudo, concretizar este tipo de distinção é um processo que depende do tipo de reticulado.

O determinante de um reticulado é o valor absoluto do determinante da matriz base, isto é:

$$\det(\mathcal{L}(B)) = |\det(B)|.$$

Seja $\langle\cdot,\cdot\rangle$ o produto interno usual. O dual do reticulado $\mathcal{L}$ em $\mathbb{R}$, denotado por $\mathcal{L}^*$, é o reticulado dado pelo conjunto de vetores $y \in \mathbb{R}^n$ que satisfazem $\langle x, y\rangle \in \mathbb{Z}$, para todos os vetores $x \in \mathcal{L}$, i.e,

$$\mathcal{L}^* = \{y : \langle x, y\rangle \in \mathbb{Z}, x \in \mathcal{L}\}.$$

Observe-se que para qualquer $B \in \mathbb{R}^{n\times n}$, $\mathcal{L}(B)^* = \mathcal{L}((B^{-1})^T)$ [1] e, consequentemente:

$$\det(\mathcal{L}^*) = \frac{1}{\det(\mathcal{L})}.$$

Na classe dos reticulados inteiros existe um exemplo de reticulados com particular importância, os reticulados modulares definidos por um inteiro q e designados por **reticulados q-ários**.

Um reticulado $\mathcal{L}$ diz-se q-ário se:

$$q\mathbb{Z}^n \subseteq \mathcal{L} \subseteq \mathbb{Z}^n.$$

Observe-se também que qualquer reticulado inteiro $\mathcal{L} \subseteq \mathbb{Z}^n$ é um reticulado q-ário para algum q, por exemplo, sempre que q é um múltiplo inteiro do determinante $\det(\mathcal{L})$.

Dada uma matriz $A \in \mathbb{Z}_q^{n\times m}$, sendo q, m, n, números inteiros, é possível definir dois reticulados q-ários de dimensão m:

$$\Lambda_q(A) = \left\{y \in \mathbb{Z}^m : y = A^T s (\mathrm{mod} q) \text{ para algum } s \in \mathbb{Z}^n\right\}$$

$$\Lambda_q^{\perp}(A) = \{y \in \mathbb{Z}^m : Ay = 0 (\mathrm{mod} q)\}$$

O primeiro reticulado q-ário é gerada pelas linhas de A e está intimamente relacionado com o problema LWE, mais à frente estudado.

O segundo é a versão ortogonal e contém todos os vetores que são ortogonais módulo q às linhas de A.

Os reticulados q-ários estão em correspondência direta com os códigos lineares em $\mathbb{Z}_q^n$, e deste modo o reticulado $\Lambda_q(A)$ corresponde ao código gerado pelas linhas de A e o reticulado $\Lambda_q^{\perp}(A)$ corresponde ao código cuja matriz de verificação de paridade é A e, portanto, é espaço vetorial. Este reticulado por sua vez é usado para resolver o problema SIS, também mais à frente mencionado.

Da definição de que estes reticulados são o dual um do outro segue-se que:

$$\Lambda_q^{\perp}(A) = q.\Lambda_q(A)^* \text{ e } \Lambda_q(A) = q.\Lambda_q^{\perp}(A)^*.$$

4.2 PROBLEMAS DIFÍCEIS EM RETICULADOS

Nesta secção faz-se uma pequena descrição dos problemas computacionais que envolvem reticulados e cuja dificuldade está na essência da ***Lattice-based Cryptography***.

Os problemas computacionais mais conhecidos em reticulados são os seguintes:

- ***Shortest Vector Problem* (SVP):** Este problema consiste em, dada uma base arbitrária B de um reticulado, encontrar o menor vetor não-nulo em $\mathcal{L}(B)$. Na prática normalmente utiliza-se um fator de aproximação γ, ou seja, SVP_γ pretende encontrar um vetor cujo módulo seja menor ou igual ao menor vetor não nulo multiplicado pelo fator γ. [1]

- ***Closest Vector Problem* (CVP):** Neste problema dada uma base B de um reticulado e um vetor t, designado por 'target' e não necessariamente pertencente ao reticulado, o objetivo é determinar o ponto $v \in \mathcal{L}(B)$ mais próximo de t.

Ao longo do tempo os problemas *standards* SVP e CVP deram origem a novos problemas, que são muito importantes para vários criptossistemas pós-quânticos. Segue-se agora uma breve descrição de alguns desses problemas:

- ***Shortest Independent Vector Problem* (SIVP):** Neste problema, dada uma base $B \in \mathbb{Z}^{n \times n}$, procura-se encontrar n vetores s_i linearmente independentes que pertençam ao reticulado ($s_i \in \mathcal{L}(B)$) e que minimizem $||S|| = \max_i ||s_i||$, sendo $S = [s_1, \ldots, s_n]$. [1]

- ***Learning With Errors* (LWE):** Este problema é muito versátil graças ao seu elevado nível teórico de segurança e flexibilidade [16] e já foi introduzido no Capítulo 3. Resumidamente, consiste em, dada uma sequência aleatória de equações lineares 'aproximadas' em s, encontrar um segredo $s \in \mathbb{Z}_q^n$. [17] Existe duas variantes do LWE, uma quando o reticulado e o problema são definidos sobre um anel (R-LWE) e outra quando são definidos sobre um grupo modular (M-LWE). Estes problemas serão descritos em detalhe na Seção 5.1.

- ***Short Integer Problem* (SIS):** Este é conhecido por ser o problema "dual" do LWE. Neste problema dada uma sequência de vetores $a_1, a_2, \ldots, a_n$ escolhidos uniformemente em $\mathbb{Z}_q^n$, objetiva-se encontrar uma combinação linear com coeficientes pequenos cuja soma seja igual a 0 ($mod\ q$). [16] Analogamente este problema possui uma variante modular (M-SIS) e também será aqui descrito mais pormenorizadamente na Seção 5.2.

4.3 CLASSES DE RETICULADOS

Como já foi mencionado no início da Seção 4.1, os reticulados classificam-se em reticulados euclidianos, inteiros e de ideais.

A aplicabilidade criptográfica dos problemas difíceis em reticulados, enunciados atrás, depende inevitavelmente da capacidade de comprimir as representações desses reticulados. Essa compressão é possível em duas classes de reticulados: os reticulados de ideais e os os reticulados cíclicos, sendo estes um caso particular dos reticulados de ideais. [18]

Adicionalmente, um criptossistema baseado em reticulados gerais normalmente é bastante ineficiente, como já foi mencionado, principalmente devido ao tamanho da chave que estes exigem. Por esse motivo, os protocolos utilizados na prática baseiam-se em reticulados de ideais e cíclicos, que serão agora apresentados.

4.3.1 *Reticulados cíclicos*

Nesta secção, todos os reticulados pertencem à classe geral dos reticulados que são sub-reticulados de algum $\mathbb{Z}^m$ e que são periódicos de período q. Tal significa que em qualquer sub-reticulado $\mathcal{L} \subseteq \mathbb{Z}^m$, se:

$$a \in \mathcal{L} \text{ e } a \equiv a' \bmod q \text{ então } a' \in \mathcal{L}.$$

O parâmetro n é responsável por determinar a complexidade dos vários problemas em reticulados e está associado ao tamanho da base B que gera o reticulado.

Quando $B = \{b_1, \ldots, b_n\}$, com $b_i \in \mathbb{Z}^m$ e $||b_i||_\infty < \frac{q}{2}$ o reticulado é:

$$\mathcal{L}(B) = \{y \mid y = z_1 b_1 + z_2 b_2 + \cdots + z_m b_m \mod q\},$$

para $z_1, \ldots, z_m \in \mathbb{Z}$.

Note-se que é usada a norma l_∞: $||a||_\infty = max_i|a_i|$, e como a geração do reticulado é feita módulo q, todos os elementos desse possuem componentes congruentes com inteiros no intervalo $[-\frac{q}{2}, \frac{q}{2}]$.

Sobre a estrutura comum dos reticulados já apresentados acresce agora um operador rotação

$$rot : \mathbb{Z}^m \longrightarrow \mathbb{Z}^m$$

definido por: $rot(a)_1 = a_m$ e $rot(a)_{i+1} = a_i$, para $i < m$.

Ou seja, $rot(v) = C \cdot v$, onde C é a matriz circulante:

$$C = \begin{bmatrix} 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \\ 1 & 0 & \cdots & 0 \end{bmatrix}$$

Note-se que esta aplicação $rot(\cdot)$ é um operador linear bijectivo porque C é uma matriz invertível, donde segue a seguinte definição:

Definição 4.3.1 *Um reticulado $\mathcal{L} \subseteq \mathbb{Z}^m$ é cíclico se for fechado para o operador de deslocamento rotacional.*

Ou seja, $\mathcal{L}$ é cíclico se: $\forall v \in \mathcal{L}$: $rot(v) \in \mathcal{L}$.

Exemplos:

- $\mathbb{Z}^m$ é um reticulado cíclico.
- Reticulados correspondentes a qualquer ideal no anel polinomial quociente $R = \mathbb{Z}[w]/\langle w^m - 1\rangle$ são cíclicos.

As propriedades destes reticulados sugerem ainda que a cada vector $a = (a_1, \cdots, a_m)$, com $a_i \in \mathbb{Z}$, se associe a uma representação polinomial:

$$a(w) = a_1 + a_2 w + \cdots + a_n w^{m-1}$$

Note-se que $a(w)$ é elemento do anel de polinómios $\mathbb{Z}[w]$.

Com esta representação o operador rot é uma representação da multiplicação por w módulo $(w^m - 1)$:

$$rot(a)(w) = w \cdot a(w) \ mod(w^m - 1).$$

Proposição 4.3.2 *$\mathcal{L} \subseteq \mathbb{Z}^m$ é um reticulado cíclico se e só se for um ideal no anel $\mathbb{Z}[w]/\langle w^m - 1\rangle$.*

Prova:

Se $\mathcal{L} \subseteq \mathbb{Z}[w]$ for um ideal no anel $\mathbb{Z}[w]/\langle w^m - 1\rangle$, então é um reticulado, pois todos os ideais de polinómios inteiros são reticulados. Adicionalmente é cíclico, porque, por ser um ideal, $a \in \mathcal{L}$ implica $rot(a) = w \cdot a \in \mathcal{L}$.

Inversamente, se $\mathcal{L}$ for reticulado cíclico, então é fechado para a soma, como qualquer reticulado, e é fechado para a multiplicação com qualquer $b \in \mathbb{Z}[w]$, porque:

$$b \cdot a = \sum_{i=1}^{m} b_i rot^i(a).$$

E, para $a \in \mathcal{L}$, todos os $rot^i(a)$ também pertencem a $\mathcal{L}$, porque $\mathcal{L}$ é cíclico. Logo, $\mathcal{L}$ é um ideal no anel $\mathbb{Z}[w]/\langle w^m - 1\rangle$. ■

Esta proposição é importante para a compressão do reticulado, pois identifica cada reticulado cíclico como um ideal no anel $\mathbb{Z}[w]/\langle w^m - 1\rangle$.

Como todo o ideal é definido por um conjunto de geradores, o reticulado é igualmente determinado por esse conjunto de geradores.

Um ideal $I \subseteq R_m = \mathbb{Z}[w]/\langle w^m - 1\rangle$ com k geradores $g_1, \ldots, g_k \in I$ é gerado da seguinte forma:

$$I = \{u_1 \cdot g_1 + u_2 \cdot g_2 + \cdots + u_k g_k\}, \text{ com } u_i \in \mathcal{R}_m.$$

Basicamente substitui-se as combinações lineares com elementos da base $\{b_1, \ldots, b_n\}$ e com coeficientes inteiros, por combinações lineares dos geradores $\{g_1, \ldots, g_k\}$ com coeficientes polinomiais.

Existem duas circunstâncias que garantem que a representação de reticulados através de geradores é muito mais eficiente, a nível de espaço, do que a representação clássica por bases:

- O tamanho de cada vetor da base b_i é sempre m. Já o tamanho de cada gerador é $1 + deg(g_i)$, onde $deg(g_i)$ é o grau do polinómio g_i. Ora, o grau de g_i pode ser substancialmente inferior a m.
- O número de geradores k é determinado pelo número de factores irredutíveis do polinómio $w^m - 1$. Já o número de elementos da base n é normalmente da ordem das várias centenas, enquanto que o número de factores do módulo $(w^m - 1)$ é da ordem das unidades.

Deste modo, a representação de reticulados cíclicos por geradores é um avanço muito considerável no que diz respeito à compressão dos reticulados.

Para determinar o número de geradores de $\mathcal{L}$, o polinómio $w^m - 1$ não pode ser irredutível.

Sabe-se que,

$$(w^m - 1) = \prod_{d/m} \phi_d(w),$$

em que $\phi_d(w)$ é designado por polinómio ciclotómico de ordem d.

Note-se que um polinómio ciclotómico pode ser muito facilmente gerado, recorrendo ao *SageMath*:

```
from sage.rings.polynomial.cyclotomic import cyclotomic_coeffs

def pol_ciclotomico(n, q = None):
    lst = cyclotomic_coeffs(n)
    if q == None:
         return PolynomialRing(ZZ,'x')(lst)
    elif not is_prime(q):
        raise ValueError('q tem de ser primo!')
    else:
        return PolynomialRing(GF(q), 'x')(lst)
```

Nomeadamente, é fácil de confirmar que, por exemplo, o polinómio ciclotómico de ordem 10 é $\phi_{10}(w) = x^4 - x^3 + x^2 - x + 1$:

```
In: pol_ciclotomico(10)
Out: x^4 - x^3 + x^2 - x + 1
```

Sem entrar em grande detalhe, sabe-se que todos os polinómios ciclotómicos são mónicos e irredutíveis e que os anéis

$$C_d = \mathbb{Z}[w]/\langle \phi_d(w) \rangle$$

verificam as seguintes propriedades:

1. Todo o ideal $I \subseteq C_d$ é principal.
2. Todo o ideal $I \subseteq C_d$ é um $\mathbb{Z}-$ módulo de dimensão igual a $m - \deg(\phi_d(w))$ em $\mathbb{Z}^m$.

Quando m é primo (caso frequentemente usado em aplicações criptográficas) o polinómio $w^m - 1$ factoriza-se como se segue:

$$w^m - 1 = (w-1) \cdot (w^{m-1} + w^{m-2} + \cdots + w + 1).$$

Observe-se que o primeiro factor é o polinómio $\phi_1(w)$ de grau 1 e o segundo factor é o polinómio $\phi_m(w)$ de grau $m-1$. Consequentemente, C_1 tem dimensão $m - \deg(\phi_1(w)) = m - 1$ e C_m tem dimensão $m - \deg(\phi_m(w)) = m - (m-1) = 1$.

Uma vez que os reticulados são periódicos, caso o período q seja primo, e não dividir $m-1$, então os factores $\phi_1(w)$ e $\phi_m(w)$ são irredutíveis módulo q. Nestas circunstâncias

cada ideal $\mathcal{L} \subseteq R_m$ é gerado por dois geradores $\{g_1, g_2\}$, ambos polinómios de grau no máximo $m-1$ tais que:

$$g_1 = 0 \bmod \phi_1 \text{ e } g_2 = 0 \bmod \phi_m.$$

Como ϕ_m tem grau $m-1$ então g_2 é um múltiplo inteiro de ϕ_m, ou seja:

$$g_2(w) = p \cdot \phi_m(w) \text{ para algum } p \in \mathbb{Z}.$$

Já g_1 possui um factor $(w-1)$, ou seja:

$$g_1 = r \cdot \phi_1(w) \text{ para } \deg(r) \leq m-2.$$

O ideal $\langle g_2 \rangle$ tem dimensão 1 enquanto que o ideal $\langle g_1 \rangle$ tem dimensão $m - \deg(r) - 1$. Como:

$$\mathcal{L} = \langle g_1 \rangle + \langle g_2 \rangle,$$

o elemento genérico $a \in \mathcal{L}$ resulta da soma de dois polinómios, $a = a_1 + a_2$, sendo $a_1 \in \langle g_1 \rangle$ e $a_2 \in \langle g_2 \rangle$.

Mais, a_1 é elemento do $\mathbb{Z}-$módulo de dimensão $m - \deg(r) - 1$ e a_2 pertence a um $\mathbb{Z}-$ módulo de dimensão 1. Logo o par (a_1, a_2) pertence a um $\mathbb{Z}-$ módulo de dimensão $n = m - \deg(r)$, sendo que este n será a **dimensão do reticulado**.

Assim, o reticulado é totalmente determinado pelo polinómio r, de grau $m-n$, e pelo inteiro p, sendo necessário $m-n+2$ inteiros para determinar o reticulado por completo.

Em síntese, para determinar um reticulado cíclico em $\mathbb{Z}^m$ de período q e dimensão no máximo n, deve-se proceder da seguinte forma:

- Escolher m e q primos tal que q não divide $(m-1)$;
- Escolher um inteiro p e um polinómio $r \in \mathbb{Z}[w]$ não necessariamente mónico e de grau $m-n$;
- Definir:
$$g_1 = r \cdot (w-1) \text{ e } g_2 = p \cdot (w^{m-1} + \cdots + w + 1);$$
- Definir $\mathcal{L} \subseteq \mathbb{Z}^m$ tal que:

$$y \in \mathcal{L} \text{ se e só se } y = a \cdot g_1 + k \cdot g_2 \bmod (w^m - 1) \bmod q,$$

para algum $a \in \mathbb{Z}[w]$ e $k \in \mathbb{Z}$, com $\deg(a) \leq n-2$.

Evidentemente este procedimento não é o único que consegue gerar reticulados cíclicos mas é vantajoso porque permite controlar a dimensão do reticulado construído em função do número de *bits* usados na sua definição. Note-se que esta construção requer $(m-n+2) \cdot |q|$ *bits* para a descrição total do reticulado.

Consoante as técnicas criptográficas pode acontecer que seja necessário uma maior flexibilidade na escolha dos parâmetros m e q. Em tais circunstâncias pode-se definir o reticulado cíclico com base no seguinte procedimento:

- Escolher um polinómio $g \in \mathbb{Z}[w]$, com $\deg(g) < m$
- Definir $\mathcal{L}(g) \subseteq \mathbb{Z}^m$ como $y \in \mathcal{L}(g)$ se e só se para algum $u \in \mathbb{Z}[w]$ se verifica:

$$y = u \cdot g \bmod (w^m - 1) \bmod q.$$

Com esta definição, reticulados da forma $\mathcal{L}(g)$ são sempre ideais principais com repetição periódica, de período q, do ideal $\langle g \rangle$ gerado por g no anel $\mathbb{Z}[w]/\langle w^m - 1 \rangle$.

Como, neste anel, nem todos os ideais são principais (até porque o polinómio $w^m + 1$ não é irredutível), vão existir naturalmente reticulados cíclicos que não podem ser descritos desta forma.

No que concerne à dimensão, a sua determinação vai depender da relação entre os factores irredutíveis (módulo q) de g e do polinómio $(w^m - 1)$.

No entanto, à partida é fácil definir essa dimensão, pois é o número de elementos linearmente independentes na base:

$$B = \{g, w \cdot g, w^2 \cdot g, \ldots, w^{m-1} \cdot g\}.$$

Considerando o caso anterior, se g for irredutível (módulo q) e um divisor de $(w^m - 1)$, então a dimensão é:

$$n = m - \deg(g).$$

Contudo, noutras circunstâncias, pode não ser tão intuitivo calcular a dimensão do reticulado.

4.3.2 *Reticulados de ideais*

Os reticulados que são ideais, denominados em inglês por *ideal lattices*, são sub-reticulados de $\mathbb{Z}^m$ que são ideais num anel

$$\mathbb{Z}[w]/\langle f(w) \rangle,$$

em que $f(w) \in \mathbb{Z}[w]$ é um polinómio mónico e de grau m. Em aplicações criptográficas $f(w)$ é ainda irredutível.

O ideal gerado por $f(w)$ é:

$$\langle f(w) \rangle := f(w) \cdot \mathbb{Z}[w] = \{f(w)g(w) : \forall g(w) \in \mathbb{Z}[w]\}.$$

O anel quociente $R = \mathbb{Z}[w]\langle f(w) \rangle$ faz a partição de $\mathbb{Z}[w]$ em classes de equivalência de polinómios de grau no máximo $m - 1$:

$$R = \mathbb{Z}[w]/\langle f(w) \rangle = \left\{ \sum_{i=0}^{m-1} a_i w^i : a_i \in \mathbb{Z} \right\},$$

onde adição e multiplicação são efetuadas módulo $f(w)$. Então, cada ideal em R define um sub-reticulado de $\mathbb{Z}^m$ designado por reticulado de ideais.

Os exemplos mais comuns para f sãos os polinómios da forma:

$$f(w) = w^m + 1,$$

ou da forma mais geral:

$$f(w) = w^m + h(w),$$

sendo $h(w)$ um polinómio mónico de grau muito inferior a m.

Normalmente o grau de $f(w)$ é da ordem das várias centenas enquanto que o grau de $h(w)$ é da ordem das unidades.

Por simplificação e sem perda de generalidade considere-se o exemplo $w^m + 1$.

O polinómio $w^m + 1$ divide sempre um polinómio do tipo $w^{2m} - 1$ usado nos reticulados cíclicos, pois:

$$(w^m + 1) \cdot (w^m - 1) = w^{2 \cdot m} - 1.$$

Quando m é uma potência de 2, i.e $m = 2^d$ então $w^m + 1$ é irredutível em $\mathbb{Z}[w]$ (Provado em 2.2) e é o polinómio ciclotómico de ordem 2^{d+1}, tal que:

$$\phi_{2^{d+1}}(w) = w^{2^d} + 1.$$

Segue-se agora alguns factos sobre ideais e respectiva construção úteis para entender melhor as suas aplicações criptográficas. Todas as seguintes observações dizem respeito a ideais em $\mathbb{Z}[w]$.

Recorde-se que um ideal $I \subseteq \mathbb{Z}[w]$ é qualquer conjunto fechado para a soma e fechado para a multiplicação por qualquer polinómio, isto é:

$$\text{Se } x, y \in I \text{ então } x + y \in I \text{ e se } u \in \mathbb{Z}[w] \text{ então } u \cdot x, u \cdot y \in I.$$

A soma de ideais I, J representa-se por $I + J$ e é o conjunto de todas as somas $x + y$, com $x \in I$ e $y \in J$.

O ideal quociente I/J é o anel definido pela equivalência $x \equiv y$ se e só se $x, y \in I$ e $x - y \in J$. Este quociente só se define quando $J \subseteq I$.

O ideal formado pelos múltiplos de um polinómio $f \in \mathbb{Z}[w]$ diz-se principal e representa-se por $\langle f \rangle$.

Considere-se agora então um polinómio $f \in \mathbb{Z}[w]$ que seja mónico e irredutível, e um segundo polinómio h que seja múltiplo de f, ou seja, $h \equiv 0 \bmod f$.

Obviamente $\langle h \rangle \subseteq \langle f \rangle$ pois cada múltiplo de h é um múltiplo de f. Nem $\langle f \rangle$ nem $\langle h \rangle$ são reticulados, porém o quociente destes já o é.

Proposição 4.3.3 *Nas condições referidas, o espaço quociente $\langle f \rangle / \langle h \rangle$ é um $\mathbb{Z}-$ módulo de dimensão igual a $deg(h) - deg(f)$.*

Nas aplicações criptográficas interessam os reticulados principais, com um gerador g que não seja nulo módulo f, i.e, $g \not\equiv 0 \bmod f$ ou, equivalentemente $g \notin \langle f \rangle$.

O ideal típico tem a forma:

$$I = \{x \in \mathbb{Z}[w] | \ \exists u : u \cdot g = x \bmod \ f\}$$

Tal significa que $x \in I$ se e só se é representável por uma soma $x = u \cdot g + v \cdot f$ para alguns $u, v \in \mathbb{Z}[w]$. Ou seja, quando $I = \langle g \rangle + \langle f \rangle$, se um qualquer h for múltiplo de f, interessa-nos o ideal com o mesmo gerador g mas com as operações feitas módulo h. Isto é:

$$J = \{x \in \mathbb{Z}[w] | \; \exists u : u \cdot g = x \bmod h\}$$

Observe-se que $J = \langle g \rangle + \langle h \rangle$ e, como $\langle h \rangle \subseteq \langle f \rangle$, tem-se $J \subseteq I$ e, considerando o quocientes destes ideais temos a seguinte proposição:

Proposição 4.3.4 *Se f é mónico e irredutível, $h \equiv 0 \bmod f$ e $g \not\equiv 0 \bmod f$, então:*

$$\frac{\langle g \rangle + \langle f \rangle}{\langle g \rangle + \langle h \rangle} \text{ é isomórfico com } \frac{\langle f \rangle}{\langle h \rangle}.$$

Como consequências destas duas últimas proposições e, para quaisquer polinómios g, f, h nas condições apresentadas, pode-se afirmar que o ideal I, atrás definido, identifica-se com um de dimensão $\deg(h) - \deg(f)$ no $\mathbb{Z}-$ módulo J.

Uma terceira proposição ajuda a perceber melhor o contexto:

Proposição 4.3.5 *Se f é mónico e irredutível em $\mathbb{Z}[w]$ então todo o ideal I é principal módulo f. Isto é, pode representar-se como:*

$$I = \langle g \rangle + \langle f \rangle,$$

para algum polinómio g.

Um outro aspecto interessante é a periodicidade. Normalmente fixa-se um inteiro q, o módulo ou período, e os ideais tem a seguinte forma:

$$I = \{x \in \mathbb{Z}[w] | \; x = u \cdot g \bmod f \bmod q, \text{ para } u \in \mathbb{Z}[w]\},$$

com $||g|| \leq \frac{q}{2}$ e $||f|| \leq \frac{q}{2}$. Igualmente:

$$J = \{x \in \mathbb{Z}[w] | \; x = u \cdot g \bmod h \bmod q, \text{ para } u \in \mathbb{Z}[w]\}.$$

Analogamente temos que I e J se podem representar como:

$$I = \langle g \rangle + \langle f \rangle + \langle q \rangle \text{ e } J = \langle g \rangle + \langle h \rangle + \langle q \rangle.$$

Logo, pela Proposição 4.3.4:

$$I / J \simeq \langle f \rangle / \langle h \rangle.$$

Para terminar observe-se que, recorrendo ao *SageMath*, é simples construir uma matriz geradora de um reticulado sobre ideais assim como o reticulado gerado por essa matriz.

Constrói-se agora, a título de exemplo, a matriz geradora de um reticulado sobre ideais, em que n e q são números primos e o polinómio quociente é $x^n - 1$.

```
import sage.crypto.lattice as lat
# http://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/lattice.html

import sage.crypto.lwe as lwe
# http://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/lwe.html

import sage.modules.free_module_integer as fmi
# http://doc.sagemath.org/html/en/reference/modules/sage/modules/free_module_integer.html

import numpy as np

k = 3; n = next_prime(2^k); m = 2*n ; q = next_prime(m); f = x^n-1

print 'n=',n,' m=',m,' q=',q,' f=',f

G = lat.gen_lattice(type='ideal',n=n,m=m,q=q,quotient=f)
print 'G=\n',pretty_print(G)
```

Observe-se que o *output* resultante da execução do código anterior é precisamente uma matriz com a seguinte forma:

$$G \quad = \quad \left[\begin{array}{cccc|cccc} q & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & q & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & q & 0 & 0 & \cdots & 0 \\ \hline 1 \cdot h \bmod f & & & & 1 & 0 & \cdots & 0 \\ x \cdot h \bmod f & & & & 0 & 1 & \cdots & 0 \\ \cdots & & & & \cdots & \cdots & \cdots & \cdots \\ x^{n-1} \cdot h \bmod f & & & & 0 & 0 & \cdots & 1 \end{array}\right]$$

Note-se que o bloco do canto inferior esquerdo da matriz G corresponde à seguinte multiplicação:

$$w \mapsto w \cdot h \bmod f,$$

sendo h o polinómio definido na primeira linha desses bloco e o módulo f definido como $f = x^n + 1$, caso n seja uma potência de 2, ou $f = x^n - 1$ caso n seja primo.

Deste modo

$$[\, r \mid w \,] \times G \quad \equiv \quad [\, q \cdot r + w \cdot h \bmod f \;\mid\; w \,] \quad \equiv \quad [\, u \mid w \,]$$

sendo

$$u \equiv q \cdot r + w \cdot h \bmod f.$$

Assim, $[\, u \mid w \,]$ é um vector do reticulado de base G se e só se

$$u \equiv w \cdot h \bmod f \bmod q.$$

Por fim, o reticulado gerado pela matriz G consiste numa matriz densa sobre o anel dos inteiros e obtém-se executando apenas:

```
L = fmi.IntegerLattice(G)
pretty_print(L)
```

E, para originar uma matriz geradora de um reticulado cíclico o processo é exatamente o mesmo, sendo apenas necessário efectuar a substituição de parâmetros conforme se segue:

```
G = lat.gen_lattice(type='cyclotomic',seed=42)
print 'G=\n',pretty_print(G)
```

5

PROBLEMAS LWE E SIS

Neste capítulo faz-se a descrição dos problemas que fundamentam os esquemas de assinatura que irão ser descritos já no capítulo seguinte. Ambos estes problemas já aqui foram apresentados mas agora serão descritos com mais algum detalhe e definidas as suas variantes.

5.1 LWE

O problema LWE surgiu pela primeira vez em 2005 por Oded Regev [19]. Este problema é parametrizado pelos inteiros n, m, q, sendo n o parâmetro de segurança, m a dimensão do espaço vetorial e q o módulo, estando este fortemente relacionado com a complexidade. Um outro parâmetro é uma distribuição de erro $\mathcal{X}$ sobre $\mathbb{Z}$, sendo que $\mathcal{X}$ é uma distribuição Gaussiana discreta com desvio padrão $\sigma = \alpha \cdot q$, para algum $\alpha < 1$.

Seja $a \in \mathbb{Z}_q^n$ escolhido uniforme e aleatoriamente. O problema LWE consiste na recuperação do segredo aleatório $s \in \mathbb{Z}_q^n$ de m ($\geq n$) amostras da forma

$$(a, b), \text{ sendo } b = \langle s, a \rangle + e \bmod q,$$

onde e é o erro obtido da distribuição de erros $\mathcal{X}$.

A suposição no pior caso da dificuldade do problema LWE é provada tanto nas reduções quânticas como clássicas.

Existem duas versões principais do problema LWE: pesquisa, que consiste em encontrar o segredo dadas m amostras LWE, e decisão, que consiste em distinguir entre amostras LWE e outras uniformemente aleatórias.

5.1.1 *R-LWE*

Em 2010 Lyubashevsk, Peikert e Regev apresentaram uma variante do LWE, o problema *Ring Learning With Errors*.

Seja $R_q = \mathbb{Z}_q[w]/\langle w^n + 1 \rangle$ o anel dos polinómios, onde n (dimensão) é uma potência de 2, q é um número primo (módulo) e $\mathcal{X}$ é a distribuição gaussiana.

Considere ainda que:

- $a_i(w)$: é sequência de polinómios aleatórios, mas conhecidos, de R_q com coeficientes em $\mathbb{Z}_q$.

- $e_i(w)$: é sequência de polinómios aleatórios e desconhecidos pertencentes ao anel R_q, cujos coeficientes seguem a distribuição $\mathcal{X}$.
- $s(w)$: é uma sequência de m polinómios pequenos, relativamente à norma $\|\cdot\|_\infty$ e aleatórios, pertencentes ao anel R_q, cujos coeficientes seguem a distribuição $\mathcal{X}$.
- $b_i(w) = a_i(w) \cdot s(w) + e_i(w)$

O problema R-LWE consiste na recuperação do segredo aleatório $s \in R_q$ dada a amostra de m pares de polinómios $(a_i(w), b_i(w)) \in R_q \times R_q$.

O problema de R-LWE decisional por sua vez consiste em, dada uma amostra de pares de polinómios $(a_i(w), b_i(w))$, determinar se os polinómios $b_i(w)$ foram construídos como $b_i(w) = (a_i(w) \cdot s(w)) + e_i(w)$ ou foram gerados aleatoriamente a partir de R_q com coeficientes em $\mathbb{Z}_q$.

Por norma usa-se uma distribuição gaussiana discreta com média $\mu = 0$ e desvio padrão $s = \alpha \cdot q$.

A distribuição gaussiana contínua D_S tem distribuição de probabilidade

$$f(w) = \rho_s(w) / \int_{\mathbb{R}^n} \rho_s(z)dz = \rho_s(w)/s^n,$$

para a função gaussiana $\rho_s(w) = \exp(-\pi||w||^2/s^2)$.

Já a distribuição gaussiana discreta $D_{\mathcal{L},s}$ sobre $\mathcal{L}$ é definida como:

$$\begin{cases} D_s(w) = \rho_s(w)/\rho_s(\mathcal{L}) \text{ se } x \in \mathcal{L} \\ D_s(w) = 0 \text{ caso contrário} \end{cases},$$

onde $\rho_s(\mathcal{L}) = \sum_{v \in \mathcal{L}} \rho_s(v)$ é um fator de normalização.

Figura 3: Curva da distribuição gaussiana com média $\mu = 0$ e desvio padrão $\sigma = \alpha q$.

Observe-se que $\alpha \cdot q > \sqrt{n}$ é uma condição exigida pela prova da complexidade no pior caso. [9]

5.1.2 *M-LWE*

O problema *Module Learning With Errors* é uma generalização do LWE e do R-LWE.

Considere os anéis $R_q = \mathbb{Z}_q[w]/\langle w^n + 1\rangle$ e $R = \mathbb{Z}[w]/\langle w^n + 1\rangle$ anéis quocientes de polinómios, onde n é uma potência de 2, q é primo e $d \in \mathbb{N}$. Considere a_i elementos de R_q^d. O problema M-LWE consiste na recuperação do segredo aleatório s com coeficientes que seguem a distribuição de erros $\mathcal{X}^d$ em R de $m \geq 1$ amostras $(a_i, \langle a_i, s\rangle + e_i \bmod q) \in R_q^d \times R_q$, onde $e_i \in R$ é o erro com coeficientes que seguem a distribuição de erros $\mathcal{X}$.

5.2 SIS

O problema *Short Integer Solution* é usado em muitos esquemas criptográficos baseados em reticulados e foi introduzido por Ajtai [9].

O problema SIS é definido como segue: para quaisquer inteiros positivos m, n, dado um $\beta \in \mathbb{R}$ positivo, um inteiro q positivo e uma qualquer norma em $\mathbb{Z}^m$, o problema SIS consiste em encontrar uma solução $z \in \mathbb{Z}^m$ tal que $A \cdot z = 0 \bmod q$ e, $0 < ||z|| \leq \beta$, para uma matriz uniformemente aleatória $n \times m$ com entradas em $\mathbb{Z}_q$, i.e, $A \in \mathbb{Z}_q^{n\times m}$.

Este problema define-se formalmente do seguinte modo:

Definição 5.2.1 *Dado $A \in \mathbb{Z}_q^{n\times m}$, o problema SIS consiste em encontrar $z = (z_1, \ldots, z_m)^T \in \mathbb{Z}^m$, de modo que $A \cdot z = 0 \bmod q$ e $0 < ||z|| \leq \beta$.*

Em particular, para garantir que a solução $z \in \mathbb{Z}^m$ não é trivial, β deve ser menor que o módulo q. De fato, se $\beta \geq q$ e $A \in \mathbb{Z}_q^{n\times m}$, então tomamos a solução $z = (q, 0, \ldots, 0)^T \in \mathbb{Z}^m$ e $A \cdot z = 0 \bmod q$.

5.2.1 *R-SIS*

O R-SIS foi introduzido por Peikert e Rosen e é definido no anel polinomial $R = \mathbb{Z}[w]/\langle w^n + 1\rangle$ e $R_q = \mathbb{Z}_q[w]/\langle w^n + 1\rangle$, onde n é uma potência de 2. Como a instância do R-SIS é polinomial, o tamanho da chave de um esquema criptográfico baseado no R-SIS pode ser menor do que a chave de um esquema criptográfico baseado no SIS [20].

Este problema define-se formalmente da seguinte forma:

Definição 5.2.2 *Dados $a_1, \cdots, a_m \in R_q$ escolhidos independentemente de uma distribuição uniforme, o R-SIS$_{q,m,\beta}$ consiste em encontrar $z_1, \cdots, z_m \in R$ não simultaneamente nulos, tal que $\sum_{i=1}^m a_i \cdot z_i = 0 \bmod q$ e uma qualquer norma $||z|| \leq \beta$, onde $z = (z_1, \cdots, z_m)^T \in R^m$.*

5.2.2 *M-SIS*

A variante modular dos problema SIS é uma estrutura generalizada da versão anelar. De facto, o problema R-SIS previamente definido pode ser estendido para o reticulado do módulo.

Este problema define-se formalmente da seguinte modo:

Definição 5.2.3 *Dados* $a_1, \cdots, a_m \in R_q^d$ *escolhidos independentemente de uma distribuição uniforme, o M-SIS*$_{q,m,\beta}$ *consiste em encontrar* $z_1, \cdots, z_m \in R$ *não simultaneamente nulos, tal que* $\sum_{i=1}^{m} a_i \cdot z_i = 0 \bmod q$ *e* $0 < ||z|| \leq \beta$, *onde* $z = (z_1, \cdots, z_m)^T \in R^m$.

6

ESQUEMAS DE ASSINATURA DIGITAL PÓS-QUÂNTICOS

O concurso PQCS já aqui referido iniciou-se em 2016 e visa avaliar e padronizar um ou mais algoritmos de criptografia de chave pública pós-quânticos. No dia 30 de janeiro de 2019 foram anunciadas 26 candidaturas aceites à segunda fase do concurso. Dessas candidaturas, 9 eram esquemas de assinatura digital.

Nos últimos anos têm prevalecido essencialmente as seguintes abordagens nos esquemas de assinatura PQC:

- Algoritmos da família NTRU e algoritmos derivados do problema LWE e suas variantes (R-LWE,M-LWE, SIS,..), designados por "*Lattice-based*".
- Algoritmos derivados de funções "*hash*" e consequentemente designados por "*Hash-based*";
- Algoritmos assentes em funções multi-variáveis polinomiais, designadas por "*MQ-based*"
- Algoritmos baseados em provas de *Zero-Knowledge*, cifras simétricas e funções de "*hash*".

Estas quatro abordagens estavam representadas nas candidaturas aceites à segunda fase. Na imagem seguinte esquematiza-se a associação entre os esquemas de assinatura digital e respetivas abordagens.

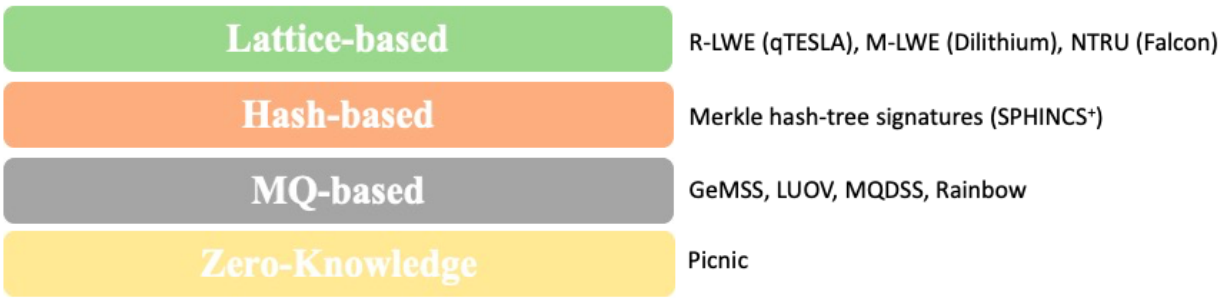

Figura 4: Esquemas de assinatura digital pós-quânticos.

No dia 22 de julho de 2020 foram apurados os finalistas para a terceira ronda do concurso PQCS. Os esquemas de assinatura digital que prevaleceram foram os seguintes:

- Crystals Dilithium;
- Falcon;
- Rainbow.

Nas próximas secções serão estudados dois esquemas de assinatura digital pós-quântica, o **qTESLA** e o **Crystals Dilithium**. Note-se que um deles já não faz parte dos classificados para a terceira ronda, o qTESLA, mas o outro ainda se encontra em concurso. O Crystals Dilithium será aqui maioritariamente abreviado por Dilithium.

Entre as opções disponíveis estes dois esquemas de assinatura foram escolhidos por permitirem uma certa coerência na abordagem aos fundamentos matemáticos que os suportam e, consequentemente, por pertencerem à abordagem da criptografia *lattice-based*, que por sua vez emergiu como um dos ramos mais promissores da criptografia moderna.

Considera-se ainda que o facto de o qTESLA ter sido desconsiderado nesta terceira ronda não faz com que se perca interesse em perceber o funcionamento do mesmo e uma vez que este era um adversário "direto" do Dilithium, dada à relativa proximidade que os une, torna-se ainda mais interessante compará-los diretamente de modo a compreender os motivos que fizeram com que o Dilithium prevalecesse em concurso ao invés do qTESLA.

6.1 qTESLA

Nesta secção apresenta-se a família de esquemas de assinatura digital *lattice-based* designada por qTESLA ("*quantum* TESLA") e cujo *site* oficial é o seguinte: https://qtesla.org/.

Esta é uma família de esquemas de assinatura pós-quânticos comprovadamente seguros, cuja segurança depende da dificuldade do problema *Ring Learning With Errors* (R-LWE).

O qTESLA resultou de um longo período de pesquisa. A história do seu algoritmo teve um marco fundamental num artigo de Bai e Galbraith, publicado em 2014 [21]. Dessa versão inicial resultaram duas abordagens: TESLA#, descrito num artigo de Barreto *et al.*, em 2017, [21] e qTESLA descrito num artigo de Alkim *et al.* , em 2017 [22]. Segue-se uma imagem que ilustra precisamente um resumo da história desta família, até à segunda ronda do PQCS, de esquemas de assinatura digital.

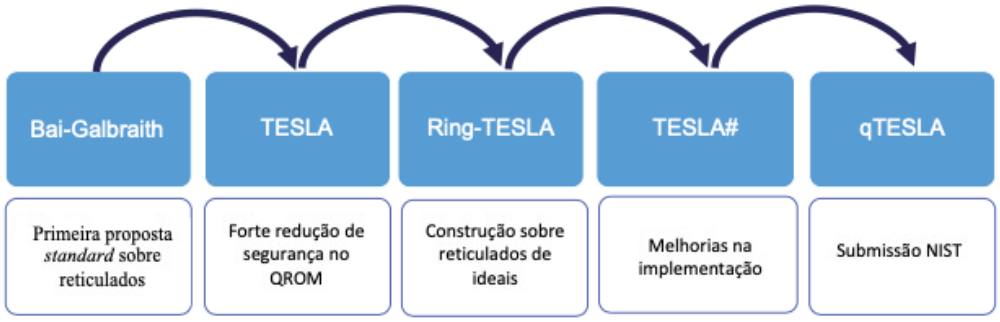

Figura 5: História do qTESLA.

As vantagens mais relevantes do qTESLA eram as seguintes:

- Flexibilidade na escolha dos parâmetros: Existiam inicialmente duas abordagens para instanciar o qTESLA: o qTESLA heurístico e o qTESLA probabilístico. Contudo, com o desenrolar do concurso na segunda ronda apenas o qTESLA probabilístico estava em competição. Esse, por sua vez incluia concretamente os seguintes conjuntos de parâmetros:
  1. qTESLA-p-I: corresponde à segurança pós quântica AES-128 (categoria de segurança 1 do NIST).
  2. qTESLA-p-III: corresponde à segurança pós quântica AES-192 (categoria de segurança 3 do NIST).
- Simplicidade: O qTESLA era estruturalmente simples e foi projetado para ser fácil de implementar. O seu *design* possibilitava a realização de implementações compactas e portáteis que atingiam alto desempenho. A distribuição gaussiana, a parte mais complexa dos esquemas tradicionais de assinatura *lattice-based*, era necessária apenas na geração das chaves.
- Segurança bem fundamentada: A segurança subjacente do qTESLA era baseada na dificuldade do problema R-LWE e era acompanhada por uma redução de segurança rígida no modelo de oráculo aleatório quântico (QROM).
- Segurança prática: O *design* do qTESLA facilitava implementações seguras contra ataques de implementação.
- Escalabilidade e portabilidade: O *design* simples do qTESLA permitia o suporte a mais de um nível de segurança e um conjunto de parâmetros com uma implementação portátil e eficiente.

6.1.1 *Notação*

Estruturas Algébricas

As estruturas base neste esquema de assinatura digital são três anéis:

- Seja q um primo diferente de 2 e seja $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ o anel de quociente de números inteiros módulo q.

  Os representantes dos elementos de $\mathbb{Z}_q$ são números inteiros pertencentes ao seguinte intervalo:
  $$I_q = \left[-\left\lfloor \frac{q}{2} \right\rfloor, \left\lfloor \frac{q}{2} \right\rfloor\right] \cap \mathbb{Z}.$$

  O tamanho de um elemento $r \in \mathbb{Z}_q$, $|r|$, é precisamente o valor absoluto do representante inteiro de r.

- Para um N potência de 2 (i.e. $N = 2^k$, para algum $k \geq 1$) define-se:
  $$\mathcal{R} = \mathbb{Z}[x]/\langle x^N + 1\rangle.$$

  Recorde-se que $x^N + 1$, com $N = 2^k$ e $k \geq 1$, é irredutível nos inteiros.

- O anel
  $$\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^N + 1\rangle$$
  é também um anel quociente mas de polinómios com coeficientes em $\mathbb{Z}_q$.

Tanto em $\mathcal{R}$ como em $\mathcal{R}_q$, os representantes de cada elemento são os polinómios de grau inferior a N. Adicionalmente, se $f = \sum_{i=0}^{N-1} f_i x^i \in \mathcal{R}$ define-se a redução de f módulo q como:

$$f = \sum_{i=0}^{N-1} (f_i \bmod q) x^i \in \mathcal{R}_q.$$

Normas infinitas

Dado $f \in \mathcal{R}_q$, a função $max_k(f)$ retorna o k-ésimo maior valor absoluto dos coeficientes de f. Para um elemento $c \in \mathbb{Z}$, temos que $||c||_\infty = |c \bmod q|$ e define-se a norma infinita para um polinómio f como $||f||_\infty = max|f_i|_\infty$, ou seja, $||f||_\infty$ é o tamanho máximo dos coeficientes do polinómio f.

Denote-se ainda por $(|f|)_h$ a soma dos tamanhos dos h maiores coeficientes de f.

Formalmente,
$$(|f|)_h = max_{I \subseteq \{0,\ldots,deg(f)\}, \#I=h} \sum_{i \in I} |f_i|$$

Nos anéis $\mathcal{R}$ e $\mathcal{R}_q$ um polinómio f diz-se B-limite ("*B-bound*") quando:

$$||f||_\infty \leq B.$$

Funções de truncatura

Seja $d \in \mathbb{N}$ e $c \in \mathbb{Z}$. Para um inteiro $q \in \mathbb{Z}_0^+$, define-se $c' = c \bmod q$ como o único elemento c' tal que $\frac{-q}{2} < c' \leq \frac{q}{2}$ e $c' = c \bmod q$. Assim, é possível determinar as funções de truncatura $[.]_L$ e $[.]_M$ nos inteiros da seguinte forma:

$$[.]_L : \mathbb{Z} \longrightarrow \mathbb{Z}, c \mapsto (c \bmod q) \bmod \ 2^d$$

$$[.]_M : \mathbb{Z} \longrightarrow \mathbb{Z}, c \mapsto \frac{(c \bmod q - [c]_L)}{2^d}$$

Note-se que a função $[.]_L$ denota os *bits* menos significativos ("*least significant bits*") e a função $[.]_M$ os *bits* mais significativos ("*most significant bits*").

Portanto, $c \bmod q = 2^d.[c]_M + [c]_L$, para $c \in \mathbb{Z}$. Estas definições são estendidas a polinómios, aplicando os operadores a cada coeficiente polinomial, ou seja,

$$[f]_L = \sum_{i=0}^{N-1} [f_i]_L x^i \text{ e } [f]_M = \sum_{i=0}^{N-1} [f_i]_M x^i, \text{ para } f = \sum_{i=0}^{N-1} f_i x^i \in \mathcal{R}.$$

Consequentemente, um polinómio $f \in \mathcal{R}$ é B-bem-arredondado ("B-*well-rounded*") quando se verifica:

$$||f||_\infty \leq \left\lfloor \frac{q}{2} \right\rfloor - B \quad \text{e} \quad ||[f]_L||_\infty \leq 2^d - B,$$

sendo que $[f]_L$ obtém-se fazendo a truncatura $[f_i]_L$ a cada coeficiente de f.

Considere também a seguinte notação:

$$\mathcal{R}_{[B]} = \left\{ \sum_{i=0}^{N-1} f_i x^i \in \mathcal{R} | f_i \in [-B, B] \right\}.$$

Representação de polinómios e strings de bits

Um determinado polinómio $f \in \mathcal{R}_q$ representa-se da seguinte forma:

$$\sum_{i=0}^{N-1} f_i x^i$$

ou, em alguns casos, como o vetor de coeficientes $(f_0, f_1, \ldots, f_{N-1}) \in \mathbb{Z}_q^N$. Quando o contexto é claro, os polinómios são representados por $a_1, \ldots, a_k$. Nestes casos, escreve-se $a_j = \sum_{i=0}^{N-1} a_{j,i} x^i$ e a representação vetorial correspondente é $a_j = (a_{j,0}, a_{j,1}, \ldots, a_{j,N-1}) \in \mathbb{Z}_q^N$, para qualquer $j \in \{1, \ldots, k\}$.

Distribuição Gaussiana $\mathcal{D}_\sigma$

A distribuição gaussiana discreta centralizada com desvio padrão σ é definida como

$$\mathcal{D}_\sigma(x) = \frac{\rho_\sigma(x)}{\rho_\sigma(\mathbb{Z})},$$

para $x \in \mathbb{Z}$, onde $\sigma > 0, \rho_\sigma(x) = \exp(\frac{-x^2}{2\sigma^2})$ e $\rho_\sigma(\mathbb{Z}) = 1 + 2\sum_{x=1}^{\infty} \rho_\sigma(x)$, sendo que este último valor só depende de σ.

Denota-se por $f \longleftarrow \mathcal{D}_\sigma$ a geração de um elemento de $\mathcal{R}$ em que os coeficientes são gerados segundo a distribuição $\mathcal{D}_\sigma$.

A transformação teórica numérica (TTN)

A multiplicação polinomial sobre um corpo finito é uma das operações fundamentais em esquemas *lattice-based*, tendo um papel muito importante no esquema de assinatura qTESLA.

A condição $q \equiv 1 \bmod 2N$ permite o uso da TTN, garantindo uma realização eficiente da multiplicação polinomial.

Por motivos de eficiência, o qTESLA especifica a geração dos polinómios $a_1, \ldots, a_k$ diretamente na estrutura TTN. Portanto, é necessário definir polinómios nessa estrutura. Considere-se assim, w uma raiz primitiva da unidade em $\mathbb{Z}_q$, ou seja, $w^N = 1 \bmod q$, e seja ϕ uma raiz da $2N$-ésima primitiva da unidade em $\mathbb{Z}_q$, de modo que $\phi^2 = w$.

Então, dado um polinómio $a = \sum_{i=0}^{N-1} a_i x^i$, a transformação é definida da seguinte forma:

$$TTN : \mathbb{Z}_q[x]/\langle x^N + 1\rangle \longrightarrow \mathbb{Z}_q^N,\ a \longrightarrow a' = \left(\sum_{j=0}^{N-1} a_j \phi^j w^{ij}\right)_{i=0,\ldots,N-1},$$

onde se diz que $a' = TTN(a)$ está na estrutura TTN. Da mesma forma, a transformação inversa do vetor a' na estrutura TTN é definida como

$$TTN^{-1} : \mathbb{Z}_q^N \longrightarrow \mathbb{Z}_q[x]/\langle x^N + 1\rangle,\ a' \longrightarrow a = \left(\sum_{i=0}^{N-1} N^{-1}\phi^{-i} \sum_{j=0}^{N-1} {a'}_j w^{-ij}\right) x^i.$$

Deste modo:

$$TTN^{-1}(TTN(a)) = a, \forall a \in \mathcal{R}_q$$

e a multiplicação polinomial de a por $b \in \mathcal{R}_q$ efectua-se da seguinte forma:

$$a \cdot b = TTN^{-1}(TTN(a) \circ TTN(b)),$$

onde "$\cdot$" denota a multiplicação polinomial em $\mathcal{R}_q$ e "$\circ$" a multiplicação entre elementos de $\mathbb{Z}_q^N$.

6.1.2 *Descrição simplificada do esquema de assinatura qTESLA*

Todos os esquemas de assinatura são determinados por um conjunto de parâmetros e três algoritmos: geração das chaves, criação da assinatura e verificação da assinatura.

As descrições informais responsáveis pelo esquema de assinatura qTESLA são apresentadas nos algoritmos 1, 2 e 3.

Antes de mais, os algoritmos a seguir apresentados requerem duas funções auxiliares:

- Como em qualquer esquema fundamentado no problema LWE, há necessidade de um gerador de inteiros que siga uma distribuição gaussiana, $\mathcal{D}_\sigma$, definido como mencionado na secção anterior.
- É fundamental ainda, como normalmente acontece, uma função de *hash*, H, que, neste caso, tem um contradomínio particular, e que é definida como se segue:

  $$H : \{0,1\}^* \longrightarrow \{-1,0,1\}^N.$$

  O contradomínio de H é formado por palavras com N símbolos ternários em que exatamente h desses símbolos são diferentes de zero.

Segue-se a notação completa da função *hash*:

$$H_{N,h} = \left\{ \sum_{i=0}^{N-1} f_i x^i \in \mathcal{R} | f_i \in \{-1,0,1\}, \sum_{i=0}^{N-1} |f_i| = h \right\}.$$

Parâmetros

Parâmetros	**Descrição**
q	Número primo responsável pelo módulo
n	Potência de 2 responsável pela ordem dos polinómios
d	Limite às truncaturas
B	Limite para o polinómio aleatório na assinatura
L_E	Constante vinculada para o polinómio de erro
L_S	Constante vinculada para o polinómio secreto
E	Limite de rejeição usado durante a assinatura e relacionado com L_E
S	Limite de rejeição usado durante a verificação e relacionado com L_S
σ	Desvio padrão da distribuição gaussiana
h	Parâmetro específico da função de *hash* H

Tabela 1: Parâmetros do qTESLA.

Algoritmo 1 Geração de chaves

Input: —

Output: A chave privada $sk = (s, e)$ e a chave pública $pk = (a, t)$

1: Gerar polinómios $e, s \longleftarrow \mathcal{D}_\sigma$
2: Se $(|e|)_h \geq L_E$ ou $(|s|)_h \geq L_S$ voltar ao passo 1
3: Gerar $a \longleftarrow \mathcal{R}_q$ invertível e calcular $t \longleftarrow a \cdot s + e \in \mathcal{R}_q$
4: A chave pública é $pk = (a, t) \in \mathcal{R}_q^2$
5: A chave privada é $sk = (s, e) \in \mathcal{R}^2$

Observe-se que a chave pública consiste num par de polinómios com coeficientes em $\mathbb{Z}_q$ e a chave privada num par de polinómios com coeficientes em $\mathbb{Z}$.

Algoritmo 2 Criação da assinatura

Input: A mensagem $m \in \{0,1\}^*$ e a chave privada $sk = (s, e)$

Output: A assinatura $sign = (z, c)$

1: Gerar aleatoriamente $r \longleftarrow \mathcal{R}_q$, com $||r||_\infty \leq B$
2: Calcular $c \longleftarrow H([ar]_M||m) \in \mathcal{R}_q$
3: Calcular $z \longleftarrow r + s \cdot c \in \mathcal{R}_q$
4: Se $||z||_\infty + L_S \geq B$ voltar ao passo 1.
5: Se $a \cdot r - e \cdot c$ não for $L_E - well - rounded$ voltar ao passo 1.
6: Devolver a assinatura $sign = (z, c)$.

Algoritmo 3 Verificação da assinatura

Input: A mensagem $m \in \{0,1\}^*$, a chave pública $pk = (a, t)$ e a assinatura $sign = (z, c)$

Output: "Aceitar" ou "rejeitar" a assinatura

1: Se $||z||_\infty + L_S \geq B$ rejeitar a assinatura
2: Calcular $w \longleftarrow a \cdot z - t \cdot c \in \mathcal{R}_q$
3: Se $c \neq H([w]_M||m)$ rejeitar a assinatura.
4: Devolver "aceitar".

Note-se ainda que, devido à geração aleatória do polinómio r, no passo 1 do algoritmo 2, este mesmo algoritmo é descrito como um algoritmo não determinístico. Esta propriedade implica que é necessário uma aleatoriedade diferente para cada assinatura. Para a especificação formal do qTESLA, incorporou-se uma melhoria adicional: o qTESLA requer uma combinação de aleatoriedade nova e um valor fixo para a geração de r. Esse novo *design* é adicionado essencialmente para proteger o código contra alguns ataques de falha simples.

Para finalizar esta secção, caso seja do interesse do leitor, segue-se o *link* do projeto oficial submetido a concurso (https://qtesla.org/wp-content/uploads/2018/08/qTESLA_v2.2_08.27.2018.pdf), onde evidentemente o qTESLA é descrito com muito maior detalhe e formalidade.

6.1.3 *Correção do qTESLA*

A correção do qTESLA, muito resumidamente, consiste em provar o seguinte:

$$\begin{aligned}[w]_M &= [a \cdot z - t \cdot c]_M \\ &= [a(r + s \cdot c) - (a \cdot s + e) \cdot c]_M \\ &= [a \cdot r - e \cdot c]_M \\ &= [a \cdot r]_M\end{aligned}$$

A especificação oficial do qTESLA exige que $a \cdot r - e \cdot c$ seja $L_E - well - bounded$, que $||r||_\infty \leq B$ e que $||z||_\infty = ||r + s \cdot c|| \leq B - L_S$. Tais condições permitem provar que

$$[w]_M = [a \cdot r]_M.$$

Portanto, a assinatura verifica-se se e só se:

$$H([w]_M||m) = H([a \cdot r]_M||m).$$

Para ter acesso à prova completa consultar [23].

6.1.4 *Parâmetros standards do qTESLA*

Parâmetros	**qTESLA-p-I**	**qTESLA-p-III**
q	343 576 577	856 145 921
n	1024	2048
d	22	24
B	2^{19}-1	2^{21}-1
L_e	554	901
L_s	554	901
σ	8,5	8,5
h	25	40
Pk size (bytes)	14880	38432
Sk size (bytes)	5224	12392
Sign size (bytes)	2592	5664

Tabela 2: Parâmetros *standards* do qTESLA.

Note-se que $q \equiv 1 \bmod 2n$ e que os tamanhos da chave pública (*Pk size*), chave privada (*Sk size*) e da assinatura (*Sign size*) são os tamanhos oficiais da implementação de referência.

6.1.5 *Implementação*

A implementação oficial do qTESLA está disponível em https://github.com/microsoft/qTESLA-Library.

No Apêndice A encontra-se a implementação não otimizada do qTESLA realizada no *SageMath 9.1*. Segue-se agora uma breve descrição dessa implementação.

Descrição da implementação não otimizada

O qTESLA possui, como já mencionado na Tabela 2, os seguintes parâmetros: q, d, B, L_e, L_s, σ e h. E, consequentemente, a implementação começa com a atribuição de valores a esses mesmos parâmetros.

Antes de mais, note-se que, algumas vezes, ao longo da implementação, define-se as estruturas algébricas já aqui apresentadas. Tal procedimento é muito simples no *SageMath*, sendo esse o factor decisivo na escolha desse *software* para as implementações feitas.

Apresenta-se agora o código responsável precisamente pela construção das estruturas algébricas do qTESLA:

```
Zx.<x> = ZZ[]
R.<x> = Zx.quotient(x^self.n+1)
Zq.<z> = PolynomialRing(GF(self.q))
Rq.<z> = Zq.quotient(z^self.n+1)
```

Segue-se agora uma breve descrição de cada uma das funções intervenientes na implementação em questão:

- *Lsb*: Função de truncatura definida como se segue:

$$[.]_L : \mathbb{Z} \longrightarrow \mathbb{Z}, c \mapsto (c \bmod q) \bmod \ 2^d.$$

- *Msb*: Função de truncatura definida do seguinte modo:

$$[.]_M : \mathbb{Z} \longrightarrow \mathbb{Z}, c \mapsto \frac{(c \bmod q - [c]_L)}{2^d}.$$

- *Hash*: Função unidirecional definida da seguinte forma:

$$H : \{0,1\}^* \longrightarrow \{-1,0,1\}^N,$$

  sendo o *output* desta formado por palavras com N símbolos ternários em que exatamente h desses símbolos são diferentes de zero.

- *pol_aux*: Função auxiliar da função *infinite_norm*. Esta recebe como *inputs* um polinómio p e um inteiro n e produz um polinómio em $\mathbb{Z}[x]$ como *output*. Esta função para além de converter o polinómio dado num polinómio em $\mathbb{Z}[x]$ faz ainda a ordenação deste por ordem decrescente dos graus dos termos.

- *infinite_norm*: Função responsável por definir a norma infinita para um polinómio.

- *well_rounded*: Função que verifica se o polinómio dado, p, é $T - well - rounded$, sendo T também um parâmetro dado.

  Mais precisamente verifica se:

  $$||p||_\infty \leq \left\lfloor \frac{q}{2} \right\rfloor - T \quad \text{e} \quad ||[p]_L||_\infty \leq 2^d - T.$$

  O *output* desta função é um valor booleano.

- *Keygen*: Função responsável pela geração das chaves e que segue a descrição do algoritmo 1. Não possui *input* e o *output* desta é a chave pública e a chave privada:

  $$PubKey = (a, t) \in \mathcal{R}_q^2 \text{ e } PrivKey = (s, e) \in \mathcal{R}^2.$$

- *Sign*: Função responsável pela geração da assinatura e que segue a descrição do algoritmo 2. O *input* desta função é apenas uma mensagem $m \in \{0,1\}^*$ e o *output* é a assinatura $sign = (z, c)$.

- *Verify*: Função responsável pela verificação da assinatura e que segue a descrição do algoritmo 3. O *input* desta é a mensagem $m \in \{0,1\}^*$ e a assinatura $sign = (z, c)$. O *output* é 'Assinatura rejeitada!' ou 'Assinatura aceite!'.

Para finalizar, observe-se que nas últimas três funções definidas no código em anexo existem comentários de modo a que leitor possa comparar diretamente a implementação com os passos dos algoritmos descritos em 6.1.2.

6.2 CRYSTALS-DILITHIUM

O algoritmo Crystals-Dilithium ("*Cryptographic Suite for Algebraic Lattices Dilithium*") foi proposto por Ducas *et al.* e consiste num esquema de assinatura digital cujo *site* oficial é o seguinte: `https://pq-crystals.org/dilithium/`.

O esquema possui, para a implementação, uma opção determinística e outra aleatória. No entanto, os autores sugerem a versão determinística como *standard*.

Este esquema de assinatura baseia-se no problema de determinar os vetores curtos de um reticulado, sendo regido por alguns critérios de implementação:

- Pretende evitar a geração de inteiros através de distribuições gaussianas (usuais nos esquemas LWE) porque é difícil de implementar tais geradores que sejam, simultaneamente, simples e imunes a alguns "*side-channel attacks*".

- Pretende assegurar, recorrendo a parâmetros apropriados, que o esquema seja seguro a longo prazo, tanto na computação clássica como na computação quântica.

- Pretende minimizar o tamanho conjunto da chave pública e da assinatura. Este é um aspecto crítico na adoção de técnicas PQC uma vez que, normalmente, as chaves e a assinatura têm um comprimento de alguns milhares de *bytes* comparando-se muito desfavoravelmente com esquemas clássicos onde esse número é, quanto muito, milhares de *bits*.

- Pretende-se que a aritmética usada seja simples e modular, de modo a que se possa construir, com muitas poucas alterações de parâmetros, uma grande variedade de configurações com diferentes níveis de segurança. Todas as configurações são baseadas no mesmo anel

  $$\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle,$$

  já aqui definido para o qTESLA, mas agora com seguintes valores fixos para n e q:

  $$n = 2^8 = 256 \text{ e } q = 2^{23} - 2^{13} + 1.$$

  Para além da multiplicação de polinómios, optimizada, tal como no qTESLA, recorrendo à TTN e ao TCR, o esquema usa apenas um XOF que, neste esquema, é o SHAKE-256.

O *design* do Dilithium baseia-se na técnica "*Fiat-Shamir with Aborts*" de Lyubashevsky, que usa distribuição por rejeição para tornar os esquemas Fiat-Shamir baseados em reticulados compactos e seguros. O esquema com os menores tamanhos de assinatura usando essa abordagem é o de Ducas, Durmus, Lepoint e Lyubashevsky, e usa a distribuição gaussiana para criar assinaturas. Como a distribuição gaussiana é difícil de implementar de forma segura e eficiente, então o Dilithium optou por usar apenas a distribuição uniforme, aprimorando o esquema mais eficiente que já o fazia.

As vantagens mais relevantes do Dilithium são:

- Simplicidade e segurança: O Dilithium usa apenas distribuição uniforme e todas as outras operações (como multiplicação polinomial e arredondamento) são facilmente implementadas em tempo constante.

- Conservação dos parâmetros: O Dilithium procura satisfazer uma segurança a longo prazo, tendo em consideração a aplicabilidade de ataques de reticulados de um ponto de vista muito favorável para o adversário. Em particular, considerando algoritmos quânticos que requerem praticamente tanto espaço quanto tempo.

- Tamanho da chave pública e da assinatura reduzido: O Dilithium foi projetado para minimizar o tamanho da chave pública e da assinatura, tendo em mente a transmissão das mesmas em cadeias de certificados e mantendo a segurança.

6.2.1 *Notação*

Estruturas Algébricas

Tal como no qTESLA a estrutura algébrica é definida pelos anéis quocientes:

$$\mathcal{R} = \mathbb{Z}[x]/\langle x^n + 1\rangle \quad \text{e} \quad \mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle,$$

com a diferença que agora q e n são fixos:

$$n = 256 \text{ e } q = 2^{23} - 2^{13} + 1.$$

Define-se ainda:

- $S_\eta \subseteq \mathcal{R}_q$ como o conjunto de todos os polinómios η-limite ("η-*bound*"):

$$\{f \in \mathcal{R}_q |\ ||f||_\infty \leq \eta\}.$$

- B_h é o conjunto de todos os polinómios em $\mathcal{R}_q$ de coeficientes ternários $\{-1, 0, 1\}$ em que exatamente h elementos são não nulos.

No Dilithium o parâmetro h é fixo no valor 60.

É ainda verificada a seguinte equivalência $q \equiv 1 \bmod 2n$ e usa-se $r = 1753$.
A TTN de um polinómio $f \in \mathcal{R}_q$ é um vetor $\hat{f} \in \mathbb{Z}_q^n$ cuja componente de ordem i é

$$\hat{f}_i \equiv f(r^i).$$

A transformação inversa $\hat{f} \longmapsto f$ é dada pela seguinte função de interpolação:

$$f(x) = \sum_{i=0}^{255} \hat{f}_i \theta_i(x),$$

em que

$$\theta_i(x) = \prod_{j \neq i} (x - r^j)/(r^i - r^j).$$

Seja $\hat{\theta}$ a matriz $\mathbb{Z}_q^{n \times n}$, em que $\hat{\theta}_{ik}$ é o coeficiente de ordem k do polinómio $\hat{\theta}_i$ e:

$$\theta_i(x) = \sum_{k=0}^{n-1} \hat{\theta}_{ik} x^k,$$

então o vector de coeficientes do polinómio f pode ser recuperado por uma simples multiplicação de matrizes $f = \hat{f}\hat{\theta}$.

Usando a TTN a multiplicação em $\mathcal{R}_q$ torna-se mais eficiente. E, para multiplicar $f \in \mathcal{R}_q$ por $g \in \mathcal{R}_q$:

(i) Calcula-se as transformadas $\hat{f}$ e $\hat{g}$

(ii) Calcula-se a transformada do produto com n multiplicações:

$$\widehat{(f \times g)}_i = \hat{f}_i \times \hat{g}_i, i \in \{0, \ldots, 255\}.$$

(iii) Recupera-se $f \times g$ a partir da transformada $\widehat{f \times g}$

Existem algoritmos que executam eficientemente os passos (i) e (ii) com complexidade $O(n \log n)$. E que permitem assim uma multiplicação, através da TTN, bem mais eficiente do que a multiplicação dita usual de polinómios e de complexidade $O(n^2)$.

O esquema Dilithium recorre à função $LowBits$ e à função $HighBits$ que recupera, respetivamente, os *bits* menos significativos e os *bits* mais significativos de um elemento $c \in \mathbb{Z}_q$.

Para qualquer α que seja divisor de $(q-1)/2$, então:

$$e = LowBits(c, 2\alpha),\ u = HighBits(c, 2\alpha),$$

quando $c = u \times 2\alpha + e$.

Estas funções estendem-se a polinómios, aplicando-se coeficiente a coeficiente.

O esquema exige também uma função de *hash* H que percorre *strings* arbitrárias em polinómios de $\mathcal{R}_q$ com vetores de coeficientes em B_h :

$$H : \{0,1\}^* \longrightarrow B_h.$$

6.2.2 *Descrição simplificada do esquema de assinatura Dilithium*

Parâmetros

Para além das constantes:

$$n = 256, q = 2^{23} - 2^{13} + 1, h = 60 \text{ e } r = 1753,$$

existem outros parâmetros que limitam os polinómios e que serão listados na tabela que se segue:

Parâmetros	**Descrição**
α	Divisor de $(q-1)/2$, usado para separar '*lowbits*' de '*highbits*'
η	Limite no tamanho dos polinómios considerados pequenos
γ, β	Limites usados para o tamanho da assinatura

Tabela 3: Parâmetros do Dilithium.

Adicionalmente o esquema usa uma matriz A, de dimensão (k, l) cujos elementos são polinómios em $\mathcal{R}_q$. As dimensões (k, e) são também parâmetros do esquema em questão.

Os algoritmos que se seguem são responsáveis pelo esquema de assinatura Dilithium:

Algoritmo 4 Geração de chaves

Input: —

Output: A chave pública $pk = (A, t)$ e a chave privada $sk = (s_1, s_2)$.

1: $A \longleftarrow \mathcal{R}_q^{k \times l}$
2: $(s_1, s_2) \longleftarrow S_\eta^l \times S_\eta^k$
3: $t \longleftarrow As_1 + s_2$
4: A chave pública é $pk = (A, t)$
5: A chave privada é $sk = (s_1, s_2)$

Algoritmo 5 Criação da assinatura

Input: A mensagem m e a chave privada $sk = (s_1, s_2)$

Output: A assinatura $\sigma = (z, c)$

1: $z :=\perp$
2: Enquanto $z =\perp$:
3: $y \longleftarrow S_{\gamma-1}^l$
4: $w \longleftarrow HighBits(Ay, 2\alpha)$
5: $c \longleftarrow H(m||w)$
6: $z \longleftarrow y + cs_1$
7: Se $||z||_\infty \geq \gamma - \beta$ ou $||LowBits(Ay - cs_2, 2\alpha)||_\infty \geq \alpha - \beta$, então $z \longleftarrow \perp$
8: Devolver $sign = (z, c)$

Algoritmo 6 Verificação da assinatura

Input: A mensagem m, a chave pública $pk = (A, t)$ e a assinatura $sign = (z, c)$

Output: "Aceitar" ou "rejeitar" a assinatura

1: Calcular $w' \longleftarrow HighBits(Az - ct, 2\alpha)$
2: Se $||z||_\infty \geq \gamma - \beta$ ou $c \neq H(m||w')$ então devolver "rejeitar"
3: Devolver "aceitar"

6.2.3 *Correção do Dilithium*

Este esquema é essencialmente análogo ao qTESLA distinguindo-se deste no facto de o polinómio grande a usado no qTESLA ser aqui substituído por uma matriz A de polinómios mais pequenos.

Distingue-se também porque aqui não exitem geradores gaussianos. Os polinómios s_1 e s_2 são gerados uniformemente num domínio de polinómios $\eta-$ *bound*.

Tem-se ainda que:

$$Az - ct = Ay + cAs_1 - cAs_1 - cs_2 = Ay - cs_2$$

$$w = HighBits(Ay, 2\alpha)$$

$$w' = HighBits(Ay - cs_2, 2\alpha)$$

Informalmente, o polinómio cs_2 tem pequenos coeficientes porque as componentes de s_2 estão limitadas ao tamanho η e c tem coeficientes ternários $\{-1, 0, 1\}$. Assim sendo, parece legítimo aceitar que os *bits* mais significativos de Ay e $Ay - cs_2$ coincidam. Os limites impostos no tamanho de z e de $Az - ct$ asseguram efetivamente que isso acontece.

6.2.4 *Parâmetros standards do Dilithum*

Para $n = 256$, $q = 2^{23} - 2^{13} + 1 = 8380417$, $h = 60$ e $r = 1753$, sendo r uma raiz primitiva de ordem n de -1:

Parâmetros	**NIST-1**	**NIST-2**	**NIST-3**
(k, l)	(4,3)	(5,4)	(6,5)
$\gamma = (q-1)/16$	523776	523776	523776
$\alpha = \gamma/2$	261888	261888	261888
η	6	5	3
β	325	275	175
Pk size (bytes)	1184	1472	1760
Sk size (bytes)	2800	3504	3856
Sign size (bytes)	2044	2701	3366

Tabela 4: Parâmetros *standards* do Dilithium.

Note-se que NIST-1, NIST-2 e NIST-3 referem-se a *Dilithium Medium*, *Dilithium Recommenden* e *Dilithium Very High*, respetivamente e que os tamanhos da chave pública (*Pk size*), chave privada (*Sk size*) e da assinatura (*Sign size*) são os tamanhos oficiais da implementação de referência.

6.2.5 *Implementação*

Quanto à implementação oficial do Dilithium esta está disponível no seguinte *site*: https://github.com/pq-crystals/dilithium.

Nessa versão, para além da multiplicação de polinómios, que recorre à TNN, a operação mais complexa é a geração da matriz A. Com valores de parâmetros $(6, 5)$, corresponde ao nível 3 de segurança da NIST, a matriz A tem 30 entradas, cada uma das quais é um polinómio com 256 coeficientes, o que faz um total de 7680 elementos de $\mathbb{Z}_q$. Tal valor é bem mais superior aos 2048 usados no qTESLA.

Por esse motivo, a matriz nunca é armazenada na chave pública, sendo armazenado uma *seed* $\rho \in \{0,1\}^{256}$, a partir da qual cada instanciação do esquema de assinaturas usa a função SHAKE-256 para construir os elementos de A.

Outro aspecto relevante, na implementação oficial, é o uso da TTN não apenas para multiplicação mas também para a soma polinómios. A parte mais complexa é a aplicação das transformadas $f \longmapsto \hat{f}$ e da sua inversa $\hat{f} \longmapsto f$. Contudo, essas transformações não são aplicadas em cada operação. Na verdade, a maioria das operações aritméticas pode ser feita na forma transformada. Nomeadamente a matriz A pode ser gerada diretamente na forma transformada $\hat{A}$.

A forma polinomial só é necessária quando se calculam normas $||\cdot||_\infty$ ou quando se aplicam $LowBits$ e $HighBits$.

Em Apêndice B encontra-se a implementação não otimizada do Dilithium realizada no *SageMath 9.1*.

Descrição da implementação não otimizada

O Dilithium possui, como já mencionado na Subseção 6.2.4, os seguintes parâmetros: $n, q, h, r, k, l, \gamma, \alpha, \eta$ e β. E, consequentemente, a implementação começa com a atribuição de valores a esses mesmos parâmetros.

Antes de mais, note-se que, tal como ocorreu no qTESLA, foi necessário definir as estruturas algébricas já aqui apresentadas e o procedimento no *SageMath* foi exatamente o mesmo.

Segue-se agora uma breve descrição de cada uma das funções intervenientes na implementação em questão:

- *gen_S*: Função que gera uma matriz linha S com entradas aleatórias em $\mathcal{R}_q$ e que é usada na geração de s_1 e s_2.

- *Decompose*: Função que recebe como *input* um inteiro c e um inteiro t. Começa por atribui valores às variáveis r e r_0, sendo a primeira o resultado de fazer $c \bmod q$ e a segunda o resultado de fazer $r \bmod t$. Se $r_0 > t/2$, então $r_0 = r_0 - t$. Se $r - r_0 = q - 1$, então $r_1 = 0$ e $r_0 = r_0 - 1$, caso contrário, $r_1 = (r - r_0)/t$. A função retorna o par (r_1, r_0).

- *HighBits*: Esta função recebe como entrada um inteiro c e consiste simplesmente em devolver a primeira entrada do *output* de $Decompose(r, 2\alpha)$.

- *LowBits*: Função análoga à anterior, sendo que em vez de devolver a primeira entrada do *output* de $Decompose(r, 2\alpha)$ devolve a segunda entrada.

- *HBpol*: Função que recorre à função $HighBits$, atrás descrita, para recuperar os *bits* mais significativos do polinómio dado como *input*.
- *LBpol*: Função que recorre à função $LowBits$ atrás descrita, para recuperar os *bits* menos significativos do polinómio dado como *input*.
- *SampleInBall*: Função responsável pela geração de um elemento aleatório em B_{60} e cujo *input* é responsável pela fonte de aleatoriedade necessária para a função $Hash$. Esta segue o próximo algoritmo:

Algoritmo 7 SampleInBall

Inicializar $c = c_0c_1 \dots c_{255} = 00 \cdots 0$
for $i := 196$ a 255
  $j \leftarrow \{0, 1, \dots, i\}$
  $s \leftarrow \{0, 1\}$
  $c_i := c_j$
  $c_j := (-1)^s$
return c

- *Shake*: Função que faz uso da bibleoteca *hashlib* para construir o *shake_256*.
- *Hash*: Função que relaciona as duas últimas funções, de modo a que o *input* da função $SampleInBall$ seja o *output* da função $Shake$.
- *norma_infinita; norma_inf_pol; norma_inf_matriz*: Funções que calculam a norma infinita do respetivo tipo de *input*, seja ele polinómio, vetor ou matriz, respetivamente.
- *Keygen*: Função responsável pela geração das chaves e que segue a descrição do algoritmo 4. Não possui *input* e o *output* desta é a chave pública e a chave privada:

$$PubKey = (A, t) \text{ e } PrivKey = (s_1, s_2).$$

- *Sign*: Função responsável pela geração da assinatura e que segue a descrição do algoritmo 5. O *input* desta função é apenas uma mensagem $m \in \{0, 1\}^*$ e o *output* é a assinatura $sign = (z, c)$.
- *Verify*: Função responsável pela verificação da assinatura e que segue a descrição do algoritmo 6. O *input* desta é a mensagem $m \in \{0, 1\}^*$ e a assinatura $sign = (z, c)$. O *output* é 'Assinatura rejeitada!' ou 'Assinatura aceite!'.

Para finalizar, observe-se que nas últimas três funções definidas no código em anexo existem comentários de modo a que leitor possa comparar diretamente a implementação com os passos dos algoritmos aqui descritos em 6.2.2.

6.3 COMPARAÇÃO ENTRE QTESLA E CRYSTALS-DILITHIUM

Nesta secção são feitas comparações inerentes ao desempenho, segurança e eficiência dos esquemas de assinatura estudados atrás. Para tal teve-se em consideração parâmetros como o modelo de segurança, os tamanhos de chave pública e da assinatura e o desempenho na execução dos algoritmos para gerar chaves, assinar e verificar.

6.3.1 *Segurança*

Os dois esquemas de assinatura digitais têm o LWE como problema computacional difícil. No entanto, são utilizadas variações deste problema. O Dilithium é fundamentado na versão modular, quer do LWE quer do SIS, e o qTESLA na versão anelar do LWE.

Segue-se agora uma tabela ilustrativa de algumas propriedades de segurança inerentes ao Dilithum e ao qTESLA, quando ambos se encontravam na segunda ronda do concurso PQCS.

	Dilithium	**qTESLA**
Níveis de segurança satisfeitos (NIST)	1,2 e 3	1 e 3
Segurança EUF-CMA	Sim	Sim
Segurança SUF-CMA	Sim	Não
Problema associado	M-LWE e M-SIS	R-LWE

Tabela 5: Comparação da Segurança do qTESLA e do Dilithium.

Para uma melhor compreensão da tabela agora mesmo apresentada segue-se uma breve apresentação de níveis de segurança satisfeitos pelo NIST e dos conceitos de segurança EUF-CMA e SUF-CMA.

Níveis de Segurança para Assinatura Digital Pós-quântica

Os níveis de segurança propostos pelo NIST e que facilitam a classificação da segurança de criptossistemas pós-quânticos são os seguintes:

1. Qualquer ataque que quebre a definição de segurança deve requerer recursos computacionais comparáveis ou superiores aos necessários para uma procura de chave de uma cifra por blocos com tamanho igual a 128 *bits* (e.g. AES128);

2. Qualquer ataque que quebre a definição de segurança deve requerer recursos computacionais comparáveis ou superiores aos necessários para uma procura de colisão de uma função *hash* com 256 *bits* de tamanho (e.g. SHA256/SHA3-256);

3. Qualquer ataque que quebre a definição de segurança deve requerer recursos computacionais comparáveis ou superiores aos necessários para uma procura de chave de uma cifra por blocos com tamanho de chave igual 192 *bits* (e.g. AES192);

4. Qualquer ataque que quebre a definição de segurança deve requerer recursos computacionais comparáveis ou superiores aos necessários para uma procura de colisão de uma função *hash* com 384 *bits* de tamanho (e.g. SHA384/SHA3-384);

5. Qualquer ataque que quebre a definição de segurança deve requerer recursos computacionais comparáveis ou superiores aos necessários para uma procura de chave de uma cifra por blocos com tamanho de chave igual a 256 *bits* (e.g. AES256).

Segurança EUF-CMA e SUF-CMA

Num sistema de assinatura digital, o objetivo de um atacante é forjar assinaturas que serão aceites como válidas. Consequentemente, a segurança de um sistema de assinatura digital pode então ser classificada conforme o poder que o adversário possui sobre o esse sistema:

1. Quebra total: o adversário pode gerar informações da chave privada do assinante ou encontrar algoritmos eficientes de assinar.

2. Quebra seletiva: o adversário consegue gerar assinaturas válidas para mensagens particulares ou para classes de mensagens escolhidas.

3. Quebra existencial: o atacante é capaz de forjar a assinatura de pelo menos uma mensagem. É um adversário mais fraco que os anteriores e os esquemas seguros contra este tipo de adversário são *existential unforgeability*, ou EUF.

4. Quebra forte: o atacante pode forjar uma nova assinatura válida sobre uma mensagem da qual ele já tinha informações sobre a assinatura. Os esquemas seguros contra esse tipo de adversário são *strong unforgeability*, ou SUF.

Existem dois tipos de ataques contra esquemas de assinatura digital:

1. Ataque de chave (*key-only attacks*): surgem quando o adversário conhece apenas a chave pública do assinante.

2. Ataque de mensagens (*message attacks*): surgem quando o atacante pode examinar as assinaturas de determinadas mensagens, conhecidas ou escolhidas.

   O ataque de mensagem conhecida surge quando o atacante conhece a chave pública e um conjunto de pares mensagem-assinaturas.

   Já no ataque de mensagem escolhida CMA, o atacante conhece apenas a chave pública e consegue gerar mensagens e receber as respectivas assinaturas válidas.

Posto isto, surgem as definições formais EUF-CMA e SUF-CMA para classificar a segurança de um esquema de assinatura digital.

Cada uma dessas definições é apresentada como um "jogo" realizado entre o atacante e a entidade honesta.

Informalmente, EUF-CMA funciona assim:

- A entidade honesta gera um par de chaves válido (pk, sk) e envia a pk ao atacante.
- O adversário solicita repetidamente assinaturas em mensagens escolhidas por si, $(m_1, \ldots, m_q)$, e recebe as assinaturas válidas $(\sigma_1, \ldots, \sigma_q)$.
- O adversário emite uma mensagem, m^*, e uma assinatura, σ^*, de modo a que a mensagem m^* não seja uma das mensagens solicitadas na etapa anterior e a assinatura seja verificada através da chave pública.

O esquema é considerado seguro se nenhum adversário tiver uma vantagem não negligenciável em satisfazer as condições acima.

Esta definição é forte, mas não tão forte quanto a definição de SUF-CMA.

Informalmente, SUF-CMA funciona do mesmo modo que o EUF-CMA, com excepção do terceiro ponto que é descrito da seguinte forma:

- O adversário emite uma mensagem, m^*, e uma assinatura, σ^*, de modo a que o par (m^*, σ^*) não seja composto por nenhuma das mensagens solicitadas nem por nenhuma assinatura devolvidas no passo anterior. A assinatura deve ainda ser verificada através da chave pública.

O atacante vence se satisfizer a condição acima.

6.3.2 *Tamanho dos parâmetros*

Um dos critérios de comparação dos esquemas de assinatura digital relaciona-se com o tamanho da chave pública e da assinatura, visto que este é também um dos critérios avaliados pelo NIST.

Seguem-se agora duas tabelas que relacionam o esquema de assinatura digital e os respetivos tamanho da chave pública e da assinatura.

Note-se que os dados aqui apresentados foram extraídos da documentação oficial de cada esquema de assinatura digital.

	Public key	**Signature**
qTESLA-p-I	14 880	2 592
qTESLA-p-III	38 432	5 664

Tabela 6: Tamanho (em *bytes*) da chave pública e da assinatura do qTESLA.

	Public key	**Signature**
Dilithium Medium	1 184	2 044
Dilithium Recommended	1 472	2 701
Dilithium Very High	1 760	3 366

Tabela 7: Tamanho (em *bytes*) da chave pública e da assinatura do Dilithium.

Os gráficos que se seguem são também referentes ao tamanho da chave pública e da assinatura, com a diferença de as comparações estarem expostos de acordo com o nível de segurança do NIST.

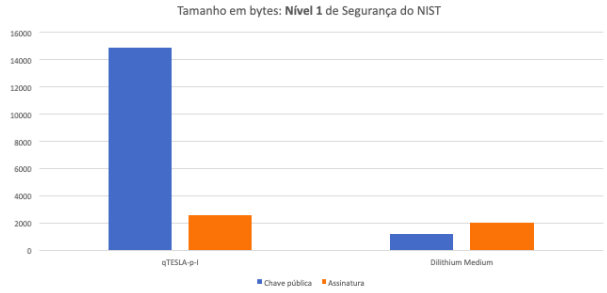

Figura 6: Comparação de tamanhos da chave pública e da assinatura para o Nível 1 de Segurança da NIST.

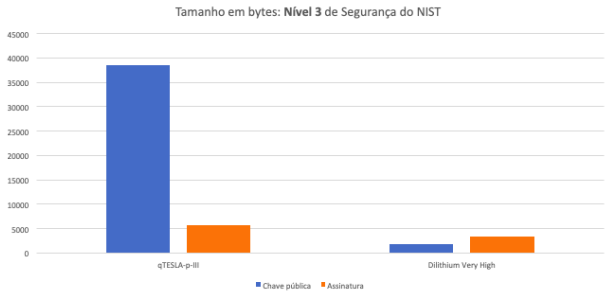

Figura 7: Comparação de tamanhos da chave pública e da assinatura para o Nível 3 de Segurança da NIST.

6.3.3 *Desempenho*

Antes de mais, a análise aqui feita tem por base as implementações oficiais dos esquemas de assinatura digital, disponíveis em linguagem **C**.

O desempenho dos algoritmos dos esquemas é dado em quilociclos de CPU e os dados recolhidos resultam precisamente da execução das implementações oficiais dos esquemas

em estudo. Para tal recorreu-se a uma máquina virtual cujo sistema operativo era o Linux (factor decisivo para conseguir correr facilmente o qTESLA) e cujo modelo do processador era: Intel (R) Core (TM) i7-8750 HCPY @ 2208 000 MHz.

As tabelas que se seguem apresentam agora o desempenho da execução das operações de assinatura, verificação e geração de chave para os diferentes esquemas de assinatura digital.

Antes de mais, note-se que existe para cada esquema de assinatura em análise uma implementação adicional que consiste numa implementação otimizada e que suporta um conjunto de instruções AVX2.

Adicionalmente, para o Dilithium existe, desde a segunda ronda do concurso PQCS, uma variante denominada por Dilithium-AES, em que se usa o AES-256 no modo contador em vez do SHAKE.

	Keygen	**Sign**	**Verify**
qTESLA-p-I	1 587 957	1 909 383	540 612
qTESLA-p-III	8 742 961	4 325 172	1 383 681

Tabela 8: Desempenho do qTESLA-p-I e qTESLA-p-III.

	Keygen	**Sign**	**Verify**
qTESLA-p-I (AVX2)	1 529 030	1 104 597	465 252
qTESLA-p-III (AVX2)	8 696 982	2 513 264	1 189 245

Tabela 9: Desempenho do qTESLA-p-I (AVX2) e qTESLA-p-III (AVX2).

	Keygen	**Sign**	**Verify**
Dilithium Medium	247 574	1 840 311	303 909
Dilithium Recommended	357 518	2 640 609	397 871
Dilithium Very High	486 683	2 394 489	542 039

Tabela 10: Desempenho do *Dilithium Medium*, *Recommended* e *Very High*.

	Keygen	Sign	Verify
Dilithium Medium (AVX2)	51 953	215 359	58 505
Dilithium Recommended (AVX2)	83 852	291 948	80 581
Dilithium Very High (AVX2)	105 929	285 563	113 418

Tabela 11: Desempenho do *Dilithium Medium* (AVX2), *Recommended* (AVX2) e *Very High*(AVX2).

	Keygen	Sign	Verify
Dilithium Medium-AES (AVX2)	33 997	157 307	41 026
Dilithium Recommended-AES (AVX2)	44 047	227 088	51 485
Dilithium Very High-AES (AVX2)	55 452	195 270	64 894

Tabela 12: Desempenho do *Dilithium Medium*-AES (AVX2), *Recommended*-AES (AVX2) e *Very High*-AES (AVX2).

Os gráficos que se seguem são também referentes ao desempenho da execução das operações de assinatura, verificação e geração de chaves, com a diferença de as comparações estarem expostas de acordo com o nível de segurança do NIST.

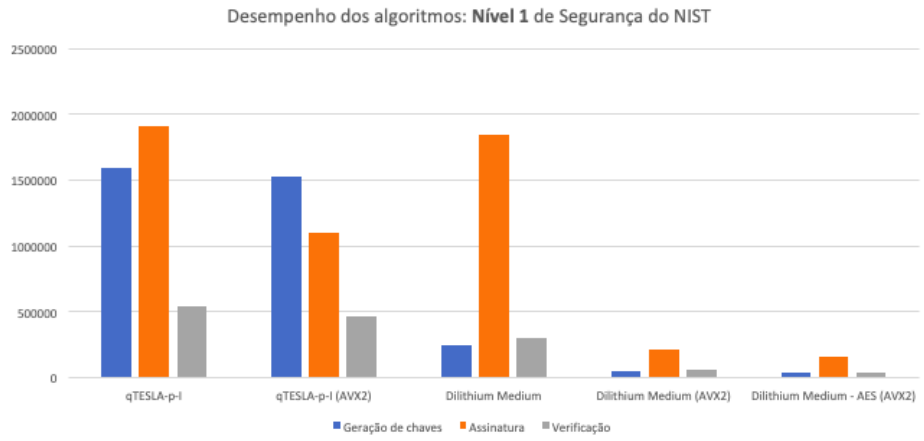

Figura 8: Comparação do desempenho dos algoritmos para o Nível 1 de Segurança da NIST.

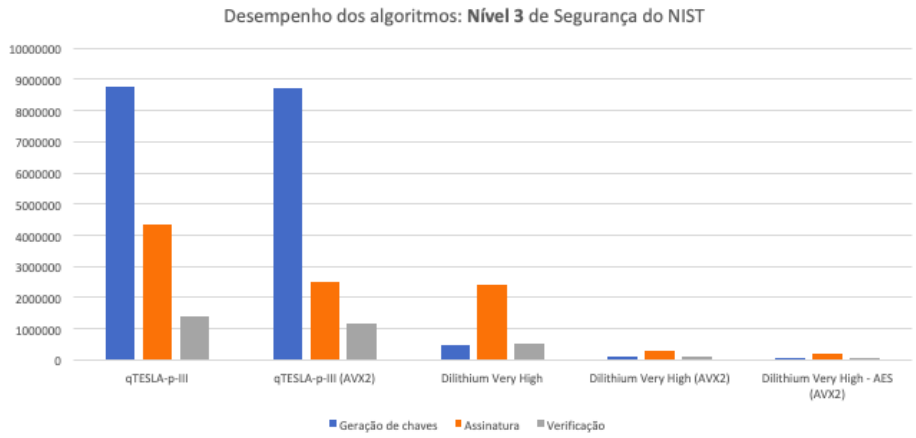

Figura 9: Comparação do desempenho dos algoritmos para o Nível 3 de Segurança da NIST.

6.3.4 *Discussão*

De modo geral, escolher um melhor esquema de assinatura digital depende de como o mesmo será aplicado e qual a maior necessidade: segurança, desempenho, etc.

Mais, o factor flexibilidade do algoritmo, isto é, a existência de diferentes parâmetros, adaptabilidade do código, etc, é muito importante devido às possíveis adaptações dos algoritmos.

Posto isto, observe-se primeiramente que nenhum dos esquemas de assinatura digital em estudo abrange todos os níveis de segurança propostos pelo NIST, sendo o Dilithium o mais completo, por apresentar três níveis ao invés de dois, como o qTESLA.

Observando a Figura 6 e a Figura 7, verifica-se que o Dilithium apresenta, para os níveis de segurança 1 e 3 do NIST, o menor tamanho dos parâmetros (soma da chave pública e da assinatura) em comparação com o qTESLA.

Analisando agora a Figura 8 e a Figura 9, verifica-se que, para os níveis de segurança 1 e 3 do NIST, o menor tempo consumido em ciclos de CPU para as três operações básicas de assinatura digital, pertence, na sua maioria, ao Dilithium.

Pensando agora em cada esquema de assinatura digital isoladamente, observe-se que o qTESLA, em geral, é mais lento para gerar as chaves, porém mais rápido para assinar e verificar, sendo este um factor muito desejável na maioria das aplicações. O Dilithium é mais lento para assinar do que para gerar as chaves. Contudo, analisando diretamente o desempenho dos esquemas em questão, é notório que o desempenho do Dilithium supera o desempenho do qTESLA com uma grande margem, quer na geração das chaves, assinatura ou verificação.

Quanto às demonstrações de segurança, tanto o qTESLA como o Dilithium apresentam demonstrações de segurança diretas baseadas em reduções de problemas.

Segue-se uma síntese com os vantagens e desvantagens dos esquemas de assinatura digital estudados:

	Vantagens	Desvantagens
Crystals-Dilithium	• É seguro contra ataques no modelo SUF-CMA e, consequentemente no modelo EUF-CMA. • Segurança no modelo do oráculo aleatório clássico e quântico. • Maioritariamente mais rápido nas três operações básicas de assinatura digital. • Abrange mais níveis de segurança. • Melhor desempenho, de modo geral.	• Mais lento para assinar, do que para gerar as chaves.
qTESLA	• Diversidade de *design*. • É mais lento para gerar as chaves, porém mais rápido para assinar e verificar. • Seguro contra ataques no modelo EUF-CMA.	• Não é seguro contra ataques no modelo SUF-CMA. • Menor desempenho, de modo geral.

Tabela 13: Vantagens e desvantagens do Dilithium e do qTESLA

Para finalizar, é possível notar que, quer em relação ao tamanhos dos parâmetros, quer em relação ao desempenho dos algoritmos, o Dilithium destacava-se consideravelmente em relação ao qTESLA. Adicionalmente, o mesmo possuía mais um nível de segurança e era seguro contra ataques no modelo SUF-CMA. Por conseguinte, sendo estes esquemas de assinatura digital 'adversários diretos', pelo facto da segurança de ambos estar fundamentada no mesmo problema, não é de estranhar que o Dilithium tenha prevalecido em relação ao qTESLA na terceira ronda do concurso PQCS.

Mais, o motivo pelo qual o qTESLA foi desclassificado na terceira ronda é apresentado na documentação oficial da NIST, disponível no seguinte *site*: https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8309.pdf, e resumidamente, consiste no facto de o "desempenho do qTESLA não ser forte o suficiente para permanecer em competição". Pois, os tamanhos de chave pública do q-TESLA-p-I e q-TESLA-p-III eram cerca de 15 a 20 vezes

maiores do que os dos esquemas de assinaturas FALCON e Dilithium, e os tamanhos de assinatura também eram superiores. Ao comparar as contagens de ciclo necessárias para gerar a assinatura e efectuar a verificação, qTESLA também era aproximadamente 2 a 5 vezes mais lento que FALCON e Dilithium.

7

CERTIFICADOS E CRIPTOGRAFIA PÓS-QUÂNTICA

Como já aqui foi mencionado, prevê-se a evolução dos computadores quânticos, o que por sua vez leva a preocupações com a criptografia de chave pública como a conhecemos. Os computadores quânticos representam uma ameaça aos algoritmos e sistemas PKI atuais. Entre outras técnicas criptográficas, estes colocam em risco o uso dos certificados PKI X.509 usados actualmente para autenticação.

As infra-estruturas de chave pública suportam criptografia de chave pública, manipulando chaves e fornecendo certificados de chave pública. A abordagem mais comum é o uso de PKIs hierárquicas, em que os certificados são emitidos pelas autoridades de certificação (CAs) de acordo com o *standard* X.509. Esses certificados vinculam a identidade do proprietário da chave à sua chave pública e, portanto, permitem a autenticação de chaves públicas. Esse é um pré-requisito básico para o uso de assinaturas digitais e criptografia de chave pública em aplicações que requerem comunicação eletrónica segura. Um exemplo é a comunicação segura na *internet* usando o protocolo TLS.

A criptografia pós-quântica precisa assim de ser integrada. E, de modo a se conseguir garantir segurança criptográfica ininterrupta, é importante iniciar a transição para a criptografia pós-quântica o mais depressa possível.

Para facilitar a transição, a infraestrutura criptográfica também deve ser adaptada. Uma abordagem para uma transição segura e "confortável" é o uso de híbridos: vários algoritmos em paralelo que são combinados de forma a que o esquema híbrido seja seguro desde que pelo menos um dos algoritmos usados em paralelo seja seguro. Para tal, um esquema clássico é combinado com um esquema pós-quântico, o que possui várias vantagens em comparação com uma mudança direta para os algoritmos pós-quânticos, como as seguintes:

- Alcança segurança e funcionalidade de maneira pós-quântica consciente e compatível com as versões anteriores de *software* ainda não atualizado;
- Simplifica as transferências de PKI e sistemas dependentes integrando flexibilidade nos sistemas já existentes;
- Alcança conformidade contínua com os *standards* X.509;
- Reduz os custos de substituição de sistemas confiáveis e de PKI, pois não necessita de nenhum sistema em particular para usar certificados híbridos.

7.1 CERTIFICADOS HÍBRIDOS

Os certificados híbridos são a base para o uso da criptografia híbrida. Eles são assinados em paralelo com dois esquemas de assinatura diferentes e vinculam adicionalmente duas chaves públicas independentes (uma clássica e uma pós-quântica) a uma identidade. Assim, a autenticação das chaves públicas contidas no certificado é protegida com a segurança combinada dos dois esquemas de assinatura. Além disso, o uso paralelo seguro de dois esquemas diferentes para fins de autenticação e troca de chaves é ativado.

Esses certificados híbridos são totalmente compatíveis com o *standard* X.509. Os campos da assinatura e da chave pública do certificado X.509 são usados para um dos esquemas de assinatura. Para a transição pós-quântica, os campos *standard* são usados para o esquema clássico. Tal permite compatibilidade com clientes que não suportam assinaturas híbridas. O segundo esquema de assinatura, por exemplo, o Dilithium, é integrado usando duas extensões X.509 não críticas. Uma das extensões contém a chave pública associada ao segundo esquema, enquanto a outra contém a segunda assinatura nos dados certificados. Para oferecer suporte completo a entidades herdadas de maneira controlada, a extensão que contém a segunda chave pública pode, opcionalmente, ficar de fora. Tal implica que a entidade certificada ainda não ofereça suporte a esquemas pós-quânticos, enquanto o conteúdo do certificado ainda é protegido de maneira híbrida.

Embora os *standards* finais de criptografia pós-quântica ainda não tenham sido aprovados, a experimentação de certificados híbridos, com criptografia quântica segura, vem permitir que as organizações deem o primeiro passo para a criptografia pós-quântica e comecem assim a trabalhar naquele que será o futuro da criptografia.

A título de exemplo, a empresa norte-americana DigiCert, que é um dos colaboradores do concurso de criptografia pós-quântica da NIST, já disponibilizou um kit de teste PQC para os seus clientes que desejem experimentar o processo de instalação do certificado híbrido RSA/PQC (TLS ou IoT). Esse kit inclui todos os comandos apropriados para gerar certificados híbridos, que contêm as chaves RSA/ECC compatíveis com versões anteriores, bem como futuras chaves pós-quânticas compatíveis usando o algoritmo Crystals-Dilithium.

7.2 EXPERIMENTAÇÃO DE CERTIFICADOS HÍBRIDOS COM CRIPTOGRAFIA PÓS-QUÂNTICA

Nesta secção faz-se a experimentação de certificados híbridos com os esquemas de assinatura qTESLA e Dilithium, tendo a seguinte página de *github*: `https://github.com/open-quantum-safe/openssl` como referência.

Recorre-se ao **OpenSSL 1.1.1**, que permite a troca da chave quântica segura e adiciona algoritmos de autenticação usando liboqs (uma biblioteca C de código aberto para algoritmos criptográficos pós-quânticos) para prototipagem e avaliação.

Antes de mais, note-se que as etapas que se seguem foram efectuadas no sistema operativo **macOS 10.15**.

7.2.1 *Construção*

Começou-se por obter os pré-requisitos necessários, e, para tal, instalou-se os seguintes pacotes usando o brew:

```
brew install cmake ninja libtool openssl@1.1
```

Em seguida, obteve-se o código-fonte do *fork open-quantum-safe/openssl*, sendo que <OPENSSL_DIR> é um diretório à escolha.

```
git clone --branch OQS-OpenSSL_1_1_1-stable
https://github.com/open-quantum-safe/openssl.git <OPENSSL_DIR>
```

Posteriormente, for necessário construir e instalar a biblioteca liboqs. Os comandos que se seguem permitiram precisamente baixar e construir a biblioteca liboqs e, em seguida, instalá-la num sub-diretório dentro da pasta OpenSSL.

```
git clone --branch master https://github.com/open-quantum-safe/liboqs.git
cd liboqs
mkdir build && cd build
cmake -GNinja -DCMAKE_INSTALL_PREFIX=<OPENSSL_DIR>/oqs ..
ninja
ninja install
```

Para finalizar esta secção, construiu-se o *fork* já mencionado, seguindo as instruções *standard* para construir OpenSSL.

Em <OPENSSL_DIR> efectuou-se os seguintes comandos:

```
./Configure no-shared darwin64-x86_64-cc
make -j
```

7.2.2 *Geração de certificados híbridos*

Nesta subsecção é possível gerar um certificado X.509, usando um algoritmo clássico, um algoritmo *quantum-safe* ou um algoritmo híbrido, consoante os algoritmos suportados que estão listados no seguinte *site*: https://github.com/open-quantum-safe/openssl#supported-algorithms.

Assim, para gerar um certificado *root* CA auto-assinado usou-se o seguinte comando, substituindo <SIG> pelos algoritmos escolhidos, nomeadamente, pelos algoritmos híbridos $rsa3072\_dilithium2$, $p256\_dilithium2$, $p384\_dilithium4$, $p256\_qteslapi$ e $p384\_qteslapiii$.

```
apps/openssl req -x509 -new -newkey <SIG> -keyout <SIG>_CA.key -out
<SIG>_CA.crt -nodes -subj "/CN=oqstest CA" -days 365 -config apps/openssl.cnf
```

Note-se que os algoritmos híbridos combinam um algoritmo quântico seguro com um algoritmo de assinatura digital tradicional tendo em conta as seguintes condições:

- se <SIG> tiver segurança NIST 1, então o *fork* em questão disponibiliza os métodos $rsa3072\_$<SIG> e $p256\_$<SIG>, que combinam <SIG> com RSA3072 e com ECDSA usando a curva P256 do NIST, respectivamente.
- se <SIG> tiver segurança NIST 3, o *fork* disponibiliza o método $p384\_$<SIG>, que combina <SIG> com ECDSA usando a curva P384 do NIST.

Considerando estas condições, são várias as combinações que se podem fazer de modo a conseguir gerar diversos certificados híbridos com criptografia pós-quântica. Mais, para consultar a lista completa dos algoritmos suportados pode-se realizar o seguinte comando:

```
apps/openssl list -public-key-algorithms
```

Optou-se por gerar os certificados híbridos já mencionados, sendo que os mesmos são apresentados no Apêndice C, Apêndice D e Apêndice E, Apêndice F e Apêndice G.

Por curiosidade fez-se também a recolha do tamanho desses certificados, efectuando os seguintes comandos:

```
(base) ZitaAbreu@Air-de-Maria New % ls -l rsa3072_dilithium2_CA.crt
-rw-r--r--@ 1 ZitaAbreu  staff  5819 13 Ago 16:57 rsa3072_dilithium2_CA.crt

(base) ZitaAbreu@Air-de-Maria New % ls -l p256_dilithium2_CA.crt
-rw-r--r--@ 1 ZitaAbreu  staff  4946 13 Ago 16:08 p256_dilithium2_CA.crt

(base) ZitaAbreu@Air-de-Maria New % ls -l p384_dilithium4_CA.crt
-rw-r--r--@ 1 ZitaAbreu  staff  7603 13 Ago 17:10 p384_dilithium4_CA.crt

(base) ZitaAbreu@Air-de-Maria NEW % ls -l p256_qteslapi_CA.crt
-rw-r--r--@ 1 ZitaAbreu  staff  24634 28 Ago 10:52 p256_qteslapi_CA.crt

(base) ZitaAbreu@Air-de-Maria NEW % ls -l p384_qteslapiii_CA.crt
-rw-r--r--@ 1 ZitaAbreu  staff  61330 28 Ago 10:53 p384_qteslapiii_CA.crt
```

Observou-se assim que o tamanho do certificado híbrido $rsa3072\_dilithium2$ ronda os 6 *kilobytes*, o $p256\_dilithium2$ os 5 *kilobytes*, $p384\_dilithium4$ os 8 *kilobytes*, o $p256\_qteslapi$ os 25 *kilobytes* e o $p384\_qteslapiii$ os 61 *kilobytes*.

Para finalizar, note-se que existe uma grande discrepância entre o tamanho dos certificados híbridos que recorrem ao Dilithium e o tamanho dos certificados que utilizam o qTESLA como esquema de assinatura digital pós-quântico. O tamanho dos certificados que recorrem ao qTESLA é muito superior, sendo expectável assim que tal provoque muitos mais problemas de compatibilidade em *software* "legacy".

8

CONCLUSÃO

A computação pós-quântica vem representar uma grande ameaça à presente criptografia. Prevê-se que um computador quântico estável seja capaz de derrubar, em tempo útil, as primitivas clássicas atuais. Em resposta a este problema, surgiu a criptografia pós-quântica, o ramo da criptografia que estuda classes de algoritmos criptográficos resistentes a ataques quânticos.

É com base neste problema, associado à necessidade de padronizar algoritmos pós-quânticos, que se assume, como objetivo primordial desta dissertação, a análise de dois esquemas de assinatura digital, o qTESLA e o Crystals-Dilithium.

Para tal, foi necessário efetuar o estudo dos fundamentos matemáticos que suportam esses dois esquemas de assinatura e, em particular, os problemas difíceis que os alicerçam, nomeadamente, o problema LWE e o problema SIS.

Tanto o problema LWE como o SIS são problemas difíceis em reticulados e cuja dificuldade está na essência da chamada criptografia *lattice-based*. Resumidamente, o problema LWE consiste em, dada uma sequência aleatória de equações lineares 'aproximadas' em s, encontrar um segredo $s \in \mathbb{Z}_q^n$. Já o problema SIS é conhecido por ser o "dual" do problema anterior e consiste em dada uma sequência de vetores $a_1, a_2, \ldots, a_n$, escolhidos uniformemente em $\mathbb{Z}_q^n$, encontrar uma combinação linear com coeficientes pequenos cuja soma seja $0 (\mathrm{mod} q)$. As variantes destes problemas e que são indispensáveis para os esquemas de assinatura Dilithium e qTESLA são o R-LWE, M-LWE e M-SIS.

Realizou-se, no seguimento do estudo individual dos dois esquemas de assinatura digital, uma comparação entre eles. Os critérios de comparação foram baseados nas operações básicas (geração de chaves, assinatura e verificação), nos aspetos técnicos (implementação) e na segurança. E, tendo em consideração todos estes critérios, foi natural destacar, por possuir menor tamanho de parâmetros, um melhor desempenho dos algoritmos, um nível extra de segurança e a capacidade de ser seguro contra ataques no modelo SUF-CMA, o esquema de assinatura Dilithium. O facto de este esquema apresentar todas estas vantagens comparativamente ao qTESLA está em total conformidade com o facto de o Dilithium ter sido aceite para a terceira ronda do concurso PQCS, ao invés do qTESLA.

O trabalho empírico envolveu, para além de uma implementação não otimizada de ambos os esquemas de assinatura digital, no *software SageMath*, a experimentação de certificados híbridos com os esquemas de assinatura pós-quânticos em questão.

Dessa experimentação de certificados híbridos efetuaram-se algumas observações, nomeadamente, constatou-se que os certificados híbridos que resultam da utilização do Dilithium possuem um menor tamanho em contraste com os que recorrem ao qTESLA, sendo expectável que tal permita uma melhor compatibilidade com as versões anteriores de *software* ainda não atualizado.

Para finalizar, este estudo constituiu apenas um contributo para o conhecimento daquele que será o futuro da criptografia, a criptografia pós-quântica e, em especial, para os esquemas de assinatura digital pós-quânticos. Dada a importância do tema e consequente urgência na transição para a criptografia pós-quântica, face à rápida evolução dos computadores quânticos, considera-se que muito há ainda para percorrer no campo da investigação nesta área e no processo de transição para a criptografia pós-quântica, sendo portanto, este um campo fértil para futuros trabalhos.

BIBLIOGRAFIA

[1] Bernstein, Daniel J., Buchmann, Johannes, Dahmen, Erik, *Post-Quantum Cryptography*, Springer, 2009.

[2] Digicert, *Report on Post-Quantum Cryptography*, https://www.digicert.com/pt/post-quantum-cryptography/ (acedido em Março de 2020).

[3] Chen, L., Jordan, S., Yi-Kai Liu, Moody, D., Peralta, R., Perlner, R., Smith-Tone, D., *Report on Post-Quantum Cryptography*, NIST, 2016 https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf (acedido em Março de 2020).

[4] *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*, https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf (acedido em Março de 2020).

[5] NIST, *Post-Quantum Cryptography*, https://csrc.nist.gov/projects/post-quantum-cryptography (acedido em Março de 2020).

[6] Bindel, N. *et al. Submission to NIST's post-quantum project (2nd round): lattice-based digital signature scheme qTESLA* , NIST, 2019.

[7] Ducas, L. *et al. CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation*, NIST, 2019.

[8] Peikert, C., *A Decade of Lattice Cryptography*, Foundations and Trends, Volume 10, 2016.

[9] Peikert, C., *Lattice-Based Cryptography: Short Integer Solution (SIS) and Learning With Errors (LWE)*, Georgia Institute of Technology, 2013.

[10] Washington, Lawrence C., *Introduction to Cyclotomic Fields. Graduate Texts in Mathematics.* Springer, 1997.

[11] Milne, James S., *Fields and Galois Theory*, www.jmilne.org/math/ (acedido em Setembro de 2020).

[12] Forest, S., Gosset, D., Kliuchnikov, V. e McKinnon, D., *Exact Synthesis of Single-Qubit Unitaries Over Clifford-Cyclotomic Gate Sets*, Journal of Mathematical Physics 56, 2015.

[13] Ding, J. e Lindner, R., *Identifying Ideal Lattices*, IACR Cryptology ePrint Archive, 2007.

[14] Ajtai, M., *Generating Hard Instances of Lattice Problems*, Electronic Colloquium on Computational Complexity, 1996.

[15] Miller, D., Narayanan, B. e Venkatesan, R., *Coppersmith's lattices and "focus groups": an attack on small-exponent RSA*, IACR Cryptology ePrint Archive, 2020.

[16] Regev, O., *The Learning with Errors Problem*, 2010 IEEE 25th Annual Conference on Computational Complexity, 2010.

[17] Crockett, E., Paquin, C., Stebila, D., *Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH*, Cryptology ePrint Archive: Artigo 2019/858, 2019.

[18] Micciancio, D, *Generalized compact knapsacks, cyclic lattices, and efficient oneway functions*, Computational Complexity, Springer 2007.

[19] Regev, O., *On Lattices, Learning with Errors, Random Linear Codes, and Cryptography*, Journal of the ACM 56(6):84-93, 2009.

[20] Kim, Y., Koo, Z., Seon, J., *Reduction from Module-SIS to Ring-SIS Under Norm Constraint of Ring-SIS*, IEEE Access, 2020.

[21] Bai, S. e Galbraith, S., *An improved compression technique for signatures based on learning with errors.* Josh Benaloh, editor, Topics in Cryptology – CT- RSA 2014, volume 8366 of Lecture Notes in Computer Science, pages 28–47, San Francisco, CA, USA, February 25–28, 2014. Springer, Heidelberg, Germany.

[22] Alkim, E., Bindel, N., Buchmann, J., Dagdelen, O., Eaton, E., Gutoski, G., Krämer, J. e Pawlega, F., *Revisiting TESLA in the quantum random oracle model*, Tanja Lange and Tsuyoshi Takagi, editors, Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017, volume 10346 of LNCS, pages 143–162. Springer, 2017.

[23] Alkim, E., Barreto, P., Bindel, N., Kramer, J., Longa, P. e Ricardini, J., *The Lattice-Based Digital Signature Scheme qTESLA*, 2019

[24] Longa, P., *The post-quantum signature scheme qTESLA*, International Cryptographic Module Conference (ICMC'19), 2019.

[25] Bergami, F., *Lattice-Based Cryptography*, Université de Bordeaux, 2016.

[26] Applebaum, B., Cash, D., Peikert, C. e Sahai, A., *Fast Cryptographic Primitives and Circular - Secure Encryption Based on Hard Learning Problems.* Advances in Cryptology -CRYPTO2009, 2009.

[27] Lyubashevsky, V., Peikert, C., e Regev, O. *On ideal lattices and learning with errors over rings.* Advances in Cryptology EUROCRYPT, 2010.

[28] Nejatollahi, H., Dutt, N., e Ray, S. *Software and Hardware Implementation of Lattice-based Cryptography Schemes.* Center For Embedded And Cyber-Physical Systems, 2017.

[29] Micciancio, D. e Peikert, C. *TrapdoorsTrapdoors for lattices: Simpler, tighter, faster, smaller*. EUROCRYPT, volume 7237 of Lecture Notes in Computer Science, páginas 700–718. Springer, 2012. ISBN 9783642290107.

[30] Brakerski, Z., Langlois, A., Peikert, C., Regev, O. e Stehle, D., *Classical hardness of learning with errors*, STOC, pages 575- 584. ACM, 2013. ISBN 9781450320290.

[31] Brakerski, Z., Langlois, A., Peikert, C., Regev, O., e Stehlé, D. *Classical Hardness of Learning with Errors*, Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing, STOC, 2013.

[32] Barreto, P., Longa, P., Naehrig, M., Jefferson E. Ricardini, e Zanon, G. *Sharper ring-LWE signatures*. Cryptology ePrint Archive, Report 2016/1026, 2016. http://eprint.iacr.org/2016/1026/ (acedido em Abril de 2020).

[33] Eaton, E., *Leighton-Micali Hash-Based Signatures*, https://datatracker.ietf.org/doc/rfc8554/ (acedido em Abril de 2020).

[34] NIST, *Request for Public Comments on Stateful Hash-Based Signatures* (HBS), https://csrc.nist.gov/news/2019/stateful-hbs-request-for-public-comments (acedido em Abril de 2020).

[35] Sjöberg, M., *Post-quantum algorithms for digital signing in Public Key Infrastructures, Master in Computer Science*, KTH Royal Institute of Technology, 2017.

[36] NIST, *Post-Quantum Cryptography (PQC)*, https://csrc.nist.gov/projects/post-quantum-cryptography (acedido em Abril de 2020).

[37] ETSI, *Quantum-Safe Cryptography*, https://www.etsi.org/technologies/quantum-safe-cryptography#mytoc2?jjj=1576684950835 (acedido em Abril de 2020).

[38] qTESLA ,*qTesta*, https://qtesla.org (acedido em Abril de 2020).

[39] CRYSTALS, *Cryptographic Suite for Algebraic Lattices*, https://pq-crystals.org (acedido em Abril de 2020).

[40] Green, M., *Thoughts on Cryptographic Engineering*, https://blog.cryptographyengineering.com/euf-cma-and-suf-cma/ (acedido em Maio de 2020).

[41] Ding, J., Xie, X., Lin, X., *A Simple Provably Secure Key Exchange Scheme Based on the Learning with Errors Problem.*, IACR Cryptol. ePrint Arch, 2017.

[42] Micciancio, D. e Mol, P., *Pseudorandom Knapsacks and the Sample Complexity of LWE Search-to-Decision Reductions*, Springer ,2011.

[43] Fitzpatrick, R., Gopfert, F. and Albrecht, M., *On the Efficacy of Solving LWE by Reduction to Unique-SVP*, 2013.

[44] Micciancio, D., Peikert, C., *Hardness of SIS and LWE with Small Parameters*, Springer, 2013.

[45] Bai, S. and Galbraith, S., *Lattice Decoding Attacks on Binary LWE*, University of Auckland, 2013.

[46] Hoffstein, J., Pipher, J., Silverman, J., *An Introduction to Mathematical Cryptography*, Springer, 2008.

[47] Esslinger, B., Development Team of Cryptool Software, *Learning and Experiencing Cryptography with CrypTool and SageMath*, CrypTool Project, 2017.

[48] Bindel, N., Braun, J., Gladiator, L., Stöckert, T., e Wirth1, J., *X.509-Compliant Hybrid Certificates for the Post-Quantum Transition*, Technische Universität Darmstadt, Germany, 2019.

[49] Kampanakis, P., Panburana, P., Daw, E. e Geest, D., *The Viability of Post-Quantum X.509 Certificates*, IACR Cryptology ePrint Archive, 2018.

[50] Michael Rose, *Lattice-based cryptgraphy: a practical implementation.* Master's thesis, University of Wollongong, 2011.

[51] Joseph H. Silverman. *An introduction to the theory of lattices and applications to cryptography*, University of Wyoming, 2006.

A

IMPLEMENTAÇÃO DO qTESLA

```
from sage.stats.distributions.discrete_gaussian_polynomial import
    DiscreteGaussianDistributionPolynomialSampler

class qTesla:
    def __init__(self):
        self.q = 343576577
        self.n = 1024
        self.d = 22
        self.B = 2^19 - 1
        self.Le = 554
        self.Ls = 554
        self.sigma = 8.5
        self.h = 25
        self.alfa = self.sigma/self.q


    #FUNCOES AUXILIARES:

    # Least significant bits
    def Lsb(self,val):
        R = IntegerModRing(self.q)
        x = mod(R(val),2^self.d)
        return int(x)

    # Most significant bits
    def Msb(self,val):
        R = IntegerModRing(self.q)
        x = self.Lsb(val)
        y = (R(val) - x)/2^self.d
        return int(y)

    # Funcao Hash
    def Hash(self,val):
        H = []
        cont = 0
        cont_num = 0
        for i in range(0,self.n,2):
            u = val[i] + val[i+1]
```

```
38              cont += 1
                if u == '11':
40                  H.append(0)
                if u == '01':
42                  H.append(1)
                    cont_num += 1
44              if u == '00':
                    H.append(0)
46              if u == '10':
                    H.append(-1)
48                  cont_num += 1
                if cont_num > self.h or cont_num < self.h:
50                  break
            for i in range(self.n - cont):
52              H.append(0)
            return H
54
    # Funcao auxiliar da funcao infinite_norm
56  def pol_aux(self,w,n):
        Zx.<x> = ZZ[]
58      r = int((n-1)//2)
        return Zx(map(lambda x: lift(x + r) - r , w.list()))
60
    # Norma infinita
62  def infinite_norm(self,p,n):
        #R = self.pol_aux(p,n)
64      J = p.list()
        for i in range(len(J)):
66          J[i] = abs(int(J[i]))
        return int(max(J))
68
    # Funcao que verifica se o polinomio p e T-well-rounded
70  def well_rounded(self,T,p):
        rounded = False
72      norm = self.infinite_norm(p,self.q)
        lista = []
74      u = p.list()
        for i in range(len(u)):
76          k = self.Lsb(u[i])
            u[i] = abs(int(k))
78      normL = max(u)
        if norm <= floor(self.q/2)-T and normL <= 2^(self.d-1) - T:
80          rounded = True
        return rounded
82
    # FUNCOES PRINCIPAIS
84


86  def Keygen(self):
        Zx.<x> = ZZ[]
88      R.<x> = Zx.quotient(x^self.n+1)
```

```
        Zq.<z> = PolynomialRing(GF(self.q))
90      Rq.<z> = Zq.quotient(z^self.n+1)

92      # PASSO 1
        e = DiscreteGaussianDistributionPolynomialSampler(R, self.n, self
    .sigma)()
94      self.eq = DiscreteGaussianDistributionPolynomialSampler(Rq, self.
    n, self.sigma)()
        s = DiscreteGaussianDistributionPolynomialSampler(R, self.n, self
    .sigma)()
96      self.sq = DiscreteGaussianDistributionPolynomialSampler(Rq, self.
    n, self.sigma)()

98      # PASSO 2
        E = e.list()
100     for i in range(len(E)):
            E[i] = abs(int(E[i]))
102     S = s.list()
        for i in range(len(S)):
104         S[i] = abs(int(S[i]))
        E.sort()
106     S.sort()
        E = E[self.n-self.h:]
108     S = S[self.n-self.h:]
        while sum(E) >= self.Le or sum(S) >= self.Ls:
110         e = DiscreteGaussianDistributionPolynomialSampler(R, self.n,
    self.sigma)()
            self.eq = DiscreteGaussianDistributionPolynomialSampler(Rq,
    self.n, self.sigma)()
112         s = DiscreteGaussianDistributionPolynomialSampler(R, self.n,
    self.sigma)()
            E = e.list()
114         for i in range(len(E)):
                E[i] = abs(int(E[i]))
116         S = s.list()
            for i in range(len(S)):
118             S[i] = abs(int(S[i]))
            E.sort()
120         S.sort()
            E = E[self.n - self.d:]
122         S = S[self.n - self.d:]

124     # PASSO 3
        a = Rq.random_element()
126     while not a.is_unit():
            a = Rq.random_element()
128     self.aq = a

130     t = self.aq*self.sq + self.eq

132     # PASSO 4 E 5
```

```
        self.PubKey = (a,t)
134     self.PrivKey = (s,e)


136

    def Sign(self,m):
138     Zx.<x> = ZZ[]
        R.<x> = Zx.quotient(x^self.n+1)
140     Zq.<z> = PolynomialRing(GF(self.q))
        Rq.<z> = Zq.quotient(z^self.n+1)
142
        # PASSO 1
144     r = []
        for i in range(self.n):
146         r.append(randint(1,self.B))
        rq = Rq(r)
148     r = R(r)

150     # PASSO 2
        a = self.PubKey[0]
152     u = self.aq*rq
        u = u.list()
154     string = ""
        l = m[2:]
156     string = string + l
        for i in range(self.n):
158         u[i] = self.Msb(u[i])
            l = bin(u[i])
160         if u[i] >= 0:
                string = string + l[2:]
162         if u[i] < 0:
                string = string + l[3:]
164     c = self.Hash(string)
        cq = Rq(c)
166     cR = R(c)

168     # PASSO 3
        s = self.PrivKey[0]
170     e = self.PrivKey[1]

172     z = rq+self.sq*cq

174     # PASSO 4, 5 E 6
        wb = self.aq*rq - self.eq*cq
176     while self.infinite_norm(z,self.q) + self.Ls >= self.B and self.
    well_rounded(self.Le,wb) == False:
            r = []
178         for i in range(self.n):
                r.append(randint(1,self.B))
180         rq = Rq(r)
            r = R(r)
182         a = self.PubKey[0]
```

```
            u = self.aq*rq
            u = u.list()
            string = ''
            l = m[2:]
            string = string + l
            for i in range(self.n):
                u[i] = self.Msb(u[i])
                l = bin(u[i])
                if u[i] >= 0:
                    string = string + l[2:]
                if u[i] < 0:
                    string = string + l[3:]
            c = self.Hash(string)
            cq = Rq(c)
            cR = R(c)
            s = self.PrivKey[0]
            e = self.PrivKey[1]
            z = rq + self.sq*cq
            wb = self.aq*rq-self.eq*cq
        sign = (z,c)
        return sign



    def Verify(self,m,sign):
        z = sign[0]
        c = sign[1]
        a = self.PubKey[0]
        t = self.PubKey[1]


        Zx.<x> = ZZ[]
        R.<x> = Zx.quotient(x^self.n+1)
        Zq.<z> = PolynomialRing(GF(self.q))
        Rq.<z> = Zq.quotient(z^self.n+1)

        # PASSO 1
        if self.infinite_norm(z,self.q) + self.Ls >= self.B:
            print ("Assinatura rejeitada!")

        # PASSO 2
        cq = Rq(c)
        w = self.aq*z - t*cq

        # PASSO 3 E 4
        u = w.list()
        string = ""
        l = m[2:]
        string = string + l
        for i in range(self.n):
            u[i] = self.Msb(u[i])
```

```
234             k = bin(u[i])
                if u[i] >= 0:
236                 string = string + l[2:]
                if u[i] < 0:
238                 string = string + l[3:]
        Hash = self.Hash(string)
240     if c != Hash:
            print ("Assinatura rejeitada!")
242     else:
            print ("Assinatura aceite!")
```

```
1 In: Teste = qTesla()
      Teste.Keygen()
3     m = bin(123456789)
      sign = Teste.Sign(m)
5     Teste.Verify(m,sign)

7 Out: Assinatura aceite!
```

B

IMPLEMENTAÇÃO DO DILITHIUM

```
import hashlib

class Dilithium:

    def __init__(self):
        self.n = 256
        self.q = 8380417
        self.h = 60
        self.r = 1753
        self.k = 4
        self.l = 3
        self.gama = 523776
        self.alfa = 261888
        self.eta = 6
        self.beta = 325


    #FUNCOES AUXILIARES:


    # Geracao do conjunto S
    def gen_S(self,lim,tam):
        Zq.<z> = PolynomialRing(GF(self.q))
        Rq.<z> = Zq.quotient(z^self.n+1)
        S = []
        for i in range(tam):
            pol = []
            for j in range(self.n):
                pol.append(randint(1,lim))
            S.append(Rq(pol))
        S = matrix(Rq,tam,1,S)
        return S

    def Decompose(self,c,t):
        Rq = IntegerModRing(self.q)
        Rt = IntegerModRing(t)
        r = int(Rq(c))
        r0 = int(Rt(r))
```

```
39          if r0 > t/2:
                r0 = r0 - int(t)
41          if r - r0 == self.q - 1:
                r1 = 0
43              r0 = r0 - 1
            else:
45              r1 = (r - r0)/(int(t))
            return (r1,r0)
47
        def HighBits(self,c):
49          x = self.Decompose(c,2*self.alfa)
            return x[0]
51
        def LowBits(self,c):
53          x = self.Decompose(c,2*self.alfa)
            return x[1]
55
        def HBpol(self,pol):
57          k = pol.list()
            for i in range(len(k)):
59              h = k[i]
                h = h.list()
61              for j in range(len(h)):
                    h[j] = self.HighBits(int(h[j]))
63              k[i] = h
            return k
65
        def LBpol(self,pol):
67          k = pol.list()
            for i in range(len(k)):
69              h = k[i]
                h = h.list()
71              for j in range(len(h)):
                    h[j] = self.LowBits(int(h[j]))
73              k[i] = h
            return k
75
        # Geracao de um elemento aleatorio em B60
77      def SampleInBall(self,r):
            """ 'r' e usado como fonte de aleatoriedade e deve ser
        suficientemente grande para possibilitar o rejection-sampling dos 60
        valores de 'j' """
79          sl = "{:064b}".format(int(r[:8].hex(),16)) # primeiros 60 bit sao
         usados para selecao sinal.
            k = 8 # posicao para 'rejection-sampling'
81          c = [0] * 256 # coeficientes
            for i in range (196,256):
83              while (int(r[k])>i):
                    k +=1 # rejeita valores fora da gama pretendida.
85              j = int(r[k])
                k += 1
```

```
87              s = int(sl[i-196])
                c[i] = c[j]
89              c[j] = (-1)^(s)
            return c
91
        def Shake(self,a,b, n=int(256)):
93          """ default-value para 'n' da uma folga para 60 rejeicoes"""
            shake = hashlib.shake_256()
95          shake.update(a)
            shake.update(b)
97          s = shake.digest(n)
            return s
99
        def Hash(self,a,b,n=int(256)):
101         r = self.Shake(a,b,n)
            c = self.SampleInBall(r)
103         return c

105     def norma_infinito(self,pol,n):
            J = pol.list()
107         for i in range(len(J)):
                k = J[i]
109             K = k.list()
                for j in range(len(K)):
111                 K[j] = abs(int(K[j]))
                J[i] = K
113         L = []
            for i in range(len(J)):
115             L.append(max(J[i]))
            return max(L)
117
        def norma_inf_vet(self,vetor):
119         Zq.<z> = PolynomialRing(GF(self.q))
            Rq.<z> = Zq.quotient(z^self.n + 1)
121         for i in range(vetor.nrows()):
                norm = self.norma_infinito(vetor[i],self.q)
123             vetor[i] = norm
            return max(vetor)
125
        def norma_inf_matriz(self,matriz):
127         L = []
            for i in range(len(matriz)):
129             k = matriz[i]
                for j in range(len(k)):
131                 if k[j] < 0:
                        k[j] = abs(k[j])
133             L.append(max(k))
            for i in range(len(L)):
135             J = []
                J.append(max(L))
137         return J[0]
```

```
139     # FUNCOES PRINCIPAIS

141
        def Keygen(self):
143         Zx.<x> = ZZ[]
            R.<x> = Zx.quotient(x^self.n+1)
145         Zq.<z> = PolynomialRing(GF(self.q))
            Rq.<z> = Zq.quotient(z^self.n+1)
147
            # PASSO 1
149         K = []
            for i in range(self.k*self.l):
151             K.append(Rq.random_element())
            A = matrix(Rq,self.k,self.l,K)
153
            # PASSO 2
155         s1 = self.gen_S(self.eta,self.l) #s1: matriz 3x1
            s2 = self.gen_S(self.eta,self.k) #s2:matriz 4x1
157
            # PASSO 3
159         t = A*s1 + s2 #t: matriz 4x1

161         # PASSO 4 E 5
            self.pubKey = (A,t)
163         self.privKey = (s1,s2)


165
        def Sign(self,m):
167
            A = self.pubKey[0]
169         s1 = self.privKey[0]
            s2 = self.privKey[1]
171
            Zx.<x> = ZZ[]
173         R.<x> = Zx.quotient(x^self.n + 1)
            Zq.<z> = PolynomialRing(GF(self.q))
175         Rq.<z> = Zq.quotient(z^self.n + 1)

177         # PASSO 1
            z = None
179
            # PASSO 2
181         while z == None:

183             # PASSO 3
                y = self.gen_S(self.gama - 1,self.l)
185
                # PASSO 4
187             Ay = A*y
                w = self.HBpol(Ay)
```

```
189
            # PASSO 5
191         u = str(w).encode()
            k = m.encode()
193         c = self.Hash(k,u)
            cq = Rq(c)
195
            # PASSO 6
197         z = matrix(y + cq*s1)

199         aa = self.norma_inf_vet(z)[0]
            bb = int(self.gama - self.beta)
201         cc = int(self.norma_inf_matriz(self.LBpol(Ay-cq*s2)))
            dd = int(self.alfa - self.beta)
203
            # PASSO 7 E 8
205         if aa >= bb or cc >= dd:
                z = None
207
        z = matrix(y + cq*s1)
209     return (z,c)


211
    def Verify(self,m,sig):
213     Zx.<x> = ZZ[]
        R.<x> = Zx.quotient(x^self.n+1)
215     Zq.<z> = PolynomialRing(GF(self.q))
        Rq.<z> = Zq.quotient(z^self.n+1)
217
        A = self.pubKey[0]
219     t = self.pubKey[1]
        z = sig[0]
221     c = sig[1]
        cq = Rq(c)
223
        # PASSO 1
225     Az = A*z
        w = self.HBpol(Az - cq*t)
227
        # PASSO 2 E 3
229     u = str(w).encode()
        k = m.encode()
231     novo_c = self.Hash(k,u)
        aa = int(self.norma_inf_vet(z)[0])
233     bb = int(self.gama - self.beta)
        if aa >= bb or c != novo_c:
235         print('Assinatura rejeitada!')
        else:
237         print('Assinatura aceite!')
```

```
In: Teste = Dilithium()
    Teste.Keygen()
    m = bin(123456789)
    sign = Teste.Sign(m)
    Teste.Verify(m,sign)

Out: Assinatura aceite!
```

C

CERTIFICADO HÍBRIDO *rsa3072_dilithium2*

```
(base) ZitaAbreu@Air-de-Maria New % apps/openssl x509 -noout -text -in
rsa3072_dilithium2_CA.crt

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            1e:86:e3:81:23:ff:72:c0:e1:cf:bd:59:d4:21:89:1e:d4:7c:f7:a0
        Signature Algorithm: rsa3072_dilithium2
        Issuer: CN = oqstest CA
        Validity
            Not Before: Aug 13 15:50:56 2020 GMT
            Not After : Aug 13 15:50:56 2021 GMT
        Subject: CN = oqstest CA
        Subject Public Key Info:
            Public Key Algorithm: rsa3072_dilithium2
                RSA Public-Key: (3072 bit)
                Modulus:
                    00:a8:e1:6b:47:66:67:98:d0:65:07:45:3e:f5:f2:
                    ...

                Exponent: 65537 (0x10001)
                dilithium2 Public-Key:
                pub:
                    04:2f:ce:92:44:f6:ef:ff:52:f7:d8:7c:30:a4:cc:
                    ...

        X509v3 extensions:
            X509v3 Subject Key Identifier:
                3D:88:03:C5:BC:9F:F5:3B:2B:51:8A:BB:71:DB:24:15:13:11:14:AA
            X509v3 Authority Key Identifier:
                keyid:3D:88:03:C5:BC:9F:F5:3B:2B:51:8A:BB:71:DB:24:15:13:11:14:AA
```

```
        X509v3 Basic Constraints: critical
            CA:TRUE
Signature Algorithm: rsa3072_dilithium2
     00:00:01:80:2c:b1:87:a8:f4:97:70:d7:d

     ...
```

D

CERTIFICADO HÍBRIDO *p256_dilithium2*

```
(base) ZitaAbreu@Air-de-Maria New % apps/openssl x509 -noout -text -in
p256_dilithium2_CA.crt

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            1c:19:e0:17:6e:79:1f:b9:72:f2:29:ed:3c:2c:4f:2f:10:5f:ed:7b
        Signature Algorithm: p256_dilithium2
        Issuer: CN = oqstest CA
        Validity
            Not Before: Aug 13 15:08:12 2020 GMT
            Not After : Aug 13 15:08:12 2021 GMT
        Subject: CN = oqstest CA
        Subject Public Key Info:
            Public Key Algorithm: p256_dilithium2
                Public-Key: (256 bit)
                pub:
                    04:ad:a4:37:b5:46:7a:29:70:a8:68:af:cb:40:ce:
                    27:92:cc:8f:ba:2f:a6:38:5e:4e:d1:80:43:57:be:
                    51:90:a6:fc:bf:0d:87:79:80:94:ce:5f:a1:4e:b1:
                    0c:74:39:31:63:de:f7:c3:be:42:63:97:d4:1c:6d:
                    1c:72:f7:d6:b5
                ASN1 OID: prime256v1
                NIST CURVE: P-256
                dilithium2 Public-Key:
                pub:
                    b3:bb:a4:cb:a6:b9:a2:e9:e3:b2:c4:90:67:ee:fe:

                    ...

        X509v3 extensions:
```

```
        X509v3 Subject Key Identifier:
            22:52:DF:C8:05:23:82:E4:9F:4E:D3:0F:21:83:2F:12:AC:FD:DC:92
        X509v3 Authority Key Identifier:
            keyid:22:52:DF:C8:05:23:82:E4:9F:4E:D3:0F:21:83:2F:12:AC:FD:DC:92

        X509v3 Basic Constraints: critical
            CA:TRUE
Signature Algorithm: p256_dilithium2
     00:00:00:46:30:44:02:20:5c:da:c9:5d:2f:30:79:62:83:d0:

     ...
```

E

CERTIFICADO HÍBRIDO *p384_dilithium4*

```
(base) ZitaAbreu@Air-de-Maria New % apps/openssl x509 -noout -text -in
p384_dilithium4_CA.crt

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0a:07:ac:ae:11:34:f0:90:e7:3a:63:c3:a6:e1:30:c3:6f:78:62:d0
        Signature Algorithm: p384_dilithium4
        Issuer: CN = oqstest CA
        Validity
            Not Before: Aug 13 16:10:59 2020 GMT
            Not After : Aug 13 16:10:59 2021 GMT
        Subject: CN = oqstest CA
        Subject Public Key Info:
            Public Key Algorithm: p384_dilithium4
                Public-Key: (384 bit)
                pub:
                    04:3a:81:c4:c0:2f:cc:d5:8a:f9:ea:2b:56:d9:bf:
                    d0:74:9f:a7:ac:14:0e:f5:5c:d7:75:de:58:12:4b:
                    d2:53:5d:7c:91:5c:6f:47:38:0b:f4:35:24:1f:cd:
                    2b:e4:1f:bb:e6:58:2e:ec:77:46:8d:dc:28:a6:f3:
                    8a:28:3c:b0:c9:9e:18:0e:63:b2:4f:f4:94:9c:78:
                    67:f9:2d:b5:0d:37:0f:dc:df:fa:a9:5a:cb:32:57:
                    e3:8e:92:0e:4e:f7:f4
                ASN1 OID: secp384r1
                NIST CURVE: P-384
                dilithium4 Public-Key:
                pub:
                    c9:72:f0:97:57:79:37:d3:6d:83:47:36:50:c6:54:

                    ...
```

```
    X509v3 extensions:
        X509v3 Subject Key Identifier:
            9E:9B:0B:0F:14:5F:4D:6C:51:6C:D2:71:7F:29:03:1A:ED:56:C3:EB
        X509v3 Authority Key Identifier:
            keyid:9E:9B:0B:0F:14:5F:4D:6C:51:6C:D2:71:7F:29:03:1A:ED:56:C3:EB

        X509v3 Basic Constraints: critical
            CA:TRUE
Signature Algorithm: p384_dilithium4
     00:00:00:67:30:65:02:31:00:ad:1d:90:21:32:f0:be:a8:5b:

     ...
```

F

CERTIFICADO HÍBRIDO *p256_qteslapi*

```
(base) ZitaAbreu@Air-de-Maria NEW % apps/openssl x509 -noout -text -in
p256_qteslapi_CA.crt

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            3d:64:49:f0:d0:6e:13:af:ac:a0:c3:7a:64:11:1c:48:a3:ea:d1:8c
        Signature Algorithm: p256_qteslapi
        Issuer: CN = oqstest CA
        Validity
            Not Before: Aug 28 12:50:30 2020 GMT
            Not After : Aug 28 12:50:30 2021 GMT
        Subject: CN = oqstest CA
        Subject Public Key Info:
            Public Key Algorithm: p256_qteslapi
                Public-Key: (256 bit)
                pub:
                    04:b0:c2:86:82:bb:9f:b8:63:78:19:e2:f2:07:be:
                    1e:74:49:91:1d:56:a9:6a:48:cf:fc:bc:2a:56:a3:
                    11:c9:e0:f4:8f:1a:95:82:ae:70:86:3a:db:5f:2e:
                    87:e2:0a:a8:ff:73:d0:75:9d:59:71:10:15:8b:d1:
                    38:11:a5:7a:32
                ASN1 OID: prime256v1
                NIST CURVE: P-256
                qteslapi Public-Key:
                pub:
                    ff:8c:5e:c6:ae:17:4f:ac:46:28:bd:c4:77:24:b3:

                    ...

        X509v3 extensions:
```

```
        X509v3 Subject Key Identifier:
            27:5E:4A:ED:C4:FB:CC:B2:E0:7C:D1:FB:15:D3:99:09:02:54:9D:06
        X509v3 Authority Key Identifier:
            keyid:27:5E:4A:ED:C4:FB:CC:B2:E0:7C:D1:FB:15:D3:99:09:02:54:9D:06

        X509v3 Basic Constraints: critical
            CA:TRUE
Signature Algorithm: p256_qteslapi
     00:00:00:48:30:46:02:21:00:aa:84:44:00:93:b7:3d:a1:c8:

     ...
```

G

CERTIFICADO HÍBRIDO *p384_qteslapiii*

```
(base) ZitaAbreu@Air-de-Maria NEW % apps/openssl x509 -noout -text -in
p384_qteslapiii_CA.crt

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            79:6d:0f:7d:f6:47:a4:0c:67:6d:4d:85:49:e9:1a:f6:50:c4:32:d8
        Signature Algorithm: p384_qteslapiii
        Issuer: CN = oqstest CA
        Validity
            Not Before: Aug 28 13:19:40 2020 GMT
            Not After : Aug 28 13:19:40 2021 GMT
        Subject: CN = oqstest CA
        Subject Public Key Info:
            Public Key Algorithm: p384_qteslapiii
                Public-Key: (384 bit)
                pub:
                    04:1e:9d:90:0c:36:4c:6b:f1:a7:ff:32:97:2c:4c:
                    5a:3e:b7:c8:59:83:58:09:be:4a:f5:93:83:58:fe:
                    9c:ba:99:7a:e7:24:b6:31:ff:67:7d:7b:07:e1:51:
                    d9:00:34:ba:8a:57:67:d5:6b:7d:a0:b6:65:98:fb:
                    35:4c:ab:9e:7f:d9:92:12:1b:85:94:a5:51:e4:a1:
                    41:9e:1b:9c:f8:d7:30:dc:1c:bb:5d:73:fa:1f:72:
                    4f:d6:fa:f7:97:e3:0f
                ASN1 OID: secp384r1
                NIST CURVE: P-384
                qteslapiii Public-Key:
                pub:
                    f9:21:1d:ec:7a:c7:59:0c:ed:3e:d5:c8:70:00:22:

                    ...
```

```
    X509v3 extensions:
        X509v3 Subject Key Identifier:
            B6:27:F9:8C:8A:22:1F:DF:04:4E:B5:0B:3B:6D:CF:FD:7D:BF:F9:37
        X509v3 Authority Key Identifier:
            keyid:B6:27:F9:8C:8A:22:1F:DF:04:4E:B5:0B:3B:6D:CF:FD:7D:BF:F9:37

        X509v3 Basic Constraints: critical
            CA:TRUE
Signature Algorithm: p384_qteslapiii
     00:00:00:66:30:64:02:30:6f:ae:a3:8f:48:5c:b6:53:3a:60:

     ...
```