UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

AFONSO COMBA DE ARAÚJO NETO

Um Algoritmo de Criptografia de Chave Pública Semanticamente Seguro Baseado em Curvas Elípticas

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Raul Fernando Weber
Orientador

Porto Alegre, abril de 2006

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Araújo Neto, Afonso Comba de

Um Algoritmo de Criptografia de Chave Pública Semanticamente Seguro Baseado em Curvas Elípticas / Afonso Comba de Araújo Neto. – Porto Alegre: PPGC da UFRGS, 2006.

95 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2006. Orientador: Raul Fernando Weber.

1. Criptografia de chave pública. 2. Curvas elípticas. 3. Multiplicação complexa. 4. Segurança semântica. 5. Geradores de bits pseudo-aleatórios criptograficamente seguros. I. Weber, Raul Fernando. II. Título.

“It is possible to write endlessly on elliptic curves. (This is not a threat.)”
— SERGE LANG

AGRADECIMENTOS

Qualquer um que já escreveu uma dissertação de mestrado sabe a dificuldade que isto representa. Ao final de um caminho tão tortuoso, resta agradecer aos que nos ajudaram a trilhá-lo. Aos meus pais, obrigado por tudo, sem exceção. À minha namorada, obrigado pela compreensão e por não me deixar desistir jamais. Aos meus amigos, obrigado pelo incentivo e pela companhia nos momentos difíceis.

Em especial, gostaria de agradecer ao meu orientador Raul Fernando Weber, que nunca duvidou do meu potencial, e a Donald E. Knuth sem o qual, talvez, este documento nunca ficasse pronto.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AES	Advanced Encryption Standard
CM	Complex Multiplication
CSPRBG	Cryptographically Secure PseudoRandom Bit Generator
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
ECAES	Elliptic Curve Advanced Encryption Standard
ECDSA	Elliptic Curve Digital Signature Algorithm
ECDH	Elliptic Curve Diffie-Hellman
ECDHP	Elliptic Curve Diffie-Hellman Problem
ECDLP	Elliptic Curve Discrete Logarithm Problem
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
MD5	Message Digest 5
NIST	National Institute of Standards and Technology
PRNG	PseudoRandom Number Generator
RSA	Rivest Shamir Adleman
SHA-1	Secure Hash Algorithm 1

LISTA DE TABELAS

LISTA DE ALGORITMOS

RESUMO

Esta dissertação apresenta o desenvolvimento de um novo algoritmo de criptografia de chave pública. Este algoritmo apresenta duas características que o tornam único, e que foram tomadas como guia para a sua concepção. A primeira característica é que ele é semanticamente seguro. Isto significa que nenhum adversário limitado polinomialmente consegue obter qualquer informação parcial sobre o conteúdo que foi cifrado, nem mesmo decidir se duas cifrações distintas correspondem ou não a um mesmo conteúdo. A segunda característica é que ele depende, para qualquer tamanho de texto claro, de uma única premissa de segurança: que o logaritmo no grupo formado pelos pontos de uma curva elíptica de ordem prima seja computacionalmente intratável. Isto é obtido garantindo-se que todas as diferentes partes do algoritmo sejam redutíveis a este problema. É apresentada também uma forma simples de estendê-lo a fim de que ele apresente segurança contra atacantes ativos, em especial, contra ataques de texto cifrado adaptativos. Para tanto, e a fim de manter a premissa de que a segurança do algoritmo seja unicamente dependente do logaritmo elíptico, é apresentada uma nova função de resumo criptográfico (*hash*) cuja segurança é baseada no mesmo problema.

Palavras-chave: Criptografia de chave pública, Curvas elípticas, Multiplicação complexa, Segurança semântica, Geradores de bits pseudo-aleatórios criptograficamente seguros.

A semantically secure public key algorithm based on elliptic curves

ABSTRACT

This dissertation presents the development of a new public key algorithm. This algorithm has two key features, which were taken to be a goal from the start. The first feature is that it is semantically secure. That means that no polynomially bounded adversary can extract any partial information about the plaintext from the ciphertext, not even decide if two different ciphertexts correspond to the same plaintext. The second feature of the algorithm is that it depends on only one security assumption: that it is computationally unfeasible to calculate the logarithm on the group formed by the points of a prime order elliptic curve. That is achieved by ensuring that all parts of the algorithm are reducible to that problem. Also, it is presented a way to extend the algorithm so that it the resists attacks of an active adversary, in special, against an adaptive chosen-ciphertext attack. In order to do that, and attain to the assumption that only the assumption of the logarithm is necessary, it is introduced a new hash function with strength based of the same problem.

Keywords: Elliptic curves, Public-key cryptography, Semantic security, Complex Multiplication, Cryptographically Strong Pseudorandom Bit Generators.

1 INTRODUÇÃO

Nos dias de hoje, praticamente todos dependem de computadores. Mesmo os que se vangloriam de nunca terem sentado em frente a um monitor, acabam por depender deles de alguma forma. E essa dependência implica que é importante garantir a integridade deles. A área computacional chamada tolerância a falhas preocupa-se com a integridade dos sistemas contra falhas naturais, que por definição, são não intencionais. Já a área de *segurança de sistemas* se preocupa em lidar com modificações maliciosas, propositadamente arquitetadas por alguém que pode ou não conhecer todos os detalhes do sistema. Porém, o alto nível de conectividade fornecido pela *Internet* faz com que os sistemas computacionais, e as relações entre diferentes sistemas, apresentem uma complexidade nunca antes vista. Prover segurança neste ambiente caótico é um desafio bastante grande.

Como bloco fundamental na construção de sistemas seguros aparece a ciência (ou como alguns a definem, arte) da *criptografia*. E dentre as técnicas criptográficas mais importantes se encontram os algoritmos de cifragem, que fornecem uma forma confiável de confidencialidade de dados baseadas em técnicas matemáticas reconhecidas. Os algoritmos de cifragem podem ser divididos em dois grandes grupos: os algoritmos de chave *simétrica* e os algoritmos de chave *assimétrica*.

É conveniente, desde já, estabelecer-se uma terminologia básica. No que diz respeito a esta dissertação, qualquer dado a ser criptografado (ou cifrado) é chamado de *texto claro*, mesmo que o dado em si não represente um texto propriamente dito. Já para o resultado da cifragem, será dado o nome de *texto cifrado*.

Os algoritmos de chave simétrica estão entre os mais rápidos algoritmos de cifragem conhecidos atualmente. Entretanto, eles possuem a característica de que a mesma chave que é utilizada para cifragem é a utilizada para decifragem, e isso faz com que a chave tenha que ser transmitida ao destinatário de alguma forma, o que nem sempre é uma tarefa trivial. Visando resolver este problema, os pesquisadores Whitfield Diffie e Martin Hellman apresentam em (DIFFIE; HELLMAN, 1976), pela primeira vez, o conceito de criptografia assimétrica, onde a chave de cifragem é diferente da chave de decifragem e a chave de decifragem (usualmente secreta) não pode ser deduzida a partir da chave de cifragem (usualmente pública). Uma desvantagem dos algoritmos de cifragem assimétrica é que, via de regra, eles são muito mais lentos que os de chave simétrica. Entretanto, a combinação de ambos paradigmas provê resultados excelentes na prática.

Todos sistemas de criptografia assimétrica são baseados em algum problema matemático de difícil solução, como a fatoração de inteiros em seus componentes primos. Na metade da década de 80, Victor Miller (MILLER, 1986) e Neal Koblitz (KOBLITZ, 1987), independentemente, sugerem o uso da matemática baseada em curvas elípticas para este propósito. A teoria de curvas elípticas fornece um problema que é considerado muito mais difícil que os problemas difíceis mais tradicionais. Sistemas criptográficos assi-

métricos que se utilizam de curvas elípticas tendem a ser mais rápidos e possuir chaves muito menores que os sistemas baseados em outros problemas, e, portanto tornaram as curvas elípticas um dos assuntos mais estudados nos dias de hoje. Outro fator que recentemente trouxe muita atenção ao uso de curvas elípticas na criptografia é a chamada *criptografia baseada em identidades* originalmente proposta por Shamir em (SHAMIR, 1985) que finalmente recebe uma implementação prática por Boneh e Franklin em (BONEH; FRANKLIN, 2001) sob a forma de emparelhamentos bilineares sobre curvas elípticas.

O objetivo desta dissertação é a proposição de um algoritmo de cifragem assimétrica que possua especificamente duas características. A primeira característica, é que ele seja baseado unicamente no problema provido pelas curvas elípticas. Obviamente, quanto menos premissas de segurança um algoritmo possuir, menor é a chance dele ser quebrado, e esta é a motivação desta primeira característica. A segunda característica é que ele possua *segurança semântica*. Um algoritmo que possua esta característica garante que ninguém consegue obter nenhuma informação acerca do texto claro a partir do texto cifrado, nem mesmo a informação de se duas cifragens correspondem ao mesmo texto claro.

Após a apresentação do algoritmo semanticamente seguro, e da sua prova de segurança, será apresentada uma extensão do mesmo que o torna *não maleável*. Isso significa que esta versão modificada será imune contra atacantes dito ativos, que tentam valer-se do pré conhecimento de alguns textos claros para obter informações sobre outros. De forma a implementar esta extensão será apresentada uma nova função de resumo criptográfico baseada na função *Chaum-van Heijst-Pfitzmann* (CHAUM; HEIJST; PFITZMANN, 1992), mas cuja segurança é redutível ao problema difícil que as curvas elípticas fornecem.

Resumidamente, as principais contribuições desta dissertação são:

- O algoritmo semanticamente seguro e a prova da sua segurança,
- A função de resumo criptográfico baseada em curvas elípticas,
- O algoritmo modificado não maleável.

1.1 Trabalhos relacionados

É possível encontrar pelo menos um outro algoritmo que apresenta segurança semântica e é baseado em curvas elípticas chamado PSEC (OKAMOTO; FUJISAKI; MORITA, 1999), que foi uma submissão para o padrão IEEE P1363. Entretanto, uma das desvantagens deste algoritmo é que ele possui mais de uma premissa de segurança. Além da premissa sobre curvas elípticas, ele utiliza uma função de *hash* genérica, e, para mensagens longas, o algoritmo vale-se de um algoritmo de chave simétrica não especificado.

Apesar de existirem escolhas de funções de *hash* e algoritmos de chave simétrica para os quais certamente o algoritmo é seguro, é interessante a existência de um algoritmo semanticamente seguro que seja dependente única e exclusivamente das curvas elípticas, para qualquer tamanho de texto claro. E esta é a principal diferença entre o PSEC e o algoritmo proposto nesta dissertação.

1.2 Estrutura da dissertação

A dissertação está estruturada da seguinte forma. O capítulo 2 apresenta uma introdução bastante profunda à teoria de geradores de bits pseudo-aleatórios criptograficamente seguros. O algoritmo proposto é extremamente dependente desta teoria, e portanto uma

apresentação bastante compreensiva é feita. Na última seção deste capítulo são apresentados formalmente o conceito de segurança semântica e as primeiras propostas de algoritmos que apresentam esta característica.

O capítulo 3 apresenta a teoria acerca de curvas elípticas. Inicialmente alguns fundamentos matemáticos são fornecidos, e, após, são apresentadas em um nível bastante acessível a teoria necessária para se compreender a utilização de curvas elípticas na criptografia. Alguns exemplos de protocolos famosos são apresentados também.

O capítulo 4 apresenta um gerador de bits pseudo-aleatórios desenvolvido por Burton Kaliski, que é baseado em curvas elípticas. Este gerador é uma base fundamental do algoritmo sendo desenvolvido, e, portanto, ele é apresentado em detalhes.

O capítulo 5 apresenta o algoritmo semanticamente seguro sendo proposto nesta dissertação. A apresentação é dividida em três grandes áreas, respectivamente criação do par de chaves, cifragem e decifragem. A seção sobre o par de chaves, em especial, apresenta um método eficiente de se encontrar uma curva elíptica e o seu *twist* tal que ambas possuam um número primo de pontos, fato que é utilizado extensivamente no algoritmo. O método utiliza a teoria de *multiplicação complexa* de curvas elípticas, a qual é somente superficialmente explicada, dando-se ênfase à sua aplicação prática.

O capítulo 6 apresenta os cenários de segurança abordados e a segurança do algoritmo relativamente a estes cenários. As provas também são fornecidas. A última seção deste capítulo apresenta uma nova função de resumo criptográfico baseada em curvas elípticas e a utiliza para estender o algoritmo proposto no capítulo 5 de forma que ele seja seguro em um último cenário de segurança.

O último capítulo brevemente encerra a dissertação com uma recapitulação geral e algumas discussões sobre o trabalho.

2 GERADORES DE BITS PSEUDO-ALEATÓRIOS CRIPTOGRAFICAMENTE SEGUROS

Números aleatórios têm um papel fundamental na criptografia. Todo o sistema criptográfico possui elementos conhecidos pelos participantes do sistema, mas que devem ser mantidos secretos para todas as outras pessoas. Em alguns casos não só secretos, mas devem possuir características imprevisíveis. Diversos exemplos destes elementos existem, como por exemplo chaves em sistemas de criptografia simétrica, desafios em protocolos desafio-resposta, números primos para geração de chaves RSA e valores aleatórios para algoritmos de assinaturas, além de diversos outros. Ao contrário do que se pode imaginar intuitivamente, gerar seqüências de números aleatórios em um computador é uma tarefa muito complicada. Isto acontece porque os sistemas computacionais foram desenvolvidos para funcionar deterministicamente. Quando funcionam aleatoriamente, é porque não estão funcionando como deveriam.

Mesmo existindo *hardware* especial desenvolvido para a geração de números aleatórios, não se pode esperar que todos os computadores que forem utilizar algum sistema criptográfico os possuam. Logo, alguma alternativa a estes deve ser encontrada.

Nos computadores pessoais atuais, existem algumas formas de se conseguir pequenas amostras de valores aleatórios. Medidas como data, hora, temperaturas, velocidade de rotação dos discos, latência de tráfego na rede, digitação e movimentação do mouse pelo usuário, são fontes que combinadas podem gerar valores satisfatoriamente aleatórios. O problema com estas fontes é que elas geram uma quantidade muito restrita de aleatoriedade. E freqüentemente é uma quantidade muito tendenciosa, e deve-se ter cuidado ao gerar números a partir destas medidas. Ainda assim, geralmente é necessário uma quantidade de valores muito maior do que a que é possível obter desta forma.

Uma solução prática para este problema é a utilização de números *pseudo-aleatórios* produzidos por algoritmos. Os algoritmos geradores de números pseudo-aleatórios (ou *pseudorandom number generators*, PRNGs) são essencialmente determinísticos, como quaisquer outros. Eles recebem como entrada uma pequena quantidade aleatória, chamada *semente*, e geram como saída uma grande seqüência de valores indistinguível de uma seqüência realmente aleatória. A principal diferença desta seqüência para uma seqüência realmente aleatória é que ela é reprodutível. Ou seja, dado um PRNG e uma mesma semente, a saída é absolutamente idêntica, como seria esperado de qualquer algoritmo.

PRNGs podem ser utilizados em quaisquer aplicações que requeiram dados de entrada com distribuição aleatória. Aplicações científicas como simulações Monte Carlo são exemplos de aplicações que apresentam este requisito. Uma lista extensa, mas não exaustiva, de aplicações onde são necessárias grandes quantidades de valores aleatórios

pode ser encontrada em (KNUTH, 1973). Uma grande vantagem da utilização de PRNGs para estas aplicações é a sua reprodutibilidade. Assim, em uma dada simulação, os efeitos de determinadas seqüências de entrada podem ser verificados novamente sempre que desejado. Para estas aplicações, o principal pré-requisito que o PRNG deve possuir é uma boa distribuição de saída.

Quando se trata de aplicações criptográficas, existe pelo menos uma outra característica que deve ser observada, que é a imprevisibilidade. Idealmente, dada uma saída de *n* bits, não deve ser possível a previsão do bit $n + 1$ com probabilidade maior que $\frac{1}{2} + \epsilon$, com ϵ sendo tão pequeno quanto se queira, por qualquer método que não seja a descoberta da semente. Em outras palavras, não deve existir um algoritmo polinomial capaz de predizer o próximo bit da seqüência. Note que um ataque de força bruta na semente do gerador sempre será bem sucedido. Entretanto, cuidando-se para que o número de sementes possíveis seja grande o suficiente, este ataque torna-se impraticável.

Da necessidade de imprevisibilidade, surge, no início da década de 80, o conceito de **geradores de bits pseudo-aleatórios criptograficamente seguros** (em inglês *cryptographically strong pseudorandom bit generators*, ou simplesmente CSPRBG, apresentados no capítulo). Baseados no trabalho de Andrew Yao (YAO, 1982) que tinha por base a teoria da informação, os pesquisadores Silvio Micali e Manuel Blum definiram uma série de condições práticas para a construção de um CSPRBG (BLUM; MICALI, 1984).

Uma das principais características deste tipo de gerador é que ele possui um problema difícil incorporado na sua construção, normalmente, uma função unidirecional. Funções unidirecionais são funções que são fáceis de calcular mas difíceis de se inverter. Atualmente, do ponto de vista computacional, os termos fácil e difícil são definidos com base na teoria da complexidade. No capítulo 2.1 será fornecida uma definição um pouco mais formal destas idéias.

No seu trabalho seminal, Yao mostrou como é possível utilizar qualquer função unidirecional perfeita para a construção de um CSPRBG. Funções unidirecionais perfeitas, da forma como Yao as definiu, de fato não existem e, até hoje, não é possível se afirmar se existe alguma nem sob a perspectiva de complexidade computacional. De fato, o problema é tão complicado, que implica o seguinte resultado: caso existam funções unidirecionais, então $P \neq NP$[1] (que é um dos grandes problemas sem solução nos dias de hoje e cuja resposta receberá de recompensa um milhão de dólares paga pelo *Clay Mathematics Institute* nos Estados Unidos). O contrário, entretanto, não é verdadeiro. A não existência de funções unidirecionais não implica necessariamente $P = NP$, pois uma das características dessas funções é que elas sejam fáceis de se calcular, fato que muito provavelmente não é partilhado por todas as funções pertencentes a NP.

Uma outra característica importante dos CSPRBGs é que eles possuem *prova de segurança*. Isto é, é possível mostrar que, presumindo-se que a sua função unidirecional seja segura, então o gerador também o será. Essa é uma característica muito importante nos dias de hoje, e que não é obtida por praticamente nenhum algoritmo de criptografia simétrica, e tampouco pelos geradores de seqüências pseudo-aleatórias baseados em registradores de deslocamento. Isso não significa, porém, que estes outros algoritmos, desenvolvidos de forma *ad hoc*, por assim dizer, são inseguros. Entretanto, não existe nada que garanta a sua segurança além do fato de que não foram quebrados. De qualquer forma, é importante lembrar que a principal vantagem destes algoritmos criptográficos *ad hoc* é que eles tendem a ser muito mais eficientes que os que possuem prova de segurança.

[1] Ou seja, o conjunto de problemas com solução em tempo polinomial é diferente (na verdade está estritamente contido) do conjunto de problemas com solução não determinística em tempo polinomial

Isso torna o seu uso prático muito mais atraente para usuários finais mesmo que o risco de serem quebrados em um dado momento seja maior.

Normalmente, a prova de que os CSPRBGs são seguros é feita a partir do seguinte raciocínio. Por contradição, o autor mostra que um algoritmo que quebre o CSPRBG de forma fácil pode ser utilizado para resolver problema no qual ele é baseado com a mesma facilidade. Como presume-se que não existam formas fáceis de se resolver este problema, conclui-se que o CSPRBG é seguro. Alguns problemas são fortes candidatos a funções unidirecionais. E é baseado nestes problemas que foram desenvolvidos os principais resultados sobre CSPRBGs.

Este capítulo apresentará uma introdução à teoria de números e seqüências pseudo-aleatórias. O algoritmo desenvolvido nesta dissertação se baseia fundamentalmente em um gerador de bits pseudo-aleatórios criptograficamente seguro, e é importante que a teoria por trás destes algoritmos seja compreendida para que o resultado final desta dissertação seja entendido. A última seção do capítulo será dedicada à criptografia probabilística, que é fundamentada na teoria de CSPRBGs mas pode ser explicada em separado.

2.1 Aleatoriedade e Pseudo-aleatoriedade

A teoria acerca de geradores de bits pseudo-aleatórios criptograficamente seguros depende fundamentalmente da definição de aleatoriedade. O matemático Andrey Kolmogorov foi um dos precursores do estudo da aleatoriedade sob uma perspectiva computacional. Em (KOLMOGOROV, 1965), ele apresenta a seguinte definição de aleatoriedade:

Definição 2.1 *A seqüência de bits $A = a_1a_2a_3...a_n$ é aleatória se o menor programa capaz de gerar A como saída possui tamanho n.*

O tamanho de um programa (dado em bytes ou outra medida uniforme equivalente), entretanto, não é uma medida natural quando se trata do ponto de vista de complexidade computacional. Por exemplo, a existência de um programa de tamanho menor que n, que pudesse gerar a seqüência A, mas que levasse um tempo muito grande para fazê-lo, faria com que A não fosse aleatória pela definição 2.1. Porém, caso o tempo tendesse ao infinito (ou aumentasse muito rapidamente de acordo com algum parâmetro de fixo k), do ponto de vista prático o programa não pararia nunca. Assim não faria sentido considerá-lo um método de geração de A. Blum e Micali sugerem um experimento mental para transmitir a idéia que é necessária para adequar o conceito de aleatoriedade à perspectiva de complexidade computacional.

Alice e Bob fazem uma disputa de cara e coroa, com uma moeda honesta, em quatro situações distintas. Por honesta entende-se que a probabilidades da moeda cair com cada um dos lados para cima é exatamente 50%. Na primeira situação, Alice pede para Bob apostar em um dos lados, e então joga a moeda. Neste cenário, presume-se que Bob ganhará com 50% de freqüência. Na segunda situação, Bob aposta com a moeda já jogada, no ar. Em princípio, desconsiderando-se eventos externos, o resultado da jogada já está definido. Bob até poderia calcular o movimento da moeda e descobrir o resultado, mas ninguém presumiria que Bob seria capaz disso. Portanto a probabilidade dele ganhar continuará sendo 50%. A terceira situação é similar à segunda, entretanto é fornecido a Bob uma calculadora de bolso. Ainda sim, ninguém consideraria que a calculadora pudesse afetar a sua chance de ganhar. Entretanto, na quarta situação, é dado a Bob um supercomputador paralelo, munido de diversas câmeras e software de análise de imagens

e movimento em tempo real capaz de calcular a velocidade da moeda no ar, sua trajetória, seu momento angular, etc. Nessas condições, ninguém afirmaria que Bob ganharia sempre, mas é plausível admitir que ele pudesse ganhar pelo menos 51% das jogadas.

A idéia por trás deste experimento mental pode ser enunciada da seguinte forma:

A aleatoriedade de um evento é relativa a um modelo de computação específico e a uma quantidade específica de recursos computacionais disponíveis.

Como um corolário desta idéia, pode-se admitir que a aleatoriedade de um evento, sob uma perspectiva computacional, depende diretamente de um adversário que tenta identificar um padrão não aleatório no mesmo. E a idéia faz muito sentido porque, já que um computador é fundamentalmente determinístico, a aleatoriedade de um evento só é relevante se existir algum fator que dependa da sua previsibilidade ou imprevisibilidade. Portanto, é necessário sempre se definir contra qual adversário um determinado evento deve ser aleatório.

Primeiramente, para se definir o conceito de adversário, é importante entender o que o adversário está tentando fazer. Utilizando como exemplo a história acima, pode-se imaginar que o objetivo de Bob é adivinhar, com uma probabilidade maior que 50%, qual será o resultado da próxima jogada. Para tanto, ele se utiliza de todo o passado de jogadas e mais a movimentação da moeda no ar para tentar ganhar alguma vantagem sobre o próximo resultado. Formalmente, a seqüência de jogadas pode ser considerada uma fonte de resultados $(r_1, r_2, r_3...r_i)$. Bob se utiliza de algum recurso que recebe todas as entradas disponíveis e tenta predizer o resultado r_{i+1} com uma probabilidade $Pr[adivinhar(r_{i+1})] > \frac{1}{2}$.

Por definição, quando se presume que um evento é realmente aleatório, não existe forma de obter vantagem alguma na predição do resultado do evento. Isto acontece pois os eventos seriam considerados independentes. Nesta história em particular, Bob só terá alguma chance caso o resultado não seja absolutamente aleatório. Se ele só possuir informações de jogadas passadas, os eventos serão independentes, e, por definição, imprevisíveis. Entretanto, se a seqüência de valores é gerada por um computador, outros detalhes podem ser considerados.

Tratando-se de uma seqüência produzida por um sistema computacional, imediatamente imagina-se um algoritmo G. Este algoritmo recebe uma entrada binária n e produz uma seqüência de bits $b_1, b_2, b_3...b_i$ como saída. Neste caso, como a seqüência depende estritamente de n, pode-se concluir que a probabilidade de se adivinhar o bit b_{i+1} é, no mínimo, $Pr[adivinhar(b_{i+1})] \geq \frac{1}{2} + \epsilon$, com $\epsilon = \frac{1}{N}$, para um valor n uniformemente distribuído no intervalo $[1..N]$. Note que como n é constante, pode-se fazer ϵ ser tão pequeno quanto desejado. Isso significa que $\lim_{n\to\infty} \epsilon = 0$. Cada bit a mais no valor N faz com ϵ caia pela metade.

Expandindo formalmente estas idéias no seu artigo, Yao define o conceito de *fonte perfeita*. Para tanto, ele cria duas definições distintas para o mesmo conceito e, após, mostra que ambas são equivalentes. Do ponto de vista computacional, ele considera que uma fonte é um algoritmo que gera uma seqüência de bits. Ele também se utiliza do conceito de adversário polinomial, que ainda não foi apresentado. Entretanto, a definição intuitiva é suficiente por hora.

Definição 2.2 *[YAO primeira versão] O algoritmo G, junto com seu conjunto de entradas de tamanho n, é uma* ***fonte perfeita*** *se nenhum adversário polinomial é capaz de distinguir a seqüência S_G gerada por G de uma seqüência puramente aleatória S_A, para algum fator de segurança constante k suficientemente grande.*

Definição 2.3 *[YAO segunda versão] Seja P um polinômio. O algoritmo G, junto com seu conjunto de entradas de tamanho n, é uma **fonte perfeita**, se nenhum adversário polinomial é capaz de adivinhar o bit $i+1$ da seqüência $b_1, b_2, b_3...b_i$ gerada por G com probabilidade $Pr[adivinhar(b_{i+1})] \geq \frac{1}{2} + \frac{1}{P(n)}$, para algum fator de segurança constante k suficientemente grande. Este também é conhecido como* **teste do próximo bit** *(next-bit test).*

É importante ressaltar que nenhuma asserção foi feita sobre como G gera a seqüência, nem sobre as propriedades da entrada n. Em princípio, n poderia ser maior que a saída de G. Além disso, as definições valem inclusive quando G leva um tempo muito grande para produzir as seqüências, mesmo que uma fonte deste tipo não fosse muito útil na prática. Em suma, a equivalência entre as definições implica que, dados i bits de uma seqüência qualquer S_n, caso seja impraticável prever o próximo bit de S_n, para qualquer i, então a fonte que gera S_n pode ser considerada aleatória.

Uma questão que fica implícita nesta equivalência diz respeito às idéias de *imprevisibilidade para frente* e *imprevisibilidade para trás*. Considere a seqüência S_n do exemplo anterior, e seus i primeiros bits. Assuma que, dados os $i-1$ primeiros bits, é impossível descobrir o bit de número i com probabilidade maior que $\frac{1}{2} + \epsilon$. Imagine agora que são fornecidos os bits $\{b_2, b_3...b_i\}$. O que é possível dizer sobre o bit b_1?

De acordo com a definição 2.2, a resposta para esta questão deve ser *nada*. Pois, caso contrário, poderíamos utilizar o nosso conhecimento no padrão dos bits anteriores para distinguir a saída de um algoritmo de uma fonte aleatória, e, portanto, as definições não seriam equivalentes.

Isso leva ao seguinte fato: uma seqüência é imprevisível para frente (previsão dos próximos bits) se e somente se ela é imprevisível para trás (previsão dos bits anteriores). Saber que isto é verdade, torna muito mais natural a aceitação de que uma fonte imprevisível para frente é indistinguível de uma fonte aleatória. A prova disto é dada no seguinte teorema.

Teorema 2.4 *Seja G uma fonte perfeita nos termos da definição 2.3. Seja $\tilde{G}$ um algoritmo que roda G e fornece a sua saída em ordem inversa. Então, $\tilde{G}$ também é uma fonte perfeita pela definição 2.3.*

Prova A prova é por contradição. Seja $\tilde{T}$ um algoritmo que prevê o próximo bit de $\tilde{G}$ com probabilidade maior que $\frac{1}{2} + \frac{1}{P(n)}$ para algum polinômio P. Execute os seguintes experimentos, $experimento_{rand}(\tilde{G})$ e $experimento_{psrand}(\tilde{G})$. O $experimento_{psrand}(\tilde{G})$ é um algoritmo definido como o seguinte:

1. Selecione um valor aleatório x de tamanho k.
2. Execute $y = \tilde{G}(x)$.
3. Retorne o valor resultante de $\tilde{T}(y)$.

O $experimento_{rand}(\tilde{G})$ é definido como o seguinte. Seja l, o tamanho de y no experimento anterior:

1. Selecione um valor aleatório y de tamanho l.
2. Retorne o valor resultante de $\tilde{T}(y)$.

É necessário provar que $Pr[experimento_{psrand}(\tilde{G}) = 1] - Pr[experimento_{rand}(\tilde{G}) = 1] < \frac{1}{P(n)}$. Isto é, a probabilidade do primeiro experimento acertar o próximo bit de $\tilde{G}$ é quase tão pequena quanto acertar um próximo bit aleatório (que, por definição, não pode ser maior que $\frac{1}{2}$).

Seja T o seguinte algoritmo: T recebe como entrada z, e calcula $\tilde{z}$ que é z com todos os bits em ordem inversa. T então executa $\tilde{T}(\tilde{z})$ e retorna o valor obtido. Agora, considere os novos dois experimentos $experimento_{rand}(G)$ e $experimento_{psrand}(G)$. O $experimento_{psrand}(G)$ segue da seguinte forma:

1. Selecione um valor aleatório x de tamanho k.
2. Execute $y = G(x)$.
3. Retorne o valor resultante de $T(y)$.

O $experimento_{rand}(G)$ segue análogo ao primeiro caso:

1. Selecione um valor aleatório y de tamanho l.
2. Retorne o valor resultante de $T(y)$.

Por pressuposto, G é uma fonte perfeita. Logo,

$$Pr[experimento_{psrand}(G) = 1] - Pr[experimento_{rand}(G) = 1] < \frac{1}{P(n)}. \tag{2.1}$$

Note que pela definição 2.3, isso é verdade para qualquer T. Por construção

$$Pr[experimento_{psrand}(G) = 1] = Pr[experimento_{psrand}(\tilde{G}) = 1] \tag{2.2}$$

pois

$$T(G(x)) = \tilde{T}(\tilde{G}(x)). \tag{2.3}$$

Também, por construção,

$$Pr[experimento_{rand}(G) = 1] = Pr[experimento_{rand}(\tilde{G}) = 1] = \frac{1}{2} < \frac{1}{2} + \frac{1}{P(n)}, \tag{2.4}$$

pois em ambos experimentos, T e $\tilde{T}$ são executados sobre uma string puramente aleatória. Portanto, dado que $Pr[experimento_{psrand}(G) = 1] - Pr[experimento_{rand}(G) = 1] < \frac{1}{P(n)}$, tem-se que

$$Pr[experimento_{psrand}(\tilde{G}) = 1] - Pr[experimento_{rand}(\tilde{G}) = 1] < \frac{1}{P(n)}. \tag{2.5}$$

□

Com estas preliminares estabelecidas, pode-se voltar à definição de adversário. Sem perda de generalidade, o aparato que Bob recebe no último cenário da história apresentada anteriormente poderia ser visto como um algoritmo que recebe como entrada as imagens da moeda e os intervalos de tempo entre as imagens, e produz como saída o lado da moeda que possui maior probabilidade de cair para cima. Naturalmente, este algoritmo

não necessariamente precisa ser determinístico, podendo ser um algoritmo probabilístico. Entretanto não é difícil visualizar que ele está vinculado a uma restrição temporal, de forma que a resposta que gere possa ser útil para Bob. De nada adianta um algoritmo que responda após a moeda já ter caído no chão.

Generaliza-se esta restrição computacionalmente dizendo que o algoritmo precisa possuir complexidade polinomial. Isso significa que precisa ter complexidade $O(n^k)$, para uma entrada de tamanho n e algum valor k fixo. Um algoritmo com complexidade exponencial não serviria, pois, para alguma entrada n suficientemente grande, o resultado não seria obtido a tempo de ser útil (no exemplo, isso aconteceria caso Alice jogasse a moeda cada vez mais alto, resultando em cada vez mais informações para o computador de Bob processar).

Ao invés de um algoritmo propriamente dito, um adversário poderia ser uma família de circuitos booleanos cujo número de portas (i.e. seu tamanho) é definido por um polinômio da entrada n. Um circuito booleano possui uma entrada de tamanho fixo e uma saída booleana, e é capaz de diferenciar entradas de classes distintas a partir de alguma propriedade. Considerando-se especificamente o problema da distinguibilidade entre seqüências, Goldreich e Meyer em (GOLDREICH; MEYER, 1996) mostraram que existem conjuntos de seqüências indistinguíveis por algoritmos probabilísticos com complexidade polinomial, mas distinguíveis por circuitos de tamanho polinomial. Dado que a definição de fonte perfeita depende justamente da capacidade de se solucionar este problema, é necessário definir um adversário polinomial da seguinte forma:

Definição 2.5 *Sejam P_1 e P_2 polinômios quaisquer e $S_1 = \{S_1^k\}$ e $S_2 = \{S_2^k\}$ dois multiconjuntos tais que cada elemento S_1^k e S_2^k contenham seqüências de bits com comprimento $P_1(k)$. Um **adversário polinomial** é uma coleção de circuitos booleanos $C = \{C_i^k\}$ de tamanho menor ou igual a $P_2(k)$, com entrada de tamanho $i < P_1(k)$ e um bit de saída, tal que, dado como entrada os i primeiros bits de uma seqüência qualquer S_k, o resultado indique se esta pertence a S_1 ou a S_2.*

Por exemplo, se ambos os conjuntos forem puramente aleatórios, por definição não existe forma de distinguir um do outro. Mas note que se S_1 é um conjunto de seqüências geradas por uma fonte qualquer e S_2 é uma fonte puramente aleatória, então um adversário polinomial que consiga distinguir entre os elementos de S_1 e S_2 pode ser utilizado para prever os próximos bits de uma seqüência de S_1 com probabilidade maior que 50%. Em outras palavras, a mesma qualidade que permite que um circuito identifique uma seqüência não aleatória pode ser utilizada como uma vantagem para a previsão da própria seqüência em si. Isso segue diretamente da equivalência entre as definições 2.2 e 2.3 demonstrada por Yao.

O que é importante ficar claro desta definição é o fato de que um adversário polinomial possui recursos polinomiais, ou seja, que são função de um polinômio da entrada. Caso o adversário encontre um problema exponencial (ou até sub-exponencial) para ser resolvido, então o problema será considerado, para todos os fins, insolúvel, pois sempre existirá um tamanho de entrada n a qual não será possível processar em tempo útil.

2.1.1 Geradores de seqüências pseudo-aleatórias

Até o momento, propositadamente, não foi mencionado o termo *pseudo-aleatório*. Uma seqüência pseudo-aleatória é fundamentalmente diferente de uma seqüência aleatória. Bruce Schneier apresenta a seguinte definição:

Definição 2.6 *(SCHNEIER, 1996) Uma seqüência é dita pseudo-aleatória, se ela parecer aleatória, ou seja, se ela passar por todos os testes estatísticos polinomiais conhecidos.*

Formalmente, um gerador de seqüências pseudo-aleatórias G é um algoritmo determinístico que recebe como entrada dois inteiros, $k > 1$ e $n \geq 1$, e possui as seguintes propriedades:

1. G termina com tempo polinomial em n.
2. G utiliza $O(n)$ bits realmente aleatórios.
3. G produz como saída uma seqüência binária α de comprimento n^k, que passe por todos os testes estatísticos polinomiais conhecidos.

Informalmente, um gerador de seqüências pseudo-aleatórias utiliza alguns poucos bits realmente aleatórios, e gera uma seqüência determinística maior que a entrada, que é estatisticamente aleatória. O fato de a entrada ser menor que a saída faz com que a saída necessariamente seja cíclica, mesmo para algum k suficientemente grande. Por este e outros fatores, essa saída é, em última instância, não aleatória e por isso é chamada de pseudo-aleatória.

Os postulados de Golomb sobre seqüências aleatórias trazem uma definição mais precisa das características que uma seqüência binária deve possuir para parecer aleatória. Sinteticamente, os postulados são os seguintes (mais detalhes em (MENEZES; OORSCHOT; VANSTONE, 1996)):

- O número de bits 1 e bits 0 devem ser praticamente iguais.
- Metade das seqüências de bits consecutivos em 1 (*blocks*) deve possuir tamanho um (dois bits 1 em seqüência), um quarto deve possuir tamanho dois (três bits 1 em seqüência), um oitavo deve possuir tamanho três, etc. O mesmo vale para seqüências de bits 0 consecutivos (*gaps*).
- A similaridade entre todas as suas subseqüências deve ser pequena. Esta propriedade é conhecida como *autocorrelação*.

Em (KNUTH, 1973), é apresentada uma lista não exaustiva de testes estatísticos conhecidos. Utilizando-se de análises estatísticas e teóricas, Knuth mostra que os Geradores Lineares Congruentes geram seqüências pseudo-aleatórias que, de fato, parecem aleatórias para todos os testes estatísticos polinomiais conhecidos.

Seqüências que passem por todos os testes estatísticos conhecidos, entretanto, não necessariamente são criptograficamente seguras. Por exemplo, nada garante que as seqüências passarão pelos testes que virão a ser inventados no futuro. Pior que isso, mesmo que passem, elas não necessariamente possuem a característica de serem imprevisíveis, o que seria esperado de uma seqüência realmente aleatória e necessário para que ela seja considerada segura. Por exemplo, os Geradores Lineares Congruentes, estudados por Knuth, não são seguros.

Sobre a insegurança dos Geradores Lineares Congruentes

Um gerador linear congruente possui a seguinte formulação básica: seja um módulo $n > 0$, um multiplicador a tal que $0 < a < n$, um incremento c tal que $0 \leq c < n$ e uma

semente x_0 aleatoriamente escolhida no intervalo $[0, n)$. Dados a, c e n adequadamente escolhidos, a seqüência pseudo-aleatória $\langle x_i \rangle$ é obtida da seguinte forma:

$$x_i = ax_{i-1} + c \mod n \tag{2.6}$$

Apesar de estatisticamente aleatória, essa seqüência não é criptograficamente segura. Por exemplo, são apresentadas em (PLUMSTEAD, 1983) técnicas para inferir os próximos valores da seqüência $\langle x_i \rangle$ mesmo quando os valores a, c e n não são conhecidos. Além disso, diversos trabalhos mostram como prever as seqüências nos casos em que somente alguns elementos da seqüência são fornecidos, e também nos casos em que apenas os bits mais significativos dos elementos x_i são fornecidos.

Esses resultados também se estendem à modificações destes geradores. Um exemplo disso são os *geradores lineares congruentes multivariados*, que também foram demonstrados inseguros. Um gerador multivariado tem a seguinte equação básica.

$$x_i = a_1 x_{i-1} + a_2 x_{i-2} + a_3 x_{i-3} \ldots + a_m x_{i-m} + c \mod n \tag{2.7}$$

Boyar estende o trabalho de Plumstead em (BOYAR, 1989), mostrando que todos os geradores multivariados também são previsíveis mesmo sem o conhecimento do módulo e dos coeficientes utilizados. Entretanto, não há conhecimento de trabalhos mostrando como prever geradores multivariados onde alguns bits de x_i são descartados. De qualquer forma, presume-se que isso não seja particularmente difícil.

O resultado mais importante dessas análises é o de que existem seqüências que são pseudo-aleatórias (que passam por todos os testes estatísticos conhecidos) mas são previsíveis. Isto faz com que seja interessante a substituição de geradores *ad hoc* por geradores que possuam prova de segurança.

Segurança criptográfica

Para ser criptograficamente segura, uma seqüência pseudo-aleatória precisa possuir, incondicionalmente, as características apresentadas na definição 2.3. Essa definição faz com que a seqüência passe por todos os testes estatísticos conhecidos e não conhecidos, pois ela é capaz de enganar todos os adversários polinomiais. De fato, a definição implica mais do que isso, ela afirma que a seqüência deve passar por todos os testes estatísticos polinomiais *concebíveis*.

Finalmente, pode-se apresentar a definição de gerador de seqüências pseudo-aleatórias criptograficamente seguro. Em síntese, um gerador de seqüências de bits pseudo-aleatórias criptograficamente seguro (CSPRBG) é um gerador de seqüências pseudo-aleatórias que é uma fonte perfeita. Formalmente tem-se a seguinte definição.

Definição 2.7 *Seja Q um polinômio, $I_k \subseteq I$ o conjunto de todas as entradas de tamanho k. Seja A um algoritmo determinístico que, na entrada da semente $x \in I_k$ gera uma seqüência s_x de comprimento $Q(k)$. Seja $S_k = \{s_x | x \in I_k\}$. O algoritmo A é um gerador de seqüências pseudo-aleatórias criptograficamente seguro se o conjunto S_k passa pelo teste do próximo bit. Nessas condições, A é dito um* Q-CSPRBG.

É importante ressaltar que, até o momento, não existe nenhuma prova incondicional da existência de CSPRBGs, como dito anteriormente. Na prática, ele são comprovadamente seguros, mas sob a conjectura plausível de que algum problema é *intratável* (i.e. não possui solução em tempo polinomial probabilístico ou por circuito de tamanho polinomial). Apesar de não ser uma prova absoluta, na prática ela é tida como suficiente.

2.2 Condições para existência de CSPRBGs

A seção anterior apresentou os CSPRBGs sob uma perspectiva teórica. O objetivo desta seção é mostrar como a segurança teórica pode, de fato, ser obtida na prática.

Blum e Micali em (BLUM; MICALI, 1984) definiram um modelo de geração de CSPRBGs, e demonstraram suas características. Um primeiro pré-requisito que se espera de um CSPRBG é que ele não seja polinomialmente periódico. O teorema a seguir demonstra como isso iria contra a definição de CSPRBG. Sejam α e β inteiros. Uma seqüência é dita (α, β)-periódica se ela se torna periódica um período menor que β e após até α bits.

Teorema 2.8 *Sejam P_1, P_2 e P_3 polinômios. Seja $Q = P_1 + P_2 + 2P_3 + 1$ e seja G um Q-CSPRBG. Seja δ_k uma fração das sementes de tamanho k para as quais G gera uma seqüência (P_1, P_2)-periódica. Então $\delta_k < \frac{1}{P_3(k)}$ para qualquer k suficientemente grande.*

Prova É fornecida somente a idéia, dado que os detalhes não são importantes. Imagine por contradição que G seja (P_1, P_2)-periódico. Se isto é verdade então existe um algoritmo que enumera os bits da seqüência até ela se tornar periódica e utiliza esta enumeração para prever alguma nova seqüência. Este algoritmo teria uma probabilidade maior que $\frac{1}{2} + \frac{1}{2P_3(k)}$ de prever os próximos bits, portanto, G não seria criptograficamente seguro. □

Este resultado pode até parecer trivial, mas na época em que foi apresentado não era. Ele está intrinsecamente ligado às características impostas às funções que são empregadas na construção do CSPRBG. Essas características são identificadas mais adiante.

Para definir o modelo de CSPRBGs, Blum e Micali se valem das definições de *predicado difícil* e de *função unidirecional*. Nas próximas seções estes termos são discutidos. Em seguida, será apresentada a forma pela qual a junção destas idéias dá origem a um CSPRBG.

2.2.1 Funções unidirecionais

Como qualquer pessoa naturalmente intui o que é uma função unidirecional, é interessante começar a discussão a partir da sua definição formal, que é dada a seguir.

Definição 2.9 *Seja U um conjunto qualquer. Uma **função unidirecional** é uma função $f : U \rightarrow U$ polinomialmente computável, mas não inversível por nenhum algoritmo com complexidade $O(\log |U|)$.*

Informalmente, a definição 2.9 implica que dado $x \in U$ é *fácil* computar $y = f(x)$ mas, dado y, é *difícil* computar $x = f^{-1}(y)$. Em algumas situações, os termos *eficiente* e *não eficiente* são utilizados como sinônimos de fácil e difícil.

Não se sabe se, de fato, funções unidirecionais existem. O que se sabe é que algumas funções podem ser concebidas baseadas em problemas para os quais não se conhece solução polinomial, os chamados *problemas difíceis*. A tabela 2.1 apresenta uma lista de problemas difíceis clássicos e sua descrição formal.

Um conceito importante que aparece na tabela 2.1 é o chamado *símbolo de Jacobi*, que é muito utilizado em teoria dos números. Ele é melhor definido em função de um outro conceito chamado símbolo de Legendre, que é definido da seguinte forma:

Tabela 2.1: Definições de alguns problemas difíceis, para os quais não se conhece solução polinomial.

Problemas difíceis clássicos	
Fatoração	Dado um número inteiro n, encontrar a sua decomposição em fatores primos na forma $n = p_1^{e_1} p_2^{e_2} p_3^{e_3} .. p_i^{e_i}$, sendo cada valor p_i um primo distinto e cada valor $e_i > 0$.
RSA	Dado um número inteiro n que é o produto de dois números primos ímpares distintos p e q, um número inteiro e tal que $mdc(e, (p-1)(q-1)) = 1$ e um inteiro c, encontrar o valor m tal que $m^e \equiv c \mod n$.
Resíduo quadrático	Dado um inteiro composto ímpar n, e um valor inteiro a tal que o símbolo de Jacobi $(\frac{a}{n}) = 1$, decidir se a é ou não um resíduo quadrático módulo n. (a é um resíduo quadrático módulo n se e somente se existe um número x tal que $a = x^2 \mod n$.).
Raiz quadrada $\mod n$	Dado um inteiro composto n e um valor a que é um resíduo quadrático módulo n, encontrar a raiz quadrada de a módulo n, ou seja, o valor inteiro x tal que $a = x^2 \mod n$.
Logaritmo discreto	Dado um inteiro primo p, um gerador α do grupo multiplicativo $\mathbb{Z}_p^*$ e um elemento $\beta \in \mathbb{Z}_p^*$, encontrar um inteiro x tal que $\alpha^x \equiv \beta \mod p$.

Definição 2.10 *Seja p um número primo positivo. Para qualquer valor $a \geq 0$, o* símbolo de Legendre $\left(\frac{a}{p}\right)$ *é definido como*

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{se } a \equiv 0 \mod p \\ 1 & \text{se a congruência } x^2 \equiv a \mod p \text{ possui solução} \\ -1 & \text{se a congruência } x^2 \equiv a \mod p \text{ não possui solução} \end{cases}$$

O símbolo de Jacobi, por sua vez, é uma generalização do símbolo de Legendre para valores não primos:

Definição 2.11 *Seja n um número positivo com respectiva fatoração em componentes primos $p_1^{e_1} p_2^{e_2} .. p_k^{e_k}$. Para qualquer valor $a \geq 0$, o* símbolo de Jacobi $\left(\frac{a}{n}\right)$ *é definido como o produto dos símbolos de Legendre relativamente aos componentes primos de n, na forma da expressão*

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i}.$$

É importante ressaltar que existem formas de se calcular o símbolo de Jacobi sem saber a fatoração de n. Isso esclarece a definição do problema do resíduo quadrático proposto na tabela pois se n é o produto de dois primos então se a é um resíduo quadrático módulo cada um destes primos ou então se não é resíduo quadrático módulo nenhum deles, em ambos os casos o símbolo de Jacobi $\left(\frac{a}{n}\right)$ resulta em 1. Entretanto, sabendo apenas isto é difícil decidir se a é um resíduo quadrático módulo n sem calcular diretamente o símbolo de Legendre relativamente a pelo menos um dos fatores de n.

A lista da tabela 2.1 não é exaustiva, considerando apenas alguns dos problemas mais conhecidos. É importante considerar o fato de que não conhecer soluções polinomiais para os problemas não implica que essas soluções não existam. Entretanto, o fato dos problemas terem resistido por tanto tempo à busca por soluções faz com que seja conjecturado que elas não serão encontradas tão cedo, mesmo que existam. É importante também atentar aos detalhes das definições. Por exemplo, existem algoritmos eficientes para decidir se um valor a é um resíduo quadrático quando tomado módulo um número primo p, mas não quando tomado módulo um número composto n cujos fatores não são conhecidos. Além disso, isso implica que o problema do resíduo quadrático é redutível ao problema da fatoração, dado que o conhecimento dos fatores de n permite resolver o problema em tempo polinomial.

Considerando-se que existem problemas que são difíceis, é possível definir funções que são fortes candidatas à funções unidirecionais. Três possibilidades são apresentadas:

Exponenciação módulo p. Seja p um inteiro primo ímpar e α um gerador do grupo multiplicativo $\mathbb{Z}_p^*$. A função $f : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^*$ é definida como $f(x) = \alpha^x \mod p$. Esta função é unidirecional dado que o problema do logaritmo discreto é difícil.

Função RSA. Sejam p e q primos ímpares distintos, $n = pq$ e e tal que $mdc(e, (p-1)(q-1)) = 1$. A função $f : \mathbb{Z}_n \rightarrow \mathbb{Z}_n$ é definida como $f(x) = x^e \mod n$. Esta função é unidirecional dado que o problema RSA é difícil.

Função Rabin. Sejam p e q primos ímpares distintos, ambos congruentes à 3 módulo 4 (i.e. $p \equiv q \equiv 3 \mod 4$. Seja $\mathbb{Q}_n \subseteq \mathbb{Z}_n$ o conjunto de todos os resíduos quadráticos em $\mathbb{Z}_n$, ou seja, todos os valores $a \in \mathbb{Z}_n$ tal que para algum valor $x \in \mathbb{Z}_n$ tem-se que $a = x^2 \mod n$. A função $f : \mathbb{Q}_n \rightarrow \mathbb{Q}_n$ é definida como $f(x) = x^2 \mod n$. Esta função é unidirecional dado que a extração de raízes quadradas módulo um número composto cujos fatores primos são desconhecidos é difícil.

2.2.2 Predicados difíceis

Considere a função unidirecional f tal que $y = f(x)$. Apesar das funções unidirecionais serem não inversíveis em tempo polinomial, isso não implica necessariamente que todos os bits de x sejam difíceis de serem recuperados em todos os casos. Em outras palavras, alguns bits podem ser mais fáceis de serem calculados do que outros. Os seguintes exemplos ajudam a clarificar o problema.

Seja $y = g^x \mod p$ onde g é um gerador do grupo multiplicativo $\mathbb{Z}_p^*$, e p um número primo tal que $p = 2^s q + 1$, q ímpar. Os s bits menos significativos de x podem ser recuperados de forma "fácil" através de um algoritmo conhecido como *Pohlig-Hellman* (MENEZES; OORSCHOT; VANSTONE, 1996). De fato, este algoritmo é capaz de calcular o logaritmo discreto de qualquer grupo baseado em primos do tipo $2^m + 1$ em tempo polinomial.

Um outro fato interessante é o seguinte. No caso de qualquer tipo de exponenciação módulo um primo p, o bit menos significativo de x é zero se e somente se y é um resíduo quadrático módulo p. Essa determinação também é simples, e, como visto anteriormente é facilmente obtida via cálculo do símbolo de Jacobi $(\frac{y}{p})$.

Certamente, se f é alguma função unidirecional, então, para qualquer constante c, mais do que $c \log\log n$ bits precisam ser difíceis de serem calculados, n sendo o argumento de f. Isso é óbvio, pois, caso contrário, seria possível inverter a função encontrando os bits fáceis da forma fácil e enumerando os bits difíceis. Por exemplo, caso a função possuísse exatamente $c \log\log n$ bits difíceis, a enumeração destes seria polinomial em $\log n$, pois $2^{c \log\log n} = O(\log n)$.

A melhor tradução possível para a idéia de predicado difícil é a de uma função que apresenta os bits difíceis de uma função unidirecional. A definição formal é a seguinte.

Definição 2.12 *Seja S_n um conjunto qualquer de strings de tamanho n, e $f : S_n \to S_n$ uma função unidirecional. Uma função booleana $B : S_n \to \{0,1\}$ é um **predicado difícil** para f se:*

1. *$B(x)$ é fácil de ser calculado para todo $x \in S_n$. Nesse caso, dizemos que **B** é **acessível**.*

2. *um oráculo que calcule $B(x)$ corretamente com probabilidade significativamente maior que $\frac{1}{2}$ tendo acesso a somente $f(x)$ (mas não ao próprio x) pode ser utilizado para obter x em tempo polinomial.*

Em outras palavras, B é um predicado difícil se e somente se obter alguma informação sobre $B(x)$ a partir de $f(x)$ é tão difícil quanto inverter f. O item dois acima mostra em que condições pode-se assumir que $B(x)$ é tão difícil de obter quanto inverter f. Basicamente predicados difíceis sintetizam a idéia de esconder bits em uma função unidirecional. Note que em inglês são empregados os termos *hard bits* e *hard predicate* (que em alguns casos também é chamado de *unapproximable predicate*).

Apesar do conceito de predicado difícil ser simples, ele é suficientemente abstrato para que seja complicado entendê-lo somente pela definição. Blum e Micali definiram o seu gerador em termos de um predicado difícil para o logaritmo discreto. A seguir, este predicado será descrito em detalhes de forma a ilustrar o conceito.

O problema da raiz quadrada principal

Alguns conceitos preliminares ainda não apresentados são necessários para que este problema seja definido.

Seja $\mathbb{Q}_p \in \mathbb{Z}_p^*$ o conjunto de resíduos quadráticos módulo p. Sabe-se que todo o resíduo quadrático $a \in \mathbb{Q}_p$, em termos de um gerador g, é da forma $a = g^{2s} \mod p$, $s \in [1, \frac{p-1}{2}]$. Esta representação de a em termos de s e g é única. Além disso, todo o resíduo quadrático a possui duas raízes quadradas distintas $\pm r$, sendo $r_+ = g^s \mod p$ e $r_- = g^{s+\frac{p-1}{2}} \mod p$. A **raiz quadrada principal** de a é o número univocamente identificado $r_+ = g^s \mod p$. O número univocamente identificado por $r_- = g^{s+\frac{p-1}{2}} \mod p$ é chamado de **raiz quadrada não principal** de a. Note que raiz quadrada é definida em termos do módulo e da base g.

Além disso, sabe-se que existe um algoritmo polinomial para determinar se um número a é um resíduo quadrático módulo p (símbolo de Jacobi), e também da existência de um algoritmo polinomial para calcular ambas as raízes de um resíduo quadrático. Entretanto, dado um resíduo quadrático $a = g^{2s} \mod p$ e suas raízes quadradas, decidir qual das raízes é a sua raiz principal é tão difícil quanto calcular o valor $2s$ a partir de a (i.e., calcular o logaritmo módulo p de a na base g).

Seja $f : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^*$ definida como $f(x) = g^x \mod p$ e g um gerador do grupo multiplicativo $\mathbb{Z}_p^*$. Pode-se definir o seguinte predicado difícil $B : \mathbb{Z}_p^* \rightarrow \{0, 1\}$ tal que dado $x_i = g^{x_{i-1}} \mod p$, $B(x_i) = 0$ se x_i é a raiz principal do resíduo quadrático representado univocamente por $g^{2x_{i-1}} \mod p$, e $B(x_i) = 1$ caso contrário.

É bom lembrar que B deve ser polinomialmente computável. Imediatamente é possível ver que isto não é um problema. Dado x_{i-1}, $g^{x_{i-1}} \mod p$ é uma raiz principal se e somente se $x_{i-1} \leq \frac{p-1}{2}$. Caso se tenha que $x_{i-1} > \frac{p-1}{2}$, então $g^{x_{i-1}} \mod p$ é uma raiz não principal, e x_{i-1} pode ser escrito na forma $s + \frac{p-1}{2}$, como dito anteriormente. Logo, o quadrado $g^{x_{i-1}} \mod p$ não é $g^{2x_{i-1}} \mod p$, mas é $g^{2s} \mod p$. Em síntese, o predicado é definido como:

$$B(x) = \begin{cases} 1 & \text{se} \quad x \leq \frac{p-1}{2}; \\ 0 & \text{se} \quad x > \frac{p-1}{2} \end{cases}$$

Para que B seja de fato um predicado difícil, é preciso mostrar que um oráculo que calcule $B(x)$ dado $f(x)$ pode ser utilizado para inverter f. Isso prova que, dado que a função é difícil, tal oráculo não existe e, portanto, não existe forma de se calcular os bits $B(x)$ mais eficientemente do que tentando adivinhá-los.

Teorema 2.13 *Seja p um primo ímpar, g um gerador de $\mathbb{Z}_p^*$, $x, y \in \mathbb{Z}_p^*$ tal que $y = g^x \mod p$. Seja $\Theta(p, g, y)$ um oráculo que calcule $B(x)$. Então Θ pode ser utilizado em um algoritmo polinomial para calcular x dado y.*

Prova O algoritmo 1 é a formalização do procedimento. São fornecidos ao algoritmo os valores p, g e y como utilizados nos exemplos até agora, com $y = g^x \mod p$. Somente fazendo chamadas ao oráculo Θ, o algoritmo calcula x bit a bit, da direita para a esquerda, e retorna o valor completo. O algoritmo também utiliza chamadas para as funções $ResiduoQuadratico(y, p)$, que retorna verdadeiro se y é um resíduo quadrático módulo p.

Também são utilizadas as operações $RaizQuadradaMod(y, p, \text{ var } r_1, \text{ var } r_2)$ que calcula as raízes quadradas de y módulo p e as retorna nos parâmetros r_1 e r_2, a multiplicação yg^{-1} que tem o efeito de subtrair 1 do índice de y e $Concat(a, b)$ retorna a concatenação de a com b. Como já foi dito, todas estas operações são polinomiais, sendo $\Theta = O(1)$ por definição.

Basicamente, o algoritmo *Logaritmo Discreto* extrai o índice de y bit a bit com testes de residuosidade quadrática. Caso x seja ímpar, ele o torna par subtraindo 1 e utiliza o oráculo para manter na variável y a raiz de g^{2s} que possui a forma de raiz principal(g^s). O algoritmo claramente é linear no número de bits de x. Como todas as outras operações são polinomiais, então ele é polinomial. □

É bom ficar claro que o que o algoritmo prova é justamente que este oráculo não existe, pois, se existisse, seria muito fácil calcular um logaritmo discreto. Existe, entretanto, um outro problema a ser resolvido. Para que uma *string* seja realmente aleatória, a probabilidade de calcular o próximo bit deve ser praticamente $\frac{1}{2}$. O que acontece se o oráculo errar freqüentemente, mas em um número significativo de vezes acertar? Um oráculo deste tipo daria uma vantagem não desprezível para a adivinhação de $B(x)$, e este caso precisa ser considerado.

Concentrando uma vantagem estocástica

Algoritmo 1 $Logaritmo\_discreto(y, p, g)$

```
1: Indice := 0;
2: while y ≠ 1 do
3:    if ResiduoQuadratico(y, p) then
4:       Indice := Concat(0, Indice);
5:       RaizQuadradaMod(y, p, r1, r2);
6:       if Θ(p, g, r1) = 1 then
7:          y := r1;
8:       else
9:          y:= r2;
10:      end if
11:   else
12:      Indice := Concat(1, Indice);
13:      y := yg^-1 mod p;
14:   end if
15: end while
16: return Indice;
```

Blum e Micali apresentam um método matemático chamado por eles de *concentrar uma vantagem estocástica* (*concentrate a stochastic advantage*). Basicamente, eles mostram como utilizar um oráculo que responde a maioria das instâncias de consulta corretamente, em um oráculo que responde uma instância específica com uma probabilidade arbitrariamente alta de estar correto. Para tanto, eles se utilizam de uma versão específica da chamada "Lei Fraca dos Grandes Números", advinda da teoria da probabilidade, que diz o seguinte.

Definição 2.14 *Se $y_1 \dots y_k$, são variáveis binárias independentes tal que $y_i = 1$ com probabilidade α e $S = \sum_{i=0}^{k} y_i$ é o somatório absoluto das variáveis, então, para dois números reais ε e δ,*

$$k > \frac{1}{4\varepsilon^2\delta}$$

implica

$$Pr\left(\left|\frac{S}{k} - \alpha\right| > \varepsilon\right) < \delta.$$

Em outras palavras, a probabilidade de que a média entre as variáveis y_i se desvie da sua distribuição individual, para algum k fixo, está limitada por constantes fixas. Mais especificamente, esta probabilidade é polinomial em ε^{-1} e δ^{-1}. Utilizando estes limites polinomiais para um k constante, eles mostram que um algoritmo probabilístico para calcular o valor $B(x)$ seria polinomial nestas constantes. Assim, eles provam que um oráculo que forneça uma vantagem significativa (i.e. polinomial) para calcular $B(x)$ é automaticamente transposto para um algoritmo que calcula o logaritmo discreto com exatamente a mesma probabilidade. E, mais uma vez, considerando-se que não existe algoritmo polinomial probabilístico com tal vantagem para o logaritmo discreto, não existe tal oráculo.

2.2.3 Predicados difíceis de k-bits

Na época da escrita do seu artigo, Blum e Micali conjecturavam como seria possível esconder mais de 1 bit em um predicado. Como já dito anteriormente, uma função uni-

direcional necessariamente possui diversos bits difíceis de serem obtidos, ou a sua busca por enumeração seria polinomial.

Definição 2.15 *Seja S_n um conjunto qualquer de strings de tamanho n, e $f : S_n \rightarrow S_n$ uma função unidirecional. Uma função $B^k : x \in S_n \rightarrow \{0,1\}^k$ é um* ***predicado difícil de k-bits*** *para f se:*

1. *$B^k(x)$ é fácil de ser calculado dado $x \in S_n$;*
2. *Para cada predicado booleano $B_i : \{0,1\}^k \rightarrow \{0,1\}$, a informação $B_i(B^k(f(x)))$ permite inverter f.*

Em outras palavras, $B^k(x)$ é um predicado difícil caso **qualquer** informação sobre B^k possa ser utilizada para inverter f.

Peralta em (PERALTA, 1986) considera que bits que obedeçam à definição acima têm "segurança simultânea fraca" (*weak simultaneously security*). Ele então redefine um predicado B^k de forma a ter uma *segurança forte*.

Definição 2.16 *Seja S_n um conjunto qualquer de strings de tamanho n, e $f : S_n \rightarrow S_n$ uma função unidirecional. Os bits $x_1, \ldots x_k$ são simultaneamente seguros se para qualquer l tal que $1 \leq l < k$, um oráculo que, tendo como entrada $f(x)$ e $x_1 \ldots x_l$, preveja o bit x_{l+1} com uma probabilidade maior que $\frac{1}{2} + \frac{1}{c \log \log n}$ (para qualquer constante c) pode ser utilizado para construir um algoritmo que inverta f em tempo probabilístico polinomial.*

Peralta então mostra que a definição 2.16 implica a 2.15. Na prática, esta nova definição significa que qualquer subseqüência do predicado deve ser imprevisível à direita, exatamente nos termos da definição 2.3 feita por Yao. Não é difícil perceber que essa condição é muito mais difícil de ser provada do que a segurança de um predicado que esconde somente um bit.

Em 1989, entretanto, Goldreich e Levin provaram que qualquer função unidirecional possui um número logarítmico de predicados booleanos difíceis e mostraram como construí-los genericamente para qualquer função (GOLDREICH; LEVIN, 1989). Além disso, eles mostraram que estes predicados são simultaneamente seguros. Isso é condizente com o fato de que caso a função possuísse menos bits difíceis, ela seria polinomialmente inversível e, portanto, não unidirecional.

2.2.4 Arquitetura de um CSPRBG

Naturalmente, Blum e Micali ainda não conheciam diversas constatações feitas após o seu artigo. Por esse motivo, a arquitetura proposta por eles era consideravelmente mais complexa que a que será apresentada. A definição atual de um CSPRBG, como pode ser vista a seguir, é de fato relativamente simples.

Definição 2.17 *Seja Q um polinômio, S_n o conjunto de strings binárias de tamanho n. Seja $f : S_n \rightarrow S_n$ uma função unidirecional e $B^k : S_n \rightarrow \{0,1\}^k$ um predicado difícil de k-bits para f, sendo $k \geq 1$. Sejam:*

1. *$f : x \in S_n \rightarrow S_n$ uma função computável em tempo polinomial para qualquer $x \in S_n$;*

2. $f : S'_n \rightarrow S'_n$ *uma permutação sobre um conjunto de valores* $S'_n \in S_n$*;*

3. $x_0 \in S'_n$ *um valor uniformemente escolhido entre os possíveis valores de* S'_n *em tempo probabilístico polinomial.*

Então, o seguinte pseudo-algoritmo é um Q-CSPRBG. O algoritmo recebe como entrada o número de bits a serem gerados $n > 0$*. Ele acumula na variável* Resultado *os bits pseudo-aleatórios, e depois os retorna.*

Algoritmo 2 $Q\_CSPRBG(n)$

```
1: Escolher uniformemente um x ∈ S'_n;
2: Resultado := " ";
3: i := ⌈Q(n)/k⌉;
4: while i > 0 do
5:     x := f(x);
6:     Bits := B^k(x);
7:     Resultado := Concat(Resultado, Bits);
8: end while
9: return Resultado;
```

Note que a simplicidade do algoritmo esconde as noções fundamentais de função unidirecional e de predicados difíceis. A saída é comprovadamente aleatória porque a função é não inversível e porque os bits que são expostos são bits advindos de um predicado difícil para a função. Desta forma, tem-se uma seqüência de bits não distinguível de uma seqüência puramente aleatória por qualquer adversário limitado polinomialmente, o que era o objetivo desde o início.

Uma diferença desta arquitetura para a arquitetura definida por Blum e Micali está no fato de que, na arquitetura deles, os bits eram retornados na ordem inversa à do algoritmo Q-CSPRBG. Pode-se ver intuitivamente que a imprevisibilidade está amparada na inversão da função f. Entretanto, pelo teorema 2.4, sabe-se que a ordem em que os bits são retornados não importa. Além do mais, ainda não existia o conceito de bits simultaneamente seguros, e, portanto, o gerador era consideravelmente mais ineficiente.

2.3 Implementações e otimizações de CSPRBGs

Desde a pioneira arquitetura de Blum e Micali, baseada no problema da raiz quadrada principal, diversos outros geradores criptograficamente seguros foram propostos. Devido à arquitetura simples destes geradores, os principais avanços dizem respeito aos predicados difíceis utilizados. Este seção será dedicada a estas variações e otimizações.

2.3.1 Otimizações no gerador Blum-Micali

Blum e Micali mostraram como esconder um bit através da função de exponenciação em módulo. Basicamente, um atacante precisaria adivinhar em qual das metades do grupo de possíveis expoentes para a função o índice se encontra, isto é, se o expoente de y na base g módulo p é maior ou menor que $\frac{p-1}{2}$.

Não muito tempo depois, Long e Wigderson (LONG; WIGDERSON, 1983) mostraram que era possível pegar a faixa superior de expoentes e a faixa inferior e dividi-las em duas subfaixas, exatamente pelo mesmo critério (maior ou menor que a metade da faixa).

Desta forma, a faixa zero (inferior) teria duas subfaixas, zero-zero e zero-um, e a faixa inferior teria as faixas um-zero e um-um. Eles mostraram que isto seria um predicado de dois bits para a função exponenciação e, desta forma, um gerador que se utilizasse disto poderia retornar com segurança dois bits ao invés de apenas um.

Generalizando a idéia, eles vão mais longe. Por um raciocínio recursivo, eles provam que seria possível dividir a faixa de valores em $c \log \log p$ subfaixas distintas. Assim, é possível retornar $c \log \log p$ bits simultaneamente. Como de praxe, eles demonstram que um oráculo que forneça qualquer informação sobre em qual faixa o índice se encontra poderia ser utilizado para calcular o logaritmo do número.

Um outro avanço importante foi apresentado por Peralta em (PERALTA, 1986). Já foi dito que para logaritmos módulo um primo do tipo $p = 2^s q + 1$, os s bits menos significativos do índice são fáceis de serem calculados. Ele então mostra que cada um dos $c \log \log p$ bits menos significativos do logaritmo a partir do bit $s + 1$ formam um predicado difícil. Uma diferença notável do seu trabalho é o fato de que ele trata dos bits do índice de fato. Este resultado difere dos anteriores na medida em que os anteriores diziam respeito a bits que representavam faixas onde o índice se encontrava, bits estes que possivelmente não seriam iguais aos bits mais significativos do índice real.

Finalmente, como uma espécie de golpe de misericórdia, Schnorr mostra que, para este gerador, todos os bits, com exceção dos s menos significativos, são individualmente seguros. Ele prova em (SCHNORR, 1998) que qualquer seqüência de j bits consecutivos que é segura implica a segurança de todas as outras seqüências de j bits. Ele demonstra esse resultado simplesmente mostrando como obter a equivalência entre seqüências de j bits através de *shifts* do índice, utilizando apenas transformações polinomiais. Em síntese, qualquer seqüência de j bits é polinomialmente equivalente a qualquer outra seqüência de j bits.

2.3.2 Geradores baseados em fatoração

Uma outra "facção", bastante populosa, de geradores criptograficamente seguros é a dos baseados na fatoração. Por exemplo, existem algoritmos construídos sobre problema RSA e sobre a dificuldade de se decidir se um número é um resíduo quadrático módulo um número composto. Ambos os problemas são redutíveis à fatoração. Algumas construções são apresentadas.

O primeiro gerador baseado na fatoração é conhecido como Blum-Blum-Shub (ou gerador BBS), apresentado em detalhes em (BLUM; BLUM; SHUB, 1986). Este gerador apareceu pela primeira vez em (BLUM; BLUM; SHUB, 1983). Um inteiro Blum N é um inteiro que é produto de dois primos ambos congruentes a 3 módulo 4. Todo o resíduo quadrático módulo N possui quatro raízes, sendo que exatamente uma também é um resíduo quadrático. O gerador então utiliza a operação de elevar ao quadrado módulo N para gerar uma permutação entre os resíduos quadráticos módulo N, e retorna, a cada passo o último bit, do resultado. Vazirani em (VAZIRANI; VAZIRANI, 1985) mostrou que é possível se obter com segurança os $c \log \log N$ bits menos significativos de cada resultado. Ele mostrou também, no mesmo artigo, que a segurança deste gerador depende somente da capacidade de se fatorar N.

Já um gerador baseado no problema RSA possui a seguinte construção. Sejam p e q dois primos de tamanho aproximado. Calcular $n = pq$. Calcular também $\phi = (p-1)(q-1)$ e escolher um número e tal que $0 < e < \phi$ e que não possua divisor em comum com ϕ. O gerador funciona elevando ao expoente e uma semente menor que n, e retornando o bit menos significativo do resultado. Baseados neste gerador, Micali e Schnorr mostraram

em (MICALI; SCHNORR, 1991) um resultado bastante revolucionário. Eles provaram que poderiam ser retornados com segurança os $k = \lfloor(((\log n) + 1)(1 - \frac{2}{e})\rfloor$ bits menos significativos de cada iteração. Isso significa que o gerador pode fornecer um número polinomial de bits e não logarítmico como em todos os outros resultados conhecidos.

Hastad, Schrift e Shamir (HASTAD; SCHRIFT; SHAMIR, 1993) demonstraram um resultado parecido com o de Micali e Schnorr para um gerador baseado na exponenciação módulo um número composto. Dado um inteiro Blum $N = pq$, a exponenciação de uma base fixa em um número qualquer módulo N pode retornar até $\frac{N}{2} + O(\log N)$ bits por operação, dado que a fatoração de N é difícil. Nesta construção, por outro lado, não é possível retornar exatamente os bits do próprio resultado pois eles não são bits simultaneamente seguros. Para tal, é utilizada uma função de *hash*, de forma que o resultado volte a ser uniformemente distribuído entre os valores possíveis, e, portanto, indistinguível de uma seqüência puramente aleatória.

Para este mesmo gerador, Goldreich e Rosen em (GOLDREICH; ROSEN, 2000) demonstraram o fato de que fazer a exponenciação de todos os bits do resultado para obter o próximo estágio é equivalente a fazer a exponenciação de apenas metade dos bits do resultado anterior. Esse resultado implica um gerador muito mais eficiente, dado que a exponenciação em si pode ser feita na metade do tempo.

Em um último resultado de otimização para este gerador de exponenciação módulo um número composto, (DEDIC; REYZIN; VADHAN, 2002) apresenta três avanços importantes. O primeiro diz respeito a uma forma de se remover a função de *hash* que o algoritmo requeria, aumentando a eficiência e aumentando o número de bits retornáveis com segurança. O segundo avanço é uma aceleração do cálculo da exponenciação pré-calculando-se um conjunto de valores. Mais especificamente, é pré-computada uma seqüência determinada de expoentes da base que está sendo utilizada. Desta forma, a combinação dos valores dessa seqüência permite gerar os valores intermediários necessários para a exponenciação de forma muito mais rápida. A terceira otimização é uma proposta de como acelerar ainda mais os cálculos reutilizando-se um mesmo valor que foi calculado a partir da semente do gerador. Uma característica importante destas otimizações é que muitas delas são aplicáveis a diversas outras construções de geradores.

2.4 Criptografia de chave pública probabilística

Criptografia de chave pública probabilística é um conceito que se fundamenta completamente da teoria de geradores de bits pseudo-aleatórios criptograficamente seguros. Ao longo desta dissertação, será utilizado o termo *criptografia probabilística* como sinônimo de *criptografia de chave pública probabilística*, apesar de o conceito também ser aplicável a algoritmos de chave simétrica.

Em 1984, Shafi Goldwasser e Silvio Micali (o mesmo da teoria de CSPRBGs) apresentam pela primeira vez o conceito de criptografia probabilística no seu artigo seminal (GOLDWASSER; MICALI, 1984). O artigo inicia afirmando que é importante para um algoritmo de chave pública possuir a seguinte característica:

> *Tudo que é eficientemente computável sobre o texto claro a partir do texto cifrado também é computável sem o texto cifrado.*

Valendo-se da teoria da complexidade, a criptografia probabilística tem como meta tornar impraticável para um atacante obter *qualquer* informação sobre o texto claro. Isso é uma restrição muito mais complicada de ser obtida do que pode parecer à primeira

vista. A melhor forma de apresentar a idéia, é mostrar os problemas que esta característica soluciona no modelo de chave pública determinístico.

Inicialmente, é necessário verificar as característica dos de chave pública tradicionais. Diferentemente dos geradores de bits pseudo-aleatórios que utilizam funções unidirecionais, no modelo de criptografia de chave pública determinístico trabalha-se com a idéia de *funções alçapão* (em inglês, *trapdoor function*). Uma função alçapão é definida da seguinte forma:

Definição 2.18 *Sejam U, V e Z conjuntos quaisquer. Uma **função alçapão** é uma função $f : U \rightarrow Z$ polinomialmente computável cuja função $f^{-1} : Z \rightarrow U$ não é diretamente computável por nenhum algoritmo com complexidade $O(\log |U|)$ mas existe uma função $f'^{-1} : Z, V \rightarrow U$ polinomialmente computável para algum $v \in V$ tal que $f'^{-1}(z, v) = u \rightarrow f(u) = z$.*

Em outras palavras, uma função alçapão é inversível somente quando uma informação extra (o alçapão) está disponível. No contexto de criptografia de chave pública, v seria a chave privada e a função f seria a chave pública. Naturalmente, se f^{-1} não é polinomialmente computável, por definição v também não é.

Continuando, os autores expoem os principais defeitos da criptografia de chave pública baseada em funções alçapão.

1. *O fato de f ser uma função alçapão não elimina a possibilidade de se calcular x a partir de $f(x)$, quando x possui alguma forma especial.* Normalmente, mensagens não se constituem de números aleatórios e possuem alguma estrutura específica, como os caracteres ASCII de um texto. Esta estrutura, em alguns casos, poderia ajudar na inversão da função. Como um exemplo simples, considere a função Rabin, como definida na seção 2.2.1. Caso $f(x) \equiv 4 \mod n$, sabe-se que $\sqrt{4} \equiv 2, n - 2 \mod n$ para **qualquer** $n > 4$, e, portanto, é fácil obter dois possíveis valores de x. Quando $n = p \times q$ então existem mais duas raízes, estas sim difíceis de serem calculadas. Entretanto, metade delas é *fácil*. Da mesma forma, $f(0) = 0$ é outra codificação com solução trivial.

2. *Uma função alçapão não garante que não se consiga obter informações parciais a respeito de x a partir de $f(x)$.* Informações parciais a respeito de x são informações do tipo x é par, x é ímpar, ou mesmo alguns bits específicos de x. Em alguns protocolos esta informação pode ser mais do que suficiente para comprometer a segurança do sistema.

Uma ameaça real que decorre destas deficiências aparece quando o *espaço de mensagens* não é uniforme dentre os possíveis valores do domínio da função utilizada. Isso acontece quando, por exemplo, sabe-se de antemão que somente algumas mensagens são possíveis. Se o número de mensagens for pequeno o suficiente, é possível cifrar todas as mensagens, e a comparação com uma base pré-computada de textos cifrados seria o suficiente para inverter polinomialmente a função. O pior caso é o da transmissão de apenas 1 bit de informação.

Os autores propõe então a substituição da função alçapão por uma outra função, chamada *predicado difícil alçapão*. Relativamente à definição de predicado difícil (veja definição 2.12), um predicado difícil alçapão possui a mesma diferença que uma função alçapão possui quando comparada com uma função unidirecional.

Definição 2.19 *Seja $f : U \to Z$ uma função alçapão cuja informação alçapão é $v \in V$. A função $B : U \to \{0, 1\}$ é um **predicado difícil alçapão** se*

1. *$B(x)$ é fácil de ser calculado para todo $x \in U$. Nesse caso, dizemos que B é **acessível**.*
2. *existe uma função f' tal que $f'(f(x), v) = B(x)$ para todo $x \in U$.*
3. *um oráculo que calcule $B(x)$ corretamente com probabilidade significativamente maior que $\frac{1}{2}$, tendo acesso a somente $f(x)$, pode ser utilizado para obter x em tempo polinomial.*

Como definida acima, a função B mapeia cada valor do conjunto U para um dos valores do conjunto $\{0, 1\}$. Um algoritmo probabilístico pode se utilizar deste fato para codificar bits da seguinte forma: um bit 1 é codificado como um elemento aleatório $f(x) \in U$ tal que $B(x) = 1$ e um bit 0 da mesma forma. Como é difícil calcular $B(x)$ a partir de $f(x)$ (a não ser que seja possível inverter a função f sem a informação de alçapão), então o algoritmo é seguro. Além disso, como existem diversas codificações para cada bit de informação, é impraticável obter qualquer informação sobre o dado sendo cifrado.

2.4.1 O algoritmo probabilístico Goldwasser-Micali

A fim de tornar a explicação mais palpável, é interessante apresentar o algoritmo que foi proposto pelos autores no mesmo artigo onde o conceito de criptografia probabilística foi introduzido. O algoritmo é baseado na dificuldade de se decidir se um número é ou não um resíduo quadrático módulo um número composto cujos componentes primos não são conhecidos. O algoritmo, então, cifra bits individualmente como explicado a seguir.

Seja $n = p \times q$ onde p e q sejam primos grandes e a fatoração de n seja impraticável. Seja Q_n o conjunto de valores que são resíduos quadráticos módulo n. Seja

$$\overline{Q}_n = \{x | x \in \mathbb{Z}_n \text{ e } \left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = -1\}$$

o conjunto de valores que possuem símbolo de Jacobi igual a -1 módulo p e q (e portanto, possuem símbolo de Jacobi igual a 1 módulo n). Os elementos de $\overline{Q}_n$ são chamados de *pseudo-quadrados* módulo n.

Seja, também, um valor qualquer $m \in \overline{Q}_n$. Os valores n e m são a chave pública e a fatoração de n é a chave privada. Seja $r \in \mathbb{Z}_n$ um valor escolhido aleatoriamente e b o bit a ser cifrado. A propriedade utilizada para a cifragem é a de que todo o pseudo-quadrado multiplicado por um resíduo quadrático, resulta em um pseudo-quadrado. A cifragem é feita calculando-se o valor

$$C = m^b r^2 \mod n.$$

Note que r^2 é, necessariamente, um resíduo quadrático e, conseqüentemente, mr^2 será um pseudo-quadrado. Portanto $C = m^b r^2 \mod n$ será um pseudo-quadrado se $b = 1$ e será um resíduo-quadrático se $b = 0$.

A decifragem é trivial, pois, por definição, um pseudo-quadrado possui símbolo Jacobi igual a -1 relativamente a qualquer fator de n. Portanto

$$b = \begin{cases} 1 & \text{se } \left(\frac{C}{p}\right) = -1 \\ 0 & \text{se } \left(\frac{C}{p}\right) = 1 \end{cases}$$

2.4.2 O algoritmo Blum-Goldwasser

Um dos problemas do esquema Goldwasser-Micali é o fato de que o texto cifrado acaba sendo muito maior que o texto claro original. De fato, como para cada bit a ser cifrado é necessário um elemento de $\mathbb{Z}_n$, o tamanho total do texto cifrado será $b \log n$ onde b é o número de bits a serem cifrados. Para solucionar este problema, Blum e Goldwasser em (BLUM; GOLDWASSER, 1985) sugerem e demonstram a segurança de um novo algoritmo probabilístico baseado no gerador de bits pseudo-aleatórios criptograficamente seguros Blum-Blum-Schub, apresentado na seção 2.3.2.

Como chave privada, tem-se os primos $p, q \equiv 3 \mod 4$ e como chave pública o valor $n = p \times q$. A cifragem ocorre escolhendo-se uma semente s_0 e executando o gerador BBS com esta semente. O número de bits gerados é exatamente o número de bits do texto claro a ser cifrado, por exemplo ℓ. O texto cifrado é gerado calculando-se a operação de ou-exclusivo entre o texto claro e o resultado do gerador. É calculado também o valor $s_{\ell+1} = s_0^{2^{\ell+1}}$, que também faz parte do texto cifrado. Já a decifragem é feita calculando-se s_0 a partir de $s_{\ell+1}$ e gerando novamente a seqüência pseudo-aleatória, o que permite recuperar o texto claro a partir do texto cifrado calculando-se novamente a operação de ou-exclusivo entre ambas.

Como já se sabia que o algoritmo BBS era seguro, a grande contribuição do esquema Blum-Goldwasser foi a demonstração de como se obter s_0 a partir de $s_{\ell+1}$. Ela é baseada no fato de que, se $p \equiv 3 \mod 4$, então $x^{(p+1)/4} \mod p$ é a raiz quadrada principal de x módulo p. Conseqüentemente,

$$x^{(p+1)/4^{(\ell+1)}} \mod p$$

é a $2^{\ell+1}$-ésima raiz de x módulo p. Desta forma, pode-se calcular a mesma raiz com respeito ao módulo q e utilizar o teorema chinês do resto para obter o valor s_0 relativamente a n.

2.4.3 Segurança semântica

No próprio artigo de introdução ao conceito de criptografia probabilística, Goldwasser e Micali apresentam pela primeira vez o termo *segurança semântica*. Considere as duas seguintes noções de segurança:

Definição 2.20 *Seja $P(x)$ um polinômio. Um algoritmo de chave pública é dito **polinomialmente seguro** se, dado duas mensagens m_1 e m_2 e o texto cifrado C correspondente a uma delas, um adversário polinomial não consegue decidir se $C = Enc(m_1)$ ou $C = Enc(m_2)$ com uma vantagem igual a $\frac{1}{2} + \frac{1}{P(x)}$.*

Definição 2.21 *Um algoritmo de chave pública é dito **semanticamente seguro** se, para todas as distribuições de texto claro, qualquer coisa que um adversário polinomial possa computar sobre o texto claro a partir do texto cifrado também pode ser computado sem o texto cifrado.*

Goldwasser e Micali provam no próprio artigo que um algoritmo polinomialmente seguro necessariamente é semanticamente seguro. Além disso, em (MICALI; RACKO; SLOAN, 1988) é provada a implicação inversa, ou seja, que segurança semântica implica em segurança polinomial. Portanto, ambas as noções são equivalentes. Esta implicação sugere um *framework* para a prova de segurança semântica, bastando-se, então, provar que duas instâncias quaisquer de cifragem são indistinguíveis entre si por um adversário polinomial.

3 CURVAS ELÍPTICAS E CRIPTOGRAFIA

A teoria de curvas elípticas é a base fundamental para a concepção do algoritmo semanticamente seguro proposto neste trabalho. Grande parte do algoritmo se baseia em idéias empregadas por outros protocolos criptográficos, que também utilizam curvas elípticas como base. Entretanto, na maioria dos algoritmos criptográficos, o principal papel das curvas elípticas é o de fornecer uma estrutura de grupo, a qual é transparentemente utilizada em algoritmos desenvolvidos especialmente para grupos. No caso do algoritmo proposto, isto não acontece. Diversas características específicas da teoria de curvas elípticas, como a teoria de multiplicação complexa e o *twist* de uma curva mostram-se fundamentais.

Este capítulo apresenta uma introdução à teoria de curvas elípticas e todas as características desta teoria que são exploradas no algoritmo principal. Alguns conceitos mais formais a respeito de teoria dos números são apresentados. De fato, alguns destes conceitos, como "grupo multiplicativo" e "logaritmo discreto" já foram apresentados informalmente no capítulo anterior. Entretanto, opta-se por reapresentá-los mais formalmente, de forma que nenhuma peculiaridade destes conceitos escapem ao entendimento do leitor. Após esta introdução inicial acerca de teoria dos números, é apresentada a teoria de curvas elípticas propriamente dita e, mais especificamente, o problema difícil que elas fornecem. Em seguida, alguns exemplos de protocolos criptográficos que se utilizam destes problemas, além de uma seção com algumas características adicionais a respeito de curvas elípticas sobre corpos finitos. O leitor que é familiarizado com a teoria de curvas elípticas pode continuar a leitura da dissertação a partir do próximo capítulo.

3.1 Fundamentos matemáticos

Esta seção tem como objetivo a apresentação de diversos conceitos matemáticos que serão utilizados como base para os demais conceitos acerca de curvas elípticas que aparecem no decorrer desta dissertação. Todas as definições apresentadas aqui são utilizadas de uma forma implícita ou explícita em alguma outra seção ou capítulo, evitando-se os conceitos que fazem parte da teoria, mas não são utilizados. Alguns livros com uma introdução semelhante, que incluem alguma ênfase na teoria da criptografia são (YAN, 2000) e (KOBLITZ et al., 1998).

3.1.1 Corpos

Apesar de não ser conhecido por este nome, corpo é um dos conceitos matemáticos mais utilizados em todas as instâncias da matemática, principalmente na aritmética mais simples ensinada nas escolas. Um corpo é um conceito fundamental a partir do qual, por

exemplo, as características das equações cartesianas são definidas.

Definição 3.1 *Um **corpo**, denotado $\langle \mathbb{K}, \oplus, \odot \rangle$ ou simplesmente $\mathbb{K}$, é um conjunto com pelo menos dois elementos mais duas operações binárias $\oplus$ e $\odot$, chamadas adição e multiplicação respectivamente, definidas sobre $\mathbb{K}$, onde os seguintes axiomas são verdadeiros:*

- *O conjunto é fechado sobre ambas as operações $\oplus$ e $\odot$:*

$$a \oplus b \in \mathbb{K}, \qquad \forall a, b \in \mathbb{K} \text{ e,}$$

$$a \odot b \in \mathbb{K}, \qquad \forall a, b \in \mathbb{K}$$

- *Ambas as operações $\oplus$ e $\odot$ são associativas:*

$$a \oplus (b \oplus c) = (a \oplus b) \oplus c, \qquad \forall a, b, c \in \mathbb{K} \text{ e,}$$

$$a \odot (b \odot c) = (a \odot b) \odot c, \qquad \forall a, b, c \in \mathbb{K}$$

- *Ambas as operações $\oplus$ e $\odot$ são comutativas:*

$$a \oplus b = b \oplus a, \qquad \forall a, b \in \mathbb{K} \text{ e,}$$

$$a \odot b = b \odot a, \qquad \forall a, b \in \mathbb{K}$$

- *Existe um elemento especial $0 \in \mathbb{K}$ chamado zero, que é a identidade aditiva de $\mathbb{K}$, isto é:*

$$a \oplus 0 = a, \qquad \forall a \in \mathbb{K}$$

- *Existe um elemento especial $1 \in \mathbb{K}$, diferente de zero, que é a identidade multiplicativa de $\mathbb{K}$, isto é:*

$$a \odot 1 = a, \qquad \forall a \in \mathbb{K}$$

- *Para todo elemento $a \in \mathbb{K}$, existe um elemento especial $-a \in \mathbb{K}$ chamado inverso aditivo de a, tal que:*

$$a \oplus (-a) = 0, \qquad \forall a \in \mathbb{K}$$

- *Para todo elemento $a \neq 0$, existe um elemento especial $a^{-1} \in \mathbb{K}$ chamado inverso multiplicativo de a, tal que:*

$$a \odot (a^{-1}) = 1, \qquad \forall a \in \mathbb{K}$$

- *A operação $\odot$ é distributiva sobre a operação $\oplus$:*

$$a \odot (b \oplus c) = a \odot b \oplus a \odot c, \qquad \forall a, b, c \in \mathbb{K}$$

Entre os exemplos mais comuns de corpos estão o dos números reais $\mathbb{R}$, o corpo dos números complexos $\mathbb{C}$ e o dos números racionais $\mathbb{Q}$, com as operações tradicionais de multiplicação e adição nestes conjuntos. O conjunto dos inteiros $\mathbb{Z}$, por sua vez, não forma um corpo, pois não existe inverso multiplicativo para todos os números. Este conjunto constitui o elemento matemático chamado um *anel*.

Definição 3.2 *Um **anel** é uma estrutura com as mesmas propriedades de um corpo, exceto que a existência de inverso multiplicativo e identidade multiplicativa são opcionais.*

Pela definição 3.2 fica claro que todo corpo também é um anel, mas não o contrário. Considerando-se o seu conjunto base, é possível dividir os corpos em duas classes fundamentais, dependendo do seu número de elementos.

Definição 3.3 *Um corpo cujo conjunto base tem um número finito de elementos é dito um **corpo finito**, caso contrário é dito um **corpo infinito**.*

Um corpo infinito geralmente é denotado pelo mesmo símbolo que é utilizado para o seu conjunto base. O corpo dos reais é denotado por $\mathbb{R}$, por exemplo. Já a notação de um corpo finito freqüentemente é feita utilizando-se sempre o mesmo símbolo, junto com a informação extra do número de elementos do conjunto do corpo. Para um corpo finito de n elementos, existem diversas notações, e entre as mais comuns estão $GF(n)$, Z/nZ e $\mathbb{F}_n$, sendo a última advinda da denominação em inglês de corpo, *field*. Nesta dissertação, será utilizada a notação $\mathbb{F}_n$.

Definição 3.4 *O número de elementos do conjunto de um corpo finito é chamado **ordem** do corpo.*

No que diz respeito à ordem dos corpos finitos, considere o seguinte teorema, de fundamental importância para os nossos propósitos:

Teorema 3.5 *Existe um corpo finito $\mathbb{F}_n$ com ordem n se e somente se n é primo ou uma potência de um primo.*

A afirmação, de fato, não é tão complicada de ser demonstrada. Considere um conjunto cuja ordem é um número composto por primos distintos, por exemplo $\mathbb{F}_n$ tal que $n = p \times q$. Este conjunto não pode formar um corpo pois, para os elementos que são múltiplos de p e q não existem inversos multiplicativos. E isto fere a definição de corpo.

Estendendo a análise da ordem dos corpos, um segundo resultado pode ser obetido, neste caso muito mais complicado de ser demonstrado (mais informações podem ser encontradas em (LIDL; NIEDERREITER, 1994)):

Teorema 3.6 *Um corpo finito com $n = p^f$ elementos, p primo e $f \geq 1$, é único a menos de isomorfismo.*

Em outras palavras, para quaisquer dois corpos com mesmo número de elementos existe um isomorfismo que leva os elementos de um corpo para os elementos de outro, tal que todas as propriedades das operações são preservadas. Este fato é extremamente útil do ponto de vista computacional, pois permite que se utilize qualquer representação para os corpos com mesma ordem. O teorema garante que as relações entre os elementos serão sempre mantidas, e portanto, quando for interessante, é possível se converter de um modelo de representação dos elementos do corpo para outro sem perdas.

Corpos finitos com um número primo de elementos normalmente são implementados utilizando-se aritmética modular, mesmo que isso não seja obrigatório. Neste caso, a adição e a multiplicação são implementadas módulo p, o inverso aditivo de um número é o seu complemento e, como o módulo é um número primo, cada número possui um inverso multiplicativo. Já corpos do tipo $\mathbb{F}_n$ com $n = p^f$, geralmente são implementados

considerando-se cada elemento um polinômio com f termos e com coeficientes pertencentes à $\mathbb{F}_p$. As operações são operações entre polinômios, sempre tomadas módulo um polinômio irredutível (ou *primo*) de grau f. Dado que todos os corpos com p^f elementos são isomorfos, conclui-se que qualquer polinômio irredutível de grau f pode ser utilizado.

Definição 3.7 *Chama-se **característica** de um corpo o número de vezes em que, adicionando-se a identidade multiplicativa a si mesma, obtém-se a identidade aditiva. Caso isso nunca aconteça, a característica do corpo é zero.*

Corpos como $\mathbb{R}$ e $\mathbb{Q}$, que são infinitos, são corpos onde qualquer somatório de identidades multiplicativas nunca resultam na identidade aditiva, e portanto têm característica zero. Corpos finitos, por outro lado, sempre têm característica diferente de zero. É fácil visualizar que corpos $\mathbb{F}_p$ com um número primo de elementos têm característica p. Corpos do tipo $\mathbb{F}_n$ com $n = p^f$ também têm característica p, pois a identidade é o polinômio constante 1, cuja adição consigo mesmo terá as mesmas propriedades da adição no corpo $\mathbb{F}_p$. Isso significa que a característica de um corpo é sempre prima, mesmo que a ordem do corpo não seja.

Em implementações de algoritmos que utilizam curvas elípticas, é comum se separar o tratamento de curvas sobre corpos com características dois e três ($\mathbb{F}_{2^f}$ e $\mathbb{F}_{3^f}$) dos corpos de característica maiores ou iguais a cinco, também chamados de **corpos de característica grande**. Isso é feito porque estes corpos são fundamentalmente diferentes. Além disso, curvas elípticas apresentam propriedades significativamente diferentes quando consideradas sobre cada um destes tipos de corpos. Todos os algoritmos a serem apresentados nesta dissertação serão exclusivos para corpos com característica grande, portanto, não se entrará em detalhes das características das curvas sobre $\mathbb{F}_{2^f}$ e $\mathbb{F}_{3^f}$.

3.1.2 Grupos

Outra entidade matemática muito utilizada para fins criptográficos é chamada *grupo*. Um grupo, é uma entidade mais simples que um corpo, e ainda assim, sua a teoria é tão ou mais densa que a teoria sobre corpos. A seguir é apresentada a definição de grupo, e em seguida, uma série de definições que são importantes para a compreensão de todas as idéias propostas.

Definição 3.8 *Um **grupo**, denotado $\langle \mathcal{G}, \star \rangle$ é um conjunto com pelo menos um elemento mais uma operação binária $\star$ definida sobre $\mathcal{G}$, onde os seguintes axiomas são verdadeiros:*

- *O conjunto é fechado sobre a operação $\star$:*

$$a \star b \in \mathcal{G}, \qquad \forall a, b \in \mathcal{G}$$

- *A operação $\star$ é associativa:*

$$a \star (b \star c) = (a \star b) \star c, \qquad \forall a, b, c \in \mathcal{G}$$

- *Existe um elemento especial $e \in \mathcal{G}$ que é a identidade da operação $\star$ em $\mathcal{G}$, isto é:*

$$a \star e = a, \qquad \forall a \in \mathcal{G}$$

- *Para todo elemento $a \in \mathcal{G}$, existe um elemento especial $-a \in \mathcal{G}$ chamado inverso aditivo de a, tal que:*

$$a \star (-a) = e, \qquad \forall a \in \mathcal{G}$$

Definição 3.9 *Quando a operação $\star$ for óbvia pelo contexto, o grupo $\langle \mathcal{G}, \star \rangle$ pode ser denotado simplesmente por $\mathcal{G}$*

Definição 3.10 *Caso a operação $\star$ no grupo $\langle \mathcal{G}, \star \rangle$ seja comutativa, ou seja*

$$a \star b = b \star a, \qquad \forall a, b \in \mathcal{G},$$

*então o grupo é chamado de **grupo abeliano**.*

Definição 3.11 *Caso a operação de um grupo seja representada pelo símbolo $+$, então a identidade do grupo é chamada zero, o inverso de um elemento a é denotado por $-a$ e o grupo é dito um **grupo aditivo**. Por outro lado, se a operação é representada pelo símbolo $\times$, então a identidade do grupo é chamada um, o inverso de um elemento a é a^{-1} e o grupo é dito um **grupo multiplicativo**.*

Assim como corpos, um grupo pode ser finito ou infinito, de acordo com o número de elementos do conjunto base.

Definição 3.12 *A **ordem** de um grupo $\mathcal{G}$, denotada $|\mathcal{G}|$ ou $\#\mathcal{G}$ é definida como o número de elementos do conjunto $\mathcal{G}$.*

Uma característica interessante de se notar é que, dado um corpo, cada uma das operações do corpo formam um grupo com o conjunto base. Por exemplo, sabendo-se que $\mathbb{Q}$ é um corpo, então o conjunto $\mathbb{Q}$ com a operação de adição forma um grupo $\langle \mathbb{Q}, + \rangle$, assim como o conjunto $\mathbb{Q} \backslash \{0\}$ com a operação de multiplicação[1] forma o grupo $\mathbb{Q}^*$. O mesmo vale para os corpos $\mathbb{R}$ e $\mathbb{C}$. Já o conjunto $\mathbb{Z}$, que não formava um corpo com multiplicação e adição, passa a formar um grupo junto com a operação de adição. Entretanto, com operação de multiplicação não, já que não existem inversos multiplicativos para a maioria dos números em $\mathbb{Z}$.

Seguindo a analogia, os corpos finitos geram equivalentemente grupos finitos. Por exemplo, o corpo $\mathbb{F}_p$ dá origem ao grupo multiplicativo $\mathbb{F}_p^*$ que é muito utilizado para fins criptográficos, como visto nos capítulos anteriores. Parte-se agora para a definição de algumas propriedades bastante importantes dos grupos, as quais serão utilizadas no algoritmo principal.

Definição 3.13 *Considere o grupo $\langle \mathcal{G}, \star \rangle$. Um subconjunto $\mathcal{G}' \subseteq \mathcal{G}$ que também é um grupo com a operação $\star$ é denominado **subgrupo** de $\mathcal{G}$. Caso se tenha $\mathcal{G}' \subset \mathcal{G}$, então o subgrupo é dito **próprio**.*

Por exemplo, um grupo formado somente pela identidade é um subgrupo de qualquer grupo. Este subgrupo é considerado um subgrupo *trivial*. Já um subgrupo não trivial é um subgrupo próprio com ordem maior que um.

[1] Para um corpo formar um grupo com a operação de multiplicação, é necessário retirar o elemento identidade da adição, pois ele não possui inverso multiplicativo.

Definição 3.14 *Considere um grupo aditivo $\mathcal{G}$. Define-se a composição sucessiva de um elemento $g \in \mathcal{G}$ consigo mesmo n vezes como o* ***produto escalar*** *por n. Normalmente, produto escalar é indicado por justaposição do valor escalar n com o elemento g, ou seja,*

$$ng = \underbrace{g \star g \star g \ldots \star g}_{n \text{ vezes.}}.$$

A composição sucessiva em um grupo é justamente a aplicação da operação sobre um elemento uma série definida de vezes. Esta composição sucessiva é a operação que possui as propriedades que são exploradas pela criptografia.

Definição 3.15 *A composição sucessiva em um grupo multiplicativo é chamada de* ***exponenciação*** *e é denotada da mesma forma que a exponenciação tradicional, com o expoente em sobrescrito.*

Por exemplo, no grupo $\langle \mathcal{R}, + \rangle$, o produto escalar de um elemento a consigo mesmo 3 vezes é $a + a + a = 3a$. Note que $3a \in \mathcal{R}$ é um elemento do grupo, que é obtido operando-se a consigo mesmo 3 vezes. Já no caso do grupo $\langle \mathcal{R}, \times \rangle$, a exponenciação do elemento a por 3 seria denotada pelo elemento $a^3 = a \times a \times a$.

No âmbito da criptografia, a composição sucessiva de um elemento de um grupo é freqüentemente por um valor escalar de *grande magnitude*. É comum a composição com valores da ordem 2^{160}, e em alguns protocolos chegando a 2^{1024} ou ainda mais. Claramente, a simples composição da operação este número de vezes é absolutamente impraticável, por mais simples que a operação seja. O que se faz, na prática, é utilizar a representação binária do valor escalar de forma a acelerar este cálculo, obtendo um algoritmo polinomial no número de bits deste valor. Este método é conhecido como **duplicar e somar** (*double and add*) quando o grupo é multiplicativo e **elevar ao quadrado e multiplicar** (*square and multiply*) quando o grupo é multiplicativo.

As próximas definições serão fornecidas em função de grupos multiplicativos. É importante lembrar, entretanto, que a distinção entre grupos aditivos e multiplicativos é meramente notacional e, portanto, elas valem da mesma forma para qualquer grupo.

Definição 3.16 *Seja o grupo multiplicativo $\mathcal{G}$, e seja e a identidade do grupo. Então*

$$a^1 = a, \qquad \forall a \in \mathcal{G} \;\; e$$

$$a^0 = e, \qquad \forall a \in \mathcal{G}\,.$$

Quando o grupo é familiar, a definição acima é bastante natural. Multiplicar por zero em um grupo aditivo em $\mathbb{R}$ resulta em zero, a identidade. Entretanto, como a composição sucessiva é uma definição que vale para qualquer grupo, é importante tratar os casos zero e um separadamente. Do ponto de vista de grupos, as composições sucessivas pelos valores zero e um não fazem sentido, pois a operação de um grupo é binária e necessita de dois argumentos. Por isso a necessidade da definição destes casos particulares.

Com a composição sucessiva definida, pode-se então definir o logaritmo, que é basicamente a operação inversa.

Definição 3.17 *O* ***logaritmo*** *de um elemento $a \in \mathcal{G}$ na base $b \in \mathcal{G}$, denotado $\log_b a$, é o valor escalar que indica o número de vezes na qual b deve ser composto consigo mesmo de forma que o resultado seja a.*

Em outras palavras, $\log_b a = n$ implica que $b^n = a$ em um grupo multiplicativo (ou então $nb = a$ em um grupo aditivo). É comum na literatura chamar o logaritmo de um elemento de o seu *índice* naquela base. É importante ressaltar, porém, que ao contrário da aritmética tradicional, nem sempre o logaritmo é definido para qualquer elemento em qualquer base. Isto é, nem sempre existirá um valor escalar n tal que $b^n = a$ para quaisquer combinações b e a.

Uma outra definição importante é a de gerador de um grupo, dada a seguir.

Definição 3.18 *Considere um grupo $\mathcal{G}$ de ordem n. Caso exista um elemento $a \in \mathcal{G}$ tal que $\{a^i | 0 \leq i < n\} = \mathcal{G}$, então a é dito um* elemento gerador *de $\mathcal{G}$.*

Definição 3.19 *Um grupo é dito **cíclico** se ele possuir um elemento gerador. Caso contrário ele é dito **acíclico**.*

Elementos geradores de grupos são muito úteis, pois todos os elementos do grupo podem ser identificados através dos n diferentes índices do gerador. Algoritmos importantes, como a extração de raiz quadrada em um corpo finito de característica prima, são eficientes devido a existência de elementos geradores no grupo multiplicativo gerado pelo corpo.

Até agora, todas as definições foram para grupos genéricos. Do ponto de vista criptográfico, os grupos que são utilizados na prática são sempre grupos finitos. Portanto, as próximas definições enfocarão principalmente as propriedades de grupos finitos. Isso não significa, entretanto, que elas necessariamente não sejam aplicáveis a grupos infinitos.

Considerando-se a nomenclatura utilizada para grupos finitos, quando as operações são tomadas em módulo, é comum se referir à exponenciação por *exponenciação modular*. Da mesma forma, para grupos finitos em geral, o logaritmo de um elemento é chamado de *logaritmo discreto*.

Definição 3.20 *Define-se a **ordem de um elemento** a em um grupo finito $\mathcal{G}$ como o menor valor escalar n tal que*

$$a^n = e.$$

É importante não confundir a ordem de um elemento com a ordem de um grupo. Muito freqüentemente ambas as definições aparecem misturadas, entretanto, é necessário lembrar que elas significam idéias completamente diferentes.

3.1.3 Geradores de um grupo

Um problema bastante complexo, e sem solução universal, é o problema de se encontrar, rapidamente, elementos geradores de um grupo. Rapidamente é uma restrição importante, pois pelo menos um método ineficiente é óbvio, que é o da listagem exaustiva das potências de todos os elementos. No algoritmo proposto esse problema aparece, e a solução adotada para o mesmo é a restrição do grupo de forma que o problema de torne simples.

A solução adotada é conseqüência direta do teorema que será provado a seguir. Inicialmente, considere os seguintes lemas, cuja prova é omitida. A partir destes lemas, será apresentada a prova do teorema que é utilizado nos algoritmos dos próximos capítulos.

Lema 3.21 *Seja a um elemento do grupo $\mathcal{G}$ com ordem r. Então o conjunto $\{a^i | 0 \leq i < r\}$ é um subgrupo de $\mathcal{G}$.*

Se todas as potências de um elemento, até a sua ordem, formam um grupo, então estes elementos por si só obedecem a todos os axiomas que definem um grupo. Em outras palavras, para cada elemento gerado por um índice de a, existe um outro índice de a que identifica outro elemento que é o seu inverso. Além disso, para a operação entre quaisquer dois elementos gerados por um índice de a existe um outro índice de a que indica o seu resultado (isto acontece porque o subgrupo também é um grupo, e portanto é fechado). Este resultado é amplamente conhecido e foi provado pelo matemático *Joseph Louis Lagrange*.

Um outro lema necessário é o que relaciona a ordem de grupos com seus subgrupos, também provado por Lagrange.

Lema 3.22 *Seja $\mathcal{G}$ um grupo finito com ordem n. Então a ordem de qualquer subgrupo $\mathcal{G}' \subseteq \mathcal{G}$ divide n.*

Combinando-se este lema e o anterior, pode-se concluir que a ordem de qualquer elemento de um grupo divide a ordem do próprio grupo. Ou seja, a ordem de qualquer elemento é alguma combinação dos fatores da ordem do grupo. Por exemplo, se a ordem de um grupo n decompõe-se nos valores primos p e q, então a ordem de qualquer elemento é uma das seguintes: 1, p, q ou pq. Se a ordem do grupo se decompõe em p, q e t, então as possíveis ordens dos elementos são: 1, p, q, t, pq, qt, pq, pqt. Da mesma forma, se existe um elemento com ordem p, então a ordem do grupo é, necessariamente divisível por p, como cp para algum valor c.

Note que os elementos com ordem igual à ordem do grupo, quando existem, são justamente os elementos geradores do grupo (definindo então o grupo como cíclico). Continuando-se o raciocínio, o seguinte teorema pode ser facilmente provado.

Teorema 3.23 *Seja $\mathcal{G}$ um grupo finito com ordem prima p. Então todos os elementos de $\mathcal{G}$ diferentes da identidade são geradores de $\mathcal{G}$.*

Prova De acordo com o lema 3.21 a ordem de qualquer elemento forma um subgrupo, e de acordo com o lema 3.22, a ordem de qualquer subgrupo divide a ordem do grupo, portanto a ordem de qualquer elemento divide a ordem do grupo. Como a ordem do grupo $\mathcal{G}$ é prima, os elementos de $\mathcal{G}$ só podem ter ordem 1 ou p. Por definição o único elemento com ordem 1 é a identidade e, pois é o único elemento $a \in \mathcal{G}$ tal que a seguinte relação é verdadeira: $a^1 = a = e$. Assim, todos os elementos diferentes da identidade têm ordem p, e, portanto, todos são geradores de $\mathcal{G}$. □

Com este teorema, pode-se concluir que uma possível solução para o problema de se descobrir elementos geradores é trabalhar com grupos que tenham ordem prima. Assim, descobrir um gerador reduz-se ao problema de se descobrir um elemento diferente da identidade.

É importante notar, entretanto, que encontrar grupos com ordem prima não é uma tarefa trivial por si só. Por exemplo, grupos multiplicativos F_p^* nunca têm ordem prima. Isto acontece pois estes grupos possuem sempre $p-1$ elementos (os valores menores que p e maiores que zero) e, portanto, a ordem é sempre um número par. Nestes casos uma alternativa é tentar restringir o grupo para uma ordem quase-prima, como por exemplo um valor $p = 2q + 1$ onde q também é primo, ou utilizar estratégias probabilísticas para encontrar um gerador.

3.2 Curvas elípticas

Esta seção se dedicará a introduzir a teoria de curvas elípticas no âmbito da criptografia. A teoria sobre curvas elípticas, entretanto, é vastíssima e, portanto, não é possível abordar todas as questões envolvidas. Por exemplo, não será abordada em detalhes a rica teoria de curvas elípticas sobre $\mathbb{Q}$ (que entretanto é abordada de forma bastante profunda em (SILVERMAN; TATE, 1992)) da qual em última instância surge a teoria sobre corpos finitos. Desta forma, será apresentada uma introdução didática ao assunto, enfatizando as características que mais interessam para os algoritmos propostos. Uma apresentação mais completa do uso criptográfico das curvas elípticas é dada em (BLAKE; SEROUSSI; SMART, 1999).

Definição 3.24 *Seja $\mathbb{K}$ um corpo. Uma* curva elíptica *sobre $\mathbb{K}$, denotada por $\mathcal{E}(\mathbb{K})$, é o conjunto de soluções $\mathbb{K}^2$ de uma equação cartesiana de forma geral*

$$y^2 + axy + by = x^3 + cx^2 + dx + e \tag{3.1}$$

com $a, b, c, d, e \in \mathbb{K}$ e $(x, y) \in \mathbb{K}^2$. Une-se ao conjunto $\mathcal{E}(\mathbb{K})$ um ponto extra chamado ponto no infinito, e denotado por ∞.

As de curvas equações elípticas não são elipses, mas são equações advindas de integrais para o cálculo de arcos de elipse, e daí o nome. Além do mais, uma curva elíptica é somente uma curva quando considerada sobre um corpo de domínio contínuo, como $\mathbb{R}$. De qualquer forma, o que se considera é o conjunto discreto de pontos que satisfazem a equação da curva, que pode ser finito ou infinito.

Seja um corpo $\mathbb{K}$ com característica diferente de 2 e 3. Para qualquer curva elíptica na forma da equação 3.1 definida sobre $\mathbb{K}$ existe uma outra curva isomórfica a ela, mas com a seguinte forma simplificada:

$$y^2 = x^3 + ax + b \tag{3.2}$$

Como a curva isomórfica simplificada sempre existe, é comum se considerar que todas as curvas sobre corpos com característica diferente de 2 e 3 possuem esta forma. De fato, existem outros isomorfismos para os corpos com característica 2 e 3 que simplificam a equação 3.1, porém, a equação pode possuir uma forma diferente 3.2. Não é possível simplificar da mesma forma a curvas sobre estes corpos, pois este isomorfismo particular, que é representado por uma mudança de coordenadas, exige divisões por 2 e 3 (na realidade 2 e 12), que não são definidas para estes corpos.

Apesar de todas as equações na forma da equação 3.2 serem curvas elípticas, para propósitos criptográficos nem todas as curvas podem ser utilizadas. Determinar se uma curva pode ou não ser utilizada depende do discriminante da curva elíptica em questão.

Definição 3.25 *Seja $\mathcal{E}(\mathbb{K})$ uma curva elíptica definida sobre o corpo $\mathbb{K}$ com característica diferente de 2 e 3, na forma da equação 3.2. Então a expressão*

$$\Delta = -16(4a^3 + 27b^2)$$

*é o **discriminante** da curva $\mathcal{E}(\mathbb{K})$.*

É necessário se impor sobre as curvas utilizadas que seja possível traçar a reta tangente à curva em qualquer ponto da mesma. Mesmo que a definição geométrica de tangente

não faça muito sentido sobre corpos finitos, a sua definição algébrica continua fazendo, e é necessário que este valor esteja definido para todos os pontos de $\mathcal{E}$. Esta restrição é satisfeita sempre que a equação cúbica do lado direito da equação 3.2 não possuir raízes múltiplas em $\mathbb{K}$, ou seja, o discriminante $-16(4a^3 + 27b^2)$ não deve ser zero e nem um múltiplo da característica do corpo $\mathbb{K}$. O caso mais óbvio é quando a e b são ambos zeros.

Dada uma curva elíptica que atenda esta restrição, é possível se definir uma operação fechada entre os pontos da curva tal que o conjunto de pontos, junto com o ponto no infinito, formem um grupo com esta operação. A operação é chamada de **adição de pontos** e, portanto, forma um grupo aditivo.

A forma mais simples de apresentar esta operação é através da sua concepção geométrica. Entretanto, ela só tem o sentido esperado para curvas definidas sobre $\mathbb{R}$. Inicialmente ela será apresentada utilizando-se a concepção geométrica, e após, utilizando-se a concepção algébrica, que é aplicável a qualquer corpo $\mathbb{K}$.

Note que o que se está definindo é a operação de um grupo. É importante, portanto, definir dois elementos especiais que são a *identidade* do grupo e o *inverso* de um elemento. Como já foi explicado, os elementos do grupo são os pontos da curva elíptica junto com o ponto no infinito. Define-se, então, a identidade como o ponto no infinito ∞.

De forma a se definir o inverso de um elemento, inicialmente considere a equação 3.2. Uma característica importante que pode ser observada analisando-se esta equação, e que não é muito óbvia na equação 3.1, é a sua simetria. Para qualquer valor da coordenada $x \neq 0$ que possua uma coordenada y correspondente, então existe um outro valor que também satisfaz a equação para o mesmo x, que é $-y$. Note que o significado da expressão $-y$ depende do corpo específico sobre o qual a curva esta definida. O valor $-y$ é definido como o inverso do valor $y \in \mathbb{K}$ com relação a operação de adição em $\mathbb{K}$. Pela definição de corpo, este inverso sempre existe, caso contrário $\mathbb{K}$ não seria um corpo.

Conclui-se, então, que é possível definir-se o inverso de um elemento como o seu ponto simétrico com relação à coordenada x. Isto é, o inverso do ponto $P = (x, y) \in \mathcal{E}(\mathbb{K})$ é o ponto $-P = (x, -y) \in \mathcal{E}(\mathbb{K})$. Naturalmente, o inverso da identidade será a identidade, e um ponto com coordenada $y = 0$ será o seu próprio inverso.

Desta forma, é possível se definir a operação de adição de pontos. Esta operação é freqüentemente chamada de *método das tangentes e secantes*, de acordo com as operações geométricas que utiliza.

Definição 3.26 *Seja $\mathcal{E}(\mathbb{R})$ uma curva elíptica definida sobre o corpo $\mathbb{R}$. A **adição de pontos** nesta curva é definida da seguinte forma. Sejam P_1 e P_2 pontos quaisquer da curva $\mathcal{E}$. Para se definir o resultado da operação $P_1 + P_2$, existem quatro casos que precisam ser analisados.*

1. *A soma de um ponto P_1 qualquer com o ponto ∞ resulta no próprio ponto P_1, isto é*
$$P_1 + \infty = P_1.$$

2. *Caso se tenha que $P_2 = -P_1$, então a soma*
$$P_1 + P_2 = P_1 + (-P_1) = \infty.$$

3. *A soma de dois pontos P_1 e P_2 distintos, diferentes de ∞, é definida da seguinte forma. Traça-se uma reta secante que passa por ambos os pontos P_1 e P_2. Como*

$P_1 \neq -P_2$ (caso anterior), necessariamente a reta interceptará a curva em um terceiro ponto P_3. A soma $P_1 + P_2$ é definida como o ponto inverso de P_3, ou seja,

$$P_1 + P_2 = -P_3.$$

4. *Caso se tenha $P_1 = P_2$ então, se está somando um ponto consigo mesmo. Neste caso, traça-se uma reta tangente à curva no ponto P_1. Esta reta tangente pode ser vertical ou não. Caso ela seja vertical, então o resultado é ∞. Se ela não for vertical, então esta reta interceptará a curva em um outro ponto P_3. Assim como no caso anterior, a soma é definida como o inverso de P_3, ou seja,*

$$P_1 + P_1 = -P_3.$$

Observando a definição acima, fica fácil entender a necessidade de que seja possível calcular a reta tangente à curva em todos os pontos. Para os pontos onde isso não fosse possível, a soma dele consigo mesmo seria indefinida e a operação não formaria um grupo. É interessante notar, também, que a definição do inverso de um ponto surge naturalmente da reta que cruza dois pontos simétricos com relação ao eixo x, que são justamente os casos em que a secante não intercepta a curva em um terceiro ponto. Como dois elementos inversos devem resultar na identidade, então esta escolha está de acordo com a definição.

As regras como definidas são estritamente geométricas, e dependem de secantes e tangentes a uma curva. Estas regras, entretanto, podem ser traduzidas em fórmulas algébricas, de forma que a operação seja aplicável à curvas definidas sobre qualquer corpo $\mathbb{K}$ onde a tangente à um ponto, por exemplo não necessariamente faz sentido. Assim, pode-se redefinir a operação da seguinte forma.

Definição 3.27 *Seja $\mathcal{E}(\mathbb{K})$ uma curva elíptica definida sobre o corpo $\mathbb{K}$. Sejam $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ pontos quaisquer da curva $\mathcal{E}$. O resultado da operação de* ***adição de pontos*** *entre P_1 e P_2 reduz-se aos seguintes casos.*

1. *Se um dos pontos é ∞, então o resultado é o outro ponto.*
2. *Se $x_1 = x_2$ e $y_1 = -y_2$, então o resultado é ∞.*
3. *Se $x_1 \neq x_2$ ou $y_1 \neq -y_2$, então o ponto $P_3 = (x_3, y_3)$ é calculado da seguinte forma. A secante que passa através de P_1 e P_2 obedece à equação*

$$y = \lambda x + \mu$$

onde o coeficiente angular λ é

$$\lambda = \frac{y_1 - y_2}{x_1 - x_2}$$

e a reta intercepta o eixo y em μ dado por

$$\mu = \frac{x_1 y_2 - x_2 y_1}{x_1 - x_2}.$$

A secante intercepta a curva $y^2 = x^3 + ax + b$ nos pontos com coordenada x que satisfazem

$$(\lambda x + \mu)^2 = x^3 + ax + b,$$

$$x^3 - \lambda^2 x^2 + (a - 2\lambda\mu)x + (b - \mu^2) = 0.$$

Sabe-se que x_1 e x_2 são duas das três raízes deste polinômio cúbico. Utilizando uma análise algébrica não muito complexa, pode-se determinar que a terceira raiz, e portanto a coordenada x_3, obedece à expressão

$$x_3 = \lambda^2 - x_1 - x_2 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)^2 - x_1 - x_2.$$

A coordenada y_3 é o valor negativo da respectiva coordenada y de x_3 na reta $y = \lambda x_3 + \mu$,e portanto,

$$y_3 = -(\lambda x_3 + \mu).$$

4. *O caso restante, onde $x_1 = x_2$ e $y_1 = y_2$ se divide em dois subcasos. Se $y_1 = 0$, então o resultado é ∞. Tendo-se $y_1 \neq 0$ então a reta tangente à curva satisfaz a equação*

$$y = \lambda x + \mu$$

com o coeficiente angular dado por

$$\lambda = \frac{3x_1^2 + a}{2y_1}.$$

A reta intercepta o eixo y em

$$\mu = y_1 - \frac{3x_1^3 + ax_1}{2y_1}$$

e as coordenadas do ponto resultado são

$$x_3 = \lambda^2 - 2x_1 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1$$

e, mais uma vez,

$$y_3 = -(\lambda x_3 + \mu)$$

É importante ressaltar que o fato de se estar utilizando a equação $y^2 = x^3 + ax + b$ para a curva elíptica faz com que estas equações algébricas sejam aplicáveis somente à curvas sobre corpos com características diferentes de 2 e 3. Nestes corpos, as equações ficam um pouco modificadas, mas não significativamente.

3.2.1 Logaritmos no grupo de pontos de uma curva elíptica

O bloco básico de construção de criptossistemas baseados em curvas elípticas é a operação de *produto escalar* no grupo de pontos de uma curva elíptica. Ou seja, os cálculos da forma

$$Q = kP = \underbrace{P + P + P + \ldots + P}_{k \text{ vezes}} \tag{3.3}$$

Dados os pontos Q e P, recuperar o valor escalar k é o mesmo que calcular o logaritmo discreto de Q na base P no grupo de pontos da curva. É comum referir-se á este problema pela sua sigla em inglês ECDLP, que significa *Elliptic Curve Discrete Logarithm Problem*. Esta nomenclatura será utilizada em toda dissertação.

Um dos grandes fatores de interesse é que no grupo formado pelos pontos de uma curva elíptica adequadamente escolhida, esta operação não possui um algoritmo eficiente para sua computação. Isso faz com que a operação de produto escalar seja considerada *função unidirecional*, nos mesmos termos do capítulo 2, com as mesmas propriedades da operação de exponenciação modular (excetuando-se o fato de que agora os elementos são pontos, e não números).

É importante ressaltar que o ECDLP não é só um problema difícil, mas é um dos problemas mais difíceis utilizados para fins criptográficos nos dias de hoje. Diferentemente do logaritmo discreto em um grupo multiplicativo $\mathbb{F}_p^*$, que possui um algoritmo subexponencial chamado *cálculo de índices*, os únicos algoritmos para logaritmos sobre o grupo de uma curva elíptica bem escolhida são os que funcionam genericamente sobre qualquer grupo, como o algoritmo *Pohlig-Hellman* e o algoritmo de *Shanks* (MENEZES; OORSCHOT; VANSTONE, 1996). Hoje, sabe-se que algoritmos para cálculo de logaritmos em grupos genéricos, isto é, que não utilizam as propriedades subjacentes do grupo, necessariamente possuem uma complexidade exponencial (SHOUP, 1997). Como não se sabe como utilizar as características do conjunto de pontos da curva para solucionar este problema, somente algoritmos genéricos são aplicáveis ao ECDLP, e portanto, só são conhecidas soluções essencialmente exponenciais.

Naturalmente, essa dificuldade extra que o ECDLP apresenta faz com que este tipo de grupo seja extremamente atrativo para fins criptográficos. Como o problema é mais difícil, os corpos sobre os quais o problema é insolúvel (na prática) são muito menores do que os corpos onde os outros problemas, como fatoração e logaritmo discreto em $\mathbb{F}_p^*$, são insolúveis.

3.3 Protocolos criptográficos baseados no ECDLP

Como já mencionando, a grande maioria dos algoritmos baseados em curvas elípticas são algoritmos que funcionam genericamente para qualquer grupo onde o cálculo do logaritmo seja difícil. Mesmo antes da proposição do uso de curvas elípticas para fins criptográficos, diversos algoritmos nesta forma já existiam, a grande maioria baseados no grupo multiplicativo $\mathbb{F}_p^*$.

Assim, os algoritmos baseados no logaritmo discreto convencional em $\mathbb{F}_p^*$ são automaticamente transpostos para o modelo elíptico. A principal diferença é que, ao invés de se lidar com números inteiros, lida-se com os pontos de uma curva elíptica, e a operação do grupo deixa de ser exponenciação em módulo e passa a ser o produto escalar no grupo de pontos da curva. Alguns destes protocolos, como ECDH, ECAES e o ECDSA, foram padronizados por instituições como IETF, ANSI e o IEEE.

3.3.1 ECDH - Elliptic Curve Diffie-Hellman

O protocolo Diffie-Hellman é um protocolo definido para troca de uma chave secreta através de um canal público e é fundamentado no famoso problema homônimo, que se conjectura ser equivalente ao logaritmo discreto no grupo multiplicativo $\mathbb{F}_p^*$. A chave criada pelo protocolo é utilizada por um sistema de criptografia simétrica, como o 3DES ou o AES para transmissão de alguma mensagem fim a fim.

O Diffie-Hellman tradicional funciona da seguinte forma: Alice e Bob desejam trocar uma chave secreta através de um canal público. Inicialmente eles definem uma *chave pública* $g \in \mathbb{F}_p^*$. Em seguida cada um deles escolhe um número inteiro randômico, x_A e x_B pertencentes ao mesmo corpo, e os mantêm em segredo. Em posse de g e do seu

número secreto, Alice calcula o número $y_A = g^{x_A} \mod p$ e Bob calcula $y_B = g^{x_B} \mod p$. Feito isso, Alice e Bob enviam seus valores y_A e y_B um para o outro através do canal público. Agora Alice calcula a chave secreta com a forma $c = y_B^{x_A} \mod p$ e Bob com $c = y_A^{x_B} \mod p$. Como resultado, ambos possuem o número $c = g^{x_A x_B}$, que não é dedutível das informações que forma transmitidas em público, y_A, y_B, g e p porque o logaritmo discreto é difícil.

O ECDH, que é a contrapartida elíptica deste sistema, é deduzido imediatamente. Alice e Bob definem um corpo $\mathbb{F}_p$ e uma curva elíptica sobre esse corpo. Definem também um ponto gerador G sobre a curva (ou pelo menos com ordem divisível por um valor primo grande). Em seguida, escolhem os inteiros secretos x_A e x_B da mesma forma que anteriormente. Cada um deles calcula o produto escalar $Q_A = x_A G$ e $Q_B = x_B G$ e enviam os pontos resultado um ao outro. Calculam, então, o ponto secreto $P = x_A Q_B$ e $P = x_B Q_A$. O ponto P pode ser utilizado, de alguma forma, como base para a chave de um algoritmo simétrico, pois não é dedutível diretamente dos pontos Q_A, Q_B, G e da curva definida porque o ECDLP é difícil.

3.3.2 ECAES - Elliptic Curve AES

O ECAES é um protocolo baseado em um esquema de chave pública chamado ElGamal (MENEZES; OORSCHOT; VANSTONE, 1996). O algoritmo ElGamal é utilizado quando uma pessoa decide enviar uma mensagem cifrada para outra. Ele também é baseado no problema Diffie-Hellman. Imaginando que Alice deseja enviar uma mensagem m cifrada à Bob tal que somente Bob consiga lê-la, eles procedem da seguinte forma.

Inicialmente Bob calcula as suas chaves pública e privada. A chave privada é um inteiro x qualquer pertencente a um corpo $\mathbb{F}_p^*$. Ele então escolhe um outro número qualquer $g \in \mathbb{F}_p^*$ e calcula $y = g^x \mod p$. Assim, y, g e $\mathbb{F}_p^*$ são sua chave pública e são enviados para Alice por qualquer método, sendo a chave secreta x protegida pelo logaritmo discreto.

Alice, de posse da chave pública de Bob, escolhe um número qualquer $k \in \mathbb{F}_p^*$, e calcula $n = g^k \mod p$. Calcula também uma máscara $d = y^k \mod p$ e finalmente cifra a mensagem calculando $c = d \times m \mod p$. Alice envia o par $\langle c, n \rangle$ para Bob e elimina k e d. É importante notar que, em particular, m necessariamente deve ser menor que p, ou então este mascaramento resulta em perda de informação. Via de regra, m será uma chave secreta utilizada em um algoritmo de chave simétrica (como o AES, por exemplo). Entretanto, caso $m < p$, então o algoritmo auxiliar não é necessário.

Ao obter c e n Bob então passa a decifrar a mensagem. Para tanto, ele utiliza sua chave privada e recalcula a máscara $d = n^x \mod p$. Com a máscara, ele calcula o inverso multiplicativo da mesma no corpo $\mathbb{F}_p$, e obtém a mensagem original $m = cd^{-1} \bmod p$. Se m era uma chave secreta, então ele a utiliza para obter o texto claro original

A tradução desse sistema para o sistema elíptico, o ECAES, é feita mais ou menos da mesma forma que para o protocolo EC Diffie-Hellman. Primeiro, devem ser definidos a curva, o corpo $\mathbb{F}_p$ e um ponto inicial G qualquer na curva. A chave privada é um $x \in \mathbb{F}_p$ qualquer e a chave pública é o produto escalar $Y = xG$. Note que Y, a chave pública, é um ponto e x, a chave privada, é um número.

A idéia, neste caso, é criar e cifrar um ponto C que seja utilizado como chave para um algoritmo simétrico, no caso o AES. Neste caso, como o valor sendo transmitido é um ponto, necessariamente é utilizado um algoritmo simétrico. Optou-se por convencionar o AES para isto. Assim, escolhe-se um número descartável aleatório k e calculam-se os produtos escalares $N = kG$ e $C = kY$. A partir do ponto C, deriva-se uma chave

simétrica e a mensagem, $M = E_C(m)$ é cifrada com ela. Em seguida, M e N são enviados ao destinatário.

Para decifrar a mensagem M, é necessário recalcular C. Isso é trivial, já que $C = xN = xkG = kxG = kY$. Com o ponto C é possível recuperar a chave simétrica e calcular $m = D_C(M)$.

3.3.3 Elliptic Curve ElGamal sem função simétrica auxiliar

O protocolo ECAES, em resumo, é utilizado para a transmissão de uma chave AES codificada como um ponto aleatório de uma curva elíptica. Existe, entretanto, uma versão do algoritmo que permite que se criptografe diretamente uma mensagem qualquer m.

O principal problema envolvido é a necessidade de se codificar m como um ponto na curva. Neste caso, presume-se que exista uma função bijetora $f : \mathbb{F}_p \rightarrow \mathbb{F}_p \times \mathbb{F}_p$ tal que o resultado de f seja um ponto na curva para qualquer m. Por ser bijetora, deve ser possível calcular $f^{-1} : \mathbb{F}_p \times \mathbb{F}_p \rightarrow \mathbb{F}_p$ para se obter novamente m a partir do ponto. Para todos os fins, $M = f(m)$ é um ponto sobre a curva elíptica definida, enquanto $m = f^{-1}(M)$ é a operação inversa equivalente.

Mais uma vez, $x \in \mathbb{F}_p$ é a chave privada e $Y = xG$ como chave pública. O algoritmo funciona da seguinte forma. Primeiro um número k aleatório é escolhido. Em seguida, três pontos são calculados. $P_1 = kG$, $P_2 = kY$ e finalmente $P_3 = M + P_2$. Note que P_3 é obtido da soma normal de pontos sobre uma curva. Os pontos P_1 e P_3 são enviados ao destinatário.

Para obter M, aplica-se o seguinte procedimento. Primeiro obtém-se P_2 calculando $P_2 = xP_1$. A partir daí é só calcular $M = P_3 - P_2$. Note que a operação de subtração é apenas a soma de P_3 com o inverso do ponto P_2, que é obtido a partir da cordenada x de P_2 e do valor negativo da coordenada y no corpo $\mathbb{F}_p$. A operação $m = f^{-1}(M)$ recupera mensagem original.

3.3.4 ECDSA - Elliptic Curve Digital Signature Algorithm

Também é possível utilizar o ECDLP como base para assinaturas digitais. O ECDSA é uma versão baseada em curvas elípticas do algoritmo de assinatura DSA.

Uma assinatura digital é o resultado de um cálculo que se faz sobre um documento que a chave privada de uma pessoa, de forma que seja possível, através da chave pública desta pessoa, demonstrar que foi ela (ou a sua chave privada) que gerou a assinatura. Presume-se que a dificuldade do logaritmo não permite que se forje esta assinatura, sem o conhecimento da chave privada. Da mesma forma, se é impossível forjar a assinatura então uma assinatura válida não pode ser refutada pelo dono da chave.

Normalmente, uma assinatura digital é feita não diretamente sobre o documento em si, mas sim sobre o resultado de uma função denominada *resumo criptográfico*, em inglês simplesmente *hash*. Exemplos de funções de *hash* muito utilizadas são os algoritmos SHA-1 e MD5. Uma vantagem de se usar estas funções é que elas mapeiam um documento de tamanho arbitrário em uma pequena quantidade de bits, de forma que é difícil encontrar duas mensagens que gerem o mesmo resultado. Esta quantidade de bits, então, pode ser considerada o resumo da mensagem. Outro motivo pelo qual às vezes estas funções são necessárias é que, em alguns algoritmos, caso se assine diretamente a mensagem, e a mensagem for maior que a ordem do corpo base sendo utilizado, é fácil encontrar mensagens diferentes mas que possuam a mesma assinatura. O emprego da função de *hash* garante que isso é difícil. Na última seção do capitulo 6 é apresentada a definição formal de funções de *hash* e é introduzida uma nova função de *hash* cuja segurança é baseada no

ECDLP. Para assinaturas digitais, entretanto, qualquer uma que seja segura é suficiente.

Dado uma chave privada x e uma chave pública $Y = x \times G$ como nos sistemas anteriores, e um documento ou *hash* $m \in \mathbb{F}_p^*$ que se deseja assinar, o ECDSA funciona da seguinte forma. Escolhe-se um inteiro k descartável. Calcula-se o ponto $N = k \times G$. Dado r igual a abscissa de N (ou seja, a coordenada x do ponto N), calcula-se $w = k(m+rx)^{-1} \mod p$. A assinatura do documento é, então, o par $\langle r, w \rangle$. Note que r e w são inteiros pertencentes à $\mathbb{F}_p^*$.

Algumas precauções devem ser tomadas. É necessário garantir que $r \neq 0$, caso contrário a assinatura não dependeria de x. Também é importante ter $r \neq -\frac{m}{x}$, pois, neste caso, seria impossível calcular w. Caso alguma destas situações ocorra, ela é resolvida escolhendo-se um valor de k alternativo.

Pode-se, também, sem perda de generalidade, calcular o valor $k^{-1} \mod p$ ao invés de $(m + rx)^{-1} \mod p$.

A verificação da assinatura é feita da seguinte forma. Calcular o ponto $V = wmG + wrY$. A coordenada x do ponto V deve ser igual à r. Este cálculo funciona devido às seguintes relações:

$$\begin{aligned} w &= k(m + rx)^{-1} \mod p \\ w(m + rx) &= k \mod p \\ k &= wm + wrx \mod p \end{aligned}$$

$$\begin{aligned} N &= kG \\ kG &= (wm + wrx)G \\ (wm + wrx)G &= wmG + wrxG \\ wmG + wr(xG) &= wmG + wrY \\ wmG + wrY &= N \end{aligned}$$

Assim sendo, se x for igual a r, se tem a certeza de que a assinatura foi gerada pela chave privada correspondente à chave pública Y.

3.4 Curvas elípticas sobre corpos finitos

Como visto nos protocolos criptográficos apresentados como exemplo, as curvas normalmente são definidas sobre algum corpo finito. Serão apresentadas então algumas outras propriedades das curvas elípticas quando definidas especificamente sobre estes tipos de corpos, ainda não abordadas na introdução inicial, mas que serão de especial relevância para os demais capítulos desta dissertação.

A primeira e mais óbvia propriedade que uma curva apresenta sobre um corpo finito $\mathbb{F}_p$ é o fato de que ela possui um número finito de pontos, formando então um grupo finito. Note que o simples fato de se utilizar um corpo infinito, não implica que a curva terá um número infinito de pontos. Por exemplo, sobre $\mathbb{Q}$ é possível encontrar tanto curvas com um número finito quanto com número infinito de pontos. Entretanto, sobre $\mathbb{F}_p$, este número será sempre finito, pois independentemente da curva, as coordenadas dos pontos podem assumir somente um número limitado de valores.

Definição 3.28 *Seja $\mathcal{E}(\mathbb{K})$ uma curva elíptica definida sobre o corpo $\mathbb{K}$. Denota-se a ordem da curva elíptica $\mathcal{E}(\mathbb{K})$, ou o número de pontos da mesma, por $\#\mathcal{E}(\mathbb{K})$.*

Uma outra característica muito importante é a de que toda a curva elíptica definida sobre um corpo finito $\mathbb{F}_p$ apresenta ordem muito próxima à ordem do corpo finito. Este

é um famoso teorema provado pelo matemático *Helmut Hasse*, conhecido como *teorema de Hasse*. Segundo o teorema, a ordem de uma curva elíptica se encontra no seguinte intervalo:

$$p + 1 - 2\sqrt{p} \leq \#\mathcal{E}(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p} \tag{3.4}$$

A prova pode ser encontrada em (SILVERMAN, 1986). Este teorema implica que a ordem da curva é igual à ordem do corpo mais ou menos um valor que depende da curva, mas que é muito menor que a ordem do corpo. Uma segunda expressão muito utilizada para indicar a ordem de uma curva elíptica específica é dada pela equação

$$\#\mathcal{E}(\mathbb{F}_p) = p + 1 - t \tag{3.5}$$

onde $|t| \leq 2\sqrt{p}$. Nesta equação, o valor t é denominado **traço de Frobenius** da curva em p. De fato, uma análise empírica revela que, sobre um mesmo corpo $\mathbb{F}_p$, existem curvas com traço igual a praticamente todos os valores de t possíveis neste intervalo. Além disso, curvas distintas, mas com mesma ordem, ocorrem com uma freqüência quase uniforme.

O valor do traço de Frobenius de uma curva elíptica define diversas propriedades da mesma. Existem pelo menos duas classes de curvas que, sob certas condições, são consideradas criptograficamente fracas. A curva elíptica $\mathcal{E}(\mathbb{F}_p)$ é dita *anômala* se ela possuir traço de Frobenius t igual a 1. Neste caso, ela possui exatamente p pontos. Uma curva anômala é extremamente fraca criptograficamente pois existe um algoritmo polinomial para cálculo de logaritmos neste tipo de curva (SMART, 1999).

Outra classe de curvas consideradas inadequadas para alguns fins criptográficos são as curvas *supersingulares*. Uma curva é dita *supersingular* quando o traço de Frobenius é igual a um múltiplo da característica do corpo. Em (MENEZES, 1993) encontra-se a prova de que uma curva sobre um corpo de característica grande (maior que 3) é supersingular se e somente se $t^2 \in \{0, p, 2p, 3p, 4p\}$.

Curvas supersingulares são particularmente suscetíveis a um ataque conhecido como MOV (MENEZES; VANSTONE; OKAMOTO, 1993). Neste ataque, o logaritmo no grupo de pontos da curva é mapeado para o logaritmo em um grupo multiplicativo que é uma extensão $\mathbb{F}_{p^b}$ do corpo $\mathbb{F}_p$ onde a curva foi definida. Este mapeamento pode ser feito através dos algoritmos de *emparelhamento*, em especial, dos emparelhamentos de *Weil* e de *Tate*, apesar de existirem outros com a mesma propriedade. Para curvas supersingulares, b é de fato um valor pequeno, fazendo com que esta extensão seja pequena o suficiente para que o logaritmo discreto neste grupo multiplicativo seja mais rápido que o logaritmo elíptico, através do algoritmo de cálculo de índices.

O valor b da extensão do corpo para o qual este mapeamento é possível é chamado de **grau de imersão** da curva. De fato, este mapeamento existe para todas as curvas elípticas, porém ele é pequeno apenas para as curvas supersingulares e para um conjunto restrito de curvas não supersingulares. Para as outras o valor de b é, em geral, altíssimo, e o cálculo do emparelhamento torna-se inviável. Curiosamente, algum tempo após a apresentação do ataque MOV, surgiu a idéia de se utilizar as características peculiares dos emparelhamentos a fim de se desenvolver novos protocolos criptográficos. Um exemplo foi a sugestão de utilização de emparelhamentos para *criptografia baseada em identidades* em (BONEH; FRANKLIN, 2001). Atualmente, emparelhamentos estão entre os assuntos mais pesquisados na área de criptografia.

3.4.1 Contando os pontos de uma curva elíptica

Mesmo que se saiba qual o intervalo específico onde o número de pontos de uma curva se encontra, descobrir o número exato de pontos de uma curva elíptica é um problema bastante complexo. Existe um algoritmo com complexidade polinomial para esta tarefa, chamado de algoritmo de *Schoof* (BLAKE; SEROUSSI; SMART, 1999). Entretanto, mesmo utilizando-se as extensões propostas para ele, este algoritmo continua sendo uma alternativa bastante lenta para calcular o número de pontos de uma curva qualquer.

O algoritmo de *Schoof* utiliza algumas propriedades particulares das curvas elíptica, como o *endomorfismo de Frobenius*, para calcular o traço de Frobenius t módulo uma seqüência de valores primos pequenos. Esses cálculos permitem que o traço seja recuperado através do teorema chinês do resto (MENEZES; OORSCHOT; VANSTONE, 1996). De posse do traço de Frobenius, a equação 3.5 retorna o número de pontos.

O principal problema do algoritmo de Schoof é justamente o número de valores primos relativo aos quais é necessário calcular o traço. Para que o teorema do resto se aplique, uma determinada quantidade de valores deve estar disponível, e esta quantidade não é tão pequena, fazendo com que o algoritmo seja consideravelmente lento, mesmo que seja polinomial. Inclusive, quanto maior a ordem da curva, mais primos são necessários para se obter a ordem. Mesmo contando com algumas extensões especiais propostas por *Elkies* e *Atkins*, a complexidade mínima do algoritmo é $O(n^5)$ (BLAKE; SEROUSSI; SMART, 1999).

Existe um método alternativo para se descobrir a ordem de uma curva e que, na verdade, consiste em se construir uma curva com uma ordem pré-determinada. Este método se chama método da **Multiplicação Complexa**. A teoria da multiplicação complexa sobre curvas elípticas é extremamente densa, e está fora do escopo deste trabalho aprofundar os detalhes sobre a mesma. Ainda assim, ela é parte fundamental do algoritmo proposto, e, portanto, será apresentada no capítulo 5 de forma superficial, somente na extensão necessária. Uma abordagem bastante completa da teoria da teoria da multiplicação complexa pode ser encontrada em (COHEN, 1993).

3.4.2 Compressão de pontos

Uma técnica muito útil para se diminuir ainda mais o tamanho em bits das chaves públicas nos algoritmos que utilizam curvas elípticas é chamada de *compressão de pontos*. Note que para uma coordenada x, dada a equação da curva, existe no máximo dois pontos com esta mesma coordenada, respectivamente $(x, +\sqrt{x^3 + ax + b})$ e $(x, -\sqrt{x^3 + ax + b})$. Considerando-se que os valores das coordenadas x e y de ambos os pontos dependem simplesmente de x, é possível representar um ponto de uma curva elíptica utilizando apenas a coordenada x e mais um bit, que identifica se o ponto em questão é o que possui coordenada y positiva ou negativa. Isto reduz a representação de um ponto praticamente pela metade, pois onde eram necessários dois valores de $\mathbb{F}_p$ agora é necessário somente um mais um bit.

A única desvantagem deste método é a necessidade de se calcular uma raiz quadrada em $\mathbb{F}_p$ no momento em que o valor completo da coordenada y for necessário. Entretanto, mesmo este cálculo é bastante simples e eficiente, não sendo um fator que torne o método não prático.

4 UM GERADOR DE BITS PSEUDO-ALEATÓRIOS SEGURO BASEADO EM LOGARITMOS ELÍPTICOS

Utilizando os princípios de construção de CSPRBGs definidos nos trabalhos de Yao, Blum e Micali, Burton Kaliski na sua tese de PhD (KALISKI, 1988) construiu um gerador baseado em curvas elípticas, e que, portanto, utiliza o ECDLP, como visto no capítulo anterior, como função unidirecional. Este capítulo descreverá este algoritmo em detalhes, pois ele é uma peça fundamental para o algoritmo proposto.

As principais contribuições de Kaliski na sua tese foram duas. A primeira é um método para se encontrar um conjunto de pontos que sejam geradores de uma curva elíptica aleatoriamente criada (onde os coeficientes e o corpo são escolhidos de forma aleatória). Sabe-se que o grupo de pontos de uma curva elíptica genérica, mesmo quando é acíclico, é o resultado do produto de dois grupos cíclicos. Isso significa que qualquer grupo deste tipo obedece ao isomorfismo

$$\mathcal{E}(\overline{\mathbb{F}_p}) \cong (\mathbb{Z}/n_1\mathbb{Z}) \times (\mathbb{Z}/n_2\mathbb{Z})$$

para dois inteiros n_1 e n_2 onde n_2 divide n_1. Este fato implica que sempre existem dois elementos com estas respectivas ordens, e que, combinados, podem ser utilizados para representar *todos* os outros elementos do grupo. Em outras palavras, caso a curva forme um grupo acíclico, então ele possui necessariamente uma *tupla geradora* com no máximo dois elementos.

Na realidade, uma curva elíptica possui um número muito grande de tuplas geradoras, mesmo quando possui também elementos geradores. Baseado neste fato, Kaliski mostrou como se poderia encontrar uma tupla geradora de uma curva elíptica aleatória em tempo polinomial, com uma alta probabilidade de sucesso. Para tanto, ele se utilizou, além de diversos algoritmos probabilísticos, das propriedades dos *emparelhamentos*[1], particularmente o *emparelhamento de Weil*, o mesmo mencionado no capítulo anterior e que permite o ataque MOV.

Como já explicado anteriormente, uma alternativa ao algoritmo para encontrar tuplas geradoras é a de utilizar curvas com um número primo de pontos, de forma que todos os pontos sejam geradores. Esta será a solução adotada no algoritmo desta dissertação, e o método através do qual isso é possível será apresentado no capítulo 5. Desta forma, torna-se desnecessário a apresentação do algoritmo para busca de tuplas geradoras.

A segunda grande contribuição de Kaliski diz respeito ao seguinte. Como visto no capítulo 2, a definição de um CSPRBG depende fundamentalmente de um predicado difícil (conforme a seção 2.2.2). Kaliski desenvolveu um novo método de prova baseado em oráculos aleatórios e o utilizou para mostrar que os $\log \log p$ bits mais significativos

[1] A tese de Kaliski apresenta uma explicação bastante acessível sobre emparelhamentos.

do logaritmo discreto de um ponto – p sendo a ordem do elemento base do logaritmo – formam um predicado difícil de k-bits se e somente se o ECDLP é um problema difícil. O que Kaliski prova, na realidade, é que em qualquer grupo onde o produto escalar seja considerado unidirecional, então os $\log \log p$ bits mais significativos de seu logaritmo formam um predicado difícil. Como o grupo formado pelos pontos de uma curva elíptica possui estas características, então eles automaticamente assumem qualidade. A prova é bastante complexa, e se estende por um capitulo inteiro da sua tese. Por conveniência, ela será omitida.

4.1 O *twist* quadrático de uma curva elíptica

Antes de apresentar o algoritmo CSPRBG propriamente dito, é necessário apresentar algumas outras propriedades das curvas elípticas ainda não apresentadas. Considere a curva elíptica

$$\mathcal{E}(\mathbb{F}_p) : y^2 = f(x) \tag{4.1}$$

sobre um corpo com característica grande $\mathbb{F}_p$, sendo $a, b \in \mathbb{F}_p$ e a função $f(x) = x^3 + ax + b$. Para cada valor $x \in \mathbb{F}_p$, existe um ponto com coordenada x na curva se e somente se $f(x)$ é um resíduo quadrático módulo p.

Seja $\beta \in \mathbb{F}_p$ um não-resíduo quadrático módulo p. É fácil mostrar que β, quando multiplicado por um outro não-resíduo, resulta em um resíduo módulo p. Além disso, β multiplicado por um resíduo resulta em um não-resíduo quadrático. Utilizando-se essas propriedades, pode-se calcular uma função $f'(x)$ tal que $f'(x)$ seja um resíduo quadrático se e somente se o resultado de $f(x)$ for um não-resíduo. Mais especificamente, é simples provar o seguinte lema.

Lema 4.1 *Seja $\beta \neq 0$ um não-resíduo quadrático no corpo $\mathbb{F}_p$ de característica diferente de 2 e 3. Seja $\mathcal{E}(\mathbb{F}_p)$ a curva elíptica $y^2 = x^3 + ax + b$. Para cada valor $x \in \mathbb{F}_p$, seja $y = \sqrt{x^3 + ax + b}$.*

1. *Se y é um resíduo quadrático em $\mathbb{F}_p$, então os pontos $(x, \pm y)$ estão na curva $\mathcal{E}(\mathbb{F}_p)$.*
2. *Se y é um não-resíduo quadrático então, os pontos $(\beta x, \pm\sqrt{\beta^3} y)$ estão na curva $\mathcal{E}^t(\mathbb{F}_p)$ definida por $y^2 = x^3 + \beta^2 ax + \beta^3 b$.*
3. *Se $y = 0$, então o ponto $(x, 0)$ se encontra em $\mathcal{E}(\mathbb{F}_p)$ e o ponto $(\beta x, 0)$ se encontra em $\mathcal{E}^t(\mathbb{F}_p)$*

Prova O primeiro item é óbvio, pela definição de curva elíptica. O segundo e terceiros ficam óbvios a partir da análise da equação de $\mathcal{E}(\mathbb{F}_p)$ quando multiplicada por β^3:

$$\begin{aligned} (\beta^3)(y^2) &= (\beta^3)(x^3 + ax + b) \\ (\sqrt{\beta^3} y)^2 &= (\beta x)^3 + a\beta^2(\beta x) + b\beta^3) \end{aligned} \tag{4.2}$$

Note que a equação é exatamente a mesma, simplesmente multiplicada por um fator β^3. Faça-se então uma mudança de coordenadas. Considere uma nova equação substituindo-se $y' = \sqrt{\beta^3} y$ e $x' = \beta x$.

$$\mathcal{E}^t(\mathbb{F}_p) : (y')^2 = x'^3 + a\beta^2 x' + b\beta^3 \tag{4.3}$$

Claramente, $\mathcal{E}^t(\mathbb{F}_p)$ também é uma curva elíptica sobre $\mathbb{F}_p$. Entretanto ela possui a seguinte propriedade. Para qualquer x específico, caso y seja um não-resíduo, então automaticamente a expressão $y' = \sqrt{\beta^3}y$ é um resíduo quadrático, pois $\sqrt{\beta^3}$ sempre será um não-resíduo. Da mesma forma, caso y seja um não-resíduo, então a expressão $y' = \sqrt{\beta^3}y$ é um resíduo, provando a parte dois do lema. A terceira parte do lema é natural, já que se $y = 0$, então ambas as expressões, y e y' são resíduos com somente uma raiz, satisfazendo ambas as equações. □

Dada uma curva elíptica $\mathcal{E}(\mathbb{F}_p) : y^2 = x^3 + ax + b$, denomina-se o seu *twist*[2] qualquer curva na forma $\mathcal{E}^t(\mathbb{F}_p) : y^2 = x^3 + a\beta^2 x + b\beta^3$, com $\beta \in \mathbb{F}_p$ um não resíduo quadrático módulo p. Note que o *twist* de um *twist*, não necessariamente será a mesma curva, porém, ele possuirá exatamente o mesmo número de pontos da curva original. Isso acontece pois exatamente os mesmo valores para a coordenada x gerarão resíduos quadráticos. De fato, a multiplicação dos coeficientes a e b de uma curva elíptica por *quaisquer* valores α^4 e α^6 gera uma curva isomórfica a curva original, com exatamente o mesmo número de pontos.

Para fins de notação, é interessante definir um par de *twists* como um elemento coeso identificado pelos coeficientes da curva original mais o elemento β que dá origem ao *twist*.

Definição 4.2 *Seja $\mathbb{F}_p$ um corpo e $\beta \in \mathbb{F}_p$ um não-resíduo quadrático em $\mathbb{F}_p$. Seja a curva elíptica $\mathcal{E}(\mathbb{F}_p)$ definida por $y^2 = x^3 + ax + b$ e o seu* twist *definido por $\mathcal{E}^t(\mathbb{F}_p) : y^2 = x^3 + a\beta^2 x + b\beta^3$. Este par de curvas será denominado um* ***twisted pair****. Denota-se união disjunta $\mathcal{E}(\mathbb{F}_p) \cup \mathcal{E}^t(\mathbb{F}_p)$ por $\mathcal{T}_{a,b,\beta}(\mathbb{F}_p)$.*

Uma conseqüência importante da relação entre uma curva elíptica e o seu *twist* diz respeito ao número de pontos das curvas. Note que se para todo elemento $x \in \mathbb{F}_p$ existem dois pontos em uma curva, dois na outra ou um em cada uma, então a soma total de pontos das duas curvas é $2p + 2$, adicionando-se os pontos no infinito de cada uma delas.

Uma outra relação é importante. Considere a equação 3.5 para o número de pontos de uma curva elíptica. Se o número de pontos da soma é conhecido, então é possível se relacionar os traços de Frobenius de ambas as curvas através da equação

$$2p + 2 = p + 1 - t + p + 1 - t^t,$$

sendo t^t o traço de Frobenius do *twist*. Essa expressão, quando simplificada, resulta na expressão

$$t^t = -t,$$

indicando que o traço de uma curva é o valor negativo do traço da outra. Isso implica que a determinação do número de pontos de uma curva automaticamente fornece o número de pontos da outra.

Analisando-se a prova do lema 4.1, pode-se notar que as propriedades do *twist* de uma curva elíptica dependem fundamentalmente do conceito de resíduo quadrático em um corpo $\mathbb{F}_p$, e da noção de que enquanto alguns elementos são resíduos quadráticos (ou possuem raiz quadrada), outros não. Imagine, então, um segundo corpo finito onde todas as raízes quadradas do primeiro estejam definidas, ou seja, para qualquer elemento do primeiro corpo, sempre existe um outro elemento no segundo corpo que é a raiz quadrada dele. Isto é o que se chama de *extensão quadrática* de um corpo finito. Por exemplo,

[2] Apesar de existirem *twists* cúbicos e sêxticos, no âmbito desta dissertação um *twist* é sempre sinônimo de *twist quadrático*

a extensão quadrática de $\mathbb{F}_p$ é denotada por $\mathbb{F}_{p^2}$ e possui a raiz quadrada de todos os elementos de $\mathbb{F}_p$. Considerando-se uma extensão onde estão definidas todas as *e-ésimas* raízes (quadrada, cúbica, quarta, etc.) de todos elementos, então ele é dito um *fechamento algébrico*. O exemplo mais clássico de fechamento algébrico é o fechamento de $\mathbb{R}$ que é o corpo dos números complexos $\mathbb{C}$. Naturalmente, $\mathbb{C}$ possui todas as *e-ésimas* raízes de $\mathbb{R}$, sem exceção.

Considerando-se uma curva elíptica sobre o corpo que é a extensão quadrática do corpo original $\mathbb{F}_{p^2}$, é fácil visualizar que ela e o seu *twist* tornam-se curvas isomórficas. Como todas as raízes são definidas (inclusive do não resíduo utilizado para definir o *twist*), então para cada ponto definido em uma das curvas, sempre existe um ponto na outra curva e, portanto, existirá uma relação bijetora entre os pontos de uma curva e da outra. Este isomorfismo sobre uma extensão quadrática é extrapolado a fim de definir o conceito de *j-invariante* de uma curva.

Definição 4.3 *Duas curvas são isomórficas sobre o fechamento algébrico de um corpo $\mathbb{F}_p$ se e somente se possuírem mesma $j-invariante$. A $j-invariante$ da curva elíptica $\mathcal{E} : y^2 = x^3 + ax + b$ e com discriminante Δ é dada por*

$$j(\mathcal{E}) = \frac{1728(4a)^3}{\Delta}$$

Esta definição implica que uma curva elíptica e seu *twist* possuem mesma j-invariante, fato que pode ser provado, mas cuja prova será omitida por não acrescentar nenhum elemento novo à discussão. A j-invariante de uma curva elíptica tem um papel fundamental na criação de curvas por multiplicação complexa.

4.2 Mapeamento entre inteiros e os pontos de um *twisted pair*

Considere o *twisted pair* $\mathcal{T}_{a,b,\beta}(\mathbb{F}_p)$ como na definição 4.2. Utilizando-se a propriedades dos *twists*, é possível se conceber uma função que mapeia os pontos do conjunto de ambas as curvas no conjunto de valores inteiros do intervalo $[0, 2p + 1]$. A função é, de fato, bijetora, já que é possível a partir de um valor do conjunto calcular qual ponto o gerou.

Note que o mapeamento bijetivo entre inteiros e pontos de uma curva não é um problema trivial. Não existe uma forma *direta* de se mapear os pontos de uma curva elíptica qualquer em um intervalo que seja contínuo. É possível, por exemplo, associar-se parte das coordenadas x dos pontos de uma curva com valores em uma seqüência contínua de tal forma que todos os valores tenham um ponto representante. Entretanto, uma função dessas jamais será bijetora, pois necessariamente existirá mais de um ponto para cada valor do intervalo.

A função definida por Kaliski para solucionar este problema se vale da mesma propriedade que permite que se saiba de antemão o número de pontos de um *twisted pair* sobre um corpo $\mathbb{F}_p$. Se para todo $x \in \mathbb{F}_p$ existem pontos com coordenada x em uma curva ou $x\beta$ na outra, então, considerando-se as coordenadas x dos pontos da primeira curva e os valores $\frac{x}{\beta}$ calculados a partir das coordenadas x dos pontos da outra curva, então todos os valores $\mathbb{F}_p$ possuem dois pontos associados (em uma ou na outra curva).

Naturalmente, cada valor de coordenada $x \in \mathbb{F}_p$ pode definir dois pontos distintos na curva, e é importante distinguir entre estes dois pontos. Com este propósito, define-se a seguir a função *sinal*, cujo princípio é o mesmo da compressão de pontos.

Definição 4.4 *Seja $\mathbb{F}_p$ um corpo. Define-se a função* sinal $: \mathbb{F}_p \rightarrow \{1, 0\}$ *como*

$$sinal(y) = \begin{cases} 0, \text{ caso } \frac{(p-1)}{2} \geq y \geq 0, \\ 1, \text{ caso contrário.} \end{cases} \tag{4.4}$$

Em outras palavras, a função sinal simplesmente retorna o sinal de um elemento de $\mathbb{F}_p$, definido como 1 se e somente se o elemento for maior que a metade da ordem do corpo. Utilizando-se a esta função sinal, pode-se distinguir entre dois pontos de mesma abscissa.

Finalmente, define-se a função que mapeia pontos em inteiros. A função recebe um *twisted pair* e um ponto de uma das curvas, e retorna um valor inteiro.

Definição 4.5 *Seja $\mathcal{T}_{a,b,\beta}(\mathbb{F}_p)$ um twisted pair e $P \in \mathcal{T}_{a,b,\beta}(\mathbb{F}_p)$ um ponto de uma das curvas do par. A função χ que recebe $\mathcal{T}_{a,b,\beta}(\mathbb{F}_p)$ e P como parâmetros e retorna um valor no intervalo $[0, 2p+1]$ é definida da seguinte forma:*

$$\chi[\mathcal{T}_{a,b,\beta}(\mathbb{F}_p)](P) = \begin{cases} 2x + sinal(y) & se \quad P = (x, y), y \neq 0; \\ \frac{2x}{\beta} + sinal(y) & se \quad P = (x, y)^t, y \neq 0; \\ 2x & se \quad P = (x, 0); \\ \frac{2x}{\beta} + 1 & se \quad P = (x, 0)^t; \\ 2p & se \quad P = \infty; \\ 2p + 1 & se \quad P = \infty^t. \end{cases} \tag{4.5}$$

sendo os pontos com um t sobrescrito pontos advindos da curva com coeficientes $a\beta^2$ e $b\beta^3$ (o twist).

A função χ mapeia cada um dos pontos do *twisted pair* em um único valor do intervalo $[0, 2p+1]$. Cada um dos dois pontos para uma coordenada x que satisfaz uma das curvas, vira dois valores, um par e outro ímpar. Para pontos com coordenada $y = 0$, uma das curvas possui o valor par e outra possui o valor ímpar. E o mesmo acontece para os pontos no infinito.

4.3 O algoritmo do CSPRBG

No trabalho de Kaliski, o objetivo era demonstrar que *existe* um gerador seguro baseado na dificuldade de se computar logaritmos elípticos e que utiliza um número polinomial de bits realmente aleatórios e retorna uma quantidade maior de bits pseudo-aleatórios. Portanto, na sua versão original, o algoritmo randomiza todos os parâmetros: o corpo finito, os coeficientes das curvas, e o valor inicial. Após isso, ele descobre geradores de cada curva utilizando os algoritmos propostos para finalmente executar o gerador e retornar um número específico de bits criptograficamente seguros.

De fato, o par de curvas, os pontos geradores e o valor inicial não precisam necessariamente ser randomizados para que o sistema seja seguro. Basta que o ECDLP sobre as curvas o seja. Desta forma, será apresentada uma versão simplificada do algoritmo, onde o par de curvas e o valor inicial são dados como parâmetros. Como já mencionado, é possível se trabalhar com curvas de ordem prima onde é fácil se obter um único ponto gerador em cada uma das curvas. Portanto, a versão apresentada do algoritmo recebe como parâmetros também um único gerador para cada curva.

O algoritmo `Kaliski_CSPRBG` recebe como entrada um corpo finito $\mathbb{F}_p$, um *twisted pair* $\mathcal{T}_{a,b,\beta}(\mathbb{F}_p)$ sobre este corpo, pontos geradores de cada uma das curvas, G, G^t – G^t

sendo o gerador da curva com coeficientes $a\beta^2$ e $b\beta^3$ – e uma semente inicial que é um valor inteiro s no intervalo $[0, 2p + 1]$. Uma outra informação que o algoritmo necessita é o número de pontos das curvas. Como a ordem do *twist* é dedutível a partir da ordem da curva original, somente a ordem da curva com coeficientes a e b é informada, através do valor o. Todas estas informações se resumem aos valores p, a, b, β, o, G, G^t e s, que são os parâmetros de fato do algoritmo. Além destes, é passado um parâmetro extra l que indica o número de bits pseudo-aleatórios a serem retornados.

Quatro funções especiais são utilizadas no algoritmo formalizado a seguir. A primeira é a função de `Concat(s1, s2)`, que retorna a concatenação das duas strings *binárias* que são passadas como parâmetro. Outra é a função `TamanhoemBits(s)` que retorna o número de bits da string s.

A função `Msb(t, n, i)` retorna os i *bits mais significativos* do valor t. Estes bits mais significativos não são os bits literais da representação binária de t, mas sim bits definidos em função do intervalo onde $t \in [0, n-1]$ se encontra, cujo limite superior é identificado pelo valor n. O primeiro bit mais significativo de t com respeito ao intervalo n é definido como

$$\text{Msb}(t, n, 1) \begin{cases} 0, \text{ caso } t < \frac{n}{2} \\ 1, \text{ caso } t \geq \frac{n}{2} \end{cases} \tag{4.6}$$

Utilizando esta formulação de forma recursiva, define-se a função genérica `Msb(t, n, i)` multiplicando-se o valor t por dois e chamando novamente a função, de forma a recuperar o valor do segundo bit mais significativo. Este procedimento se traduz no seguinte algoritmo:

Algoritmo 3 $Msb(t, n, i)$

1: **if** $(i = 0)$ **then**
2: $\quad Terminar$;
3: **end if**
4: **if** $(t \geq \frac{n}{2})$ **then**
5: $\quad$ **return** $Concat(1, Msb(2 * t \mod n, n, i - 1))$;
6: **else**
7: $\quad$ **return** $Concat(0, Msb(2 * t \mod n, n, i - 1))$;
8: **end if**

A última função que aparece no algoritmo gerador representa a operação de *produto escalar* sobre uma curva elíptica, `ProdEscalar(p, a, b, G, i)`. Ela recebe os valores a e b, um primo p, um ponto qualquer da curva elíptica definida pelos coeficientes a e b sobre $\mathbb{F}_p$ e retorna um ponto que é o produto escalar do valor i com o ponto G sobre a mesma curva. Os detalhes da implementação da função são irrelevantes para o algoritmo, mesmo que, de fato, ela seja a operação mais custosa do mesmo. Referenciando-se a teoria apresentada no capítulo 2, é interessante notar que a função `ProdEscalar` é a função unidirecional do algoritmo e a que função `Msb` é o predicado difícil.

Utilizando-se estas funções, pode-se expressar o gerador de bits pseudo-aleatórios criptograficamente seguros definido por Kaliski algoritmicamente. No algoritmo 4, P é uma variável que assume o valor de um ponto de uma das curvas.

Pode-se verificar que, em linha gerais, o algoritmo `Kaliski_CSPRBG` é muito simples. A partir da semente s, ele gera uma seqüência pseudo-aleatória de valores utilizando o produto escalar no par de curvas elípticas e a função de mapeamento. Sempre que o valor de um passo for maior que a ordem da primeira curva, então ele subtrai a ordem da

Algoritmo 4 $Kaliski\_CSPRBG(p, a, b, \beta, o, G, G^t, s, l)$

1: $Resultado :=$ " ";
2: **while** $(TamanhoEmBits(Resultado) < l)$ **do**
3: **if** $(s < o)$ **then**
4: $P := ProdEscalar(p, a, b, G, s)$;
5: **else**
6: $P := ProdEscalar(p, a\beta^2, b\beta^3, G^t, s - o)$;
7: **end if**
8: $s := \chi[\mathcal{T}_{a,b,\beta}(\mathbb{F}_p)](P)$;
9: **if** $(s < o)$ **then**
10: $Resultado := Concat(Resultado, Msb(s, o, \log\log p))$;
11: **else**
12: $Resultado := Concat(Resultado, Msb(s - o, 2p + 2 - o, \log\log p))$;
13: **end if**
14: **end while**
15: **return** $Resultado$;

mesma, fazendo com que o valor necessariamente fique menor que a ordem da segunda curva, e então calcular o produto escalar nela. Ao ponto resultado é aplicada a função χ que então gera um novo valor no intervalo e o processo se repete. Cada uma das curvas gera, aproximadamente, metade dos valores dos valores do intervalo, com uma diferença que depende do traço de Frobenius das curvas.

A seqüência de bits pseudo-aleatórios é gerada concatenando-se os bits mais significativos (retornados pela função `Msb`) dos logaritmos, que são justamente o predicado difícil da função unidirecional do gerador. Os logaritmos são os próprios valores da gerados pela função χ, sendo subtraídos pela ordem da primeira curva quando forem maiores que a mesma.

5 UM NOVO ALGORITMO DE CHAVE PÚBLICA SEMANTICAMENTE SEGURO BASEADO EM CURVAS ELÍPTICAS

Este capítulo vai abordar a contribuição de fato deste trabalho. A contribuição diz respeito a um algoritmo de chave pública que possui como vantagem a seu favor as seguintes características:

- **Segurança redutível unicamente ao ECDLP.** A segurança do algoritmo é completamente redutível à dificuldade do ECDLP e portanto somente uma premissa é necessária para que o algoritmo seja seguro. É trivial entender que quanto menos premissas um algoritmo de criptografia fizer com respeito a segurança, menos provável de ser quebrado ele será. Isto é claro, dado que cada premissa é uma característica a mais que um atacante possui para explorar na atividade de quebrá-lo. Além disso, não são utilizados conceitos genéricos comumente empregados, como o modelo de oráculo aleatório (ou *random oracle model*). Oráculos aleatórios, apesar de teoricamente seguros, implicam a utilização de funções de *hash* reais na sua implementação, e incorporam toda a fragilidade que a função utilizada apresentar (WANG; YU, 2005).

- **Segurança semântica.** Como será mostrado na análise de segurança, é possível provar que o algoritmo possui segurança semântica, o que talvez seja uma das suas mais importantes características. Como já dito anteriormente, isso faz com que a obtenção de qualquer texto cifrado não forneça ao atacante informação alguma sobre o texto claro correspondente.

- **Implementação para curvas elípticas sobre corpos onde o ECDLP é seguro.** O fato do ECDLP ser um problema mais difícil que os outros problemas sobre os quais os demais algoritmos semanticamente seguros são baseados, como fatoração ou residuosidade quadrática, faz com que em média a execução do algoritmo ocupe muito menos memória do que os outros algoritmos. A diferença de tamanho entre os corpos utilizados é, de fato, imensa, caindo de 1024 bits para fatoração para 160 bits para o ECDLP. Isso faz com que o algoritmo ocupe menos memória em tempo de execução e possua chaves menores. Mais especificamente, a chave pública no algoritmo proposto, utilizando a alternativa de compressão, ocupa menos de 500 bits no total, enquanto nos outros casos ela possui no mínimo 1024 bits. Da mesma forma, enquanto nos demais algoritmos a chave privada ocupa em torno de 1024 bits, no algoritmo proposto ela ocupa 320 bits (na realidade, dois valores de 160 bits).

Como para a apresentação de qualquer algoritmo de chave pública, é necessário que os seguintes itens sejam apresentados em detalhes:

- **Criação do par de chaves.** Serão apresentados algoritmos para criar os parâmetros públicos necessários e deduzir a chave pública a partir da chave privada. Serão discutidas algumas técnicas para a criação destes elementos, com considerações sobre vantagens e possíveis modificações. Além disso, serão apresentadas algumas estratégias de compressão dos parâmetros públicos que se mostram bastante extremamente efetivas.

- **Cifragem.** Este é o procedimento necessário para, a partir do texto claro e da chave pública construída utilizando alguma estratégia do item anterior, criar-se o texto cifrado. Ele consistirá em um ponto de uma curva elíptica, mais um número de bits cifrados exatamente igual ao número de bits do texto claro original.

- **Decifragem.** De posse do texto cifrado, os procedimentos necessários para, utilizando a chave privada, obter o texto claro respectivo.

- **Análise de segurança.** O próximo capítulo é completamente dedicado à análise de segurança do algoritmo. Será mostrado que, de fato, ele possui toda a sua segurança baseada no ECDLP, como dito. Será demonstrado, também, que ele é semanticamente seguro. Além disso, neste mesmo capítulo, é proposta uma extensão do algoritmo a qual possui segurança contra ataques de texto cifrado adaptativo. Para esta extensão, é apresentada uma função de resumo criptográfico (*hash*) baseada no ECDLP, preservando a premissa de que o algoritmo tem sua segurança completamente redutível a este problema.

5.1 Criação do par de chaves

Antes de se calcular, de fato, uma instância do par de chaves do algoritmo, é necessário o estabelecimento do que é freqüentemente chamado de *parâmetros públicos* de um algoritmo de chave pública. Os parâmetros públicos de um algoritmo são os parâmetros a partir dos quais um ou mais pares de chaves pública e privada são criados. Apesar de, em alguns algoritmos, as chaves pública e privada se confundirem com os parâmetros públicos, em diversos algoritmos isto não ocorre, em especial os baseados em grupos.

A título de exemplo, no protocolo Diffie-Hellman, as chaves públicas são os valores $g^{x_a} \mod p$ e $g^{x_b} \mod p$, sendo as respectivas chaves privadas os valores x_a e x_b. Desta forma, os valores g e p são considerados os parâmetros públicos do algoritmo, e podem ser reutilizados diversas vezes para muitos pares de chaves. Utilizando-se a mesma lógica, a curva elíptica, o corpo $\mathbb{F}_p$ e o ponto G utilizados no algoritmo ECDH são os parâmetros públicos deste algoritmo. Já como um contra-exemplo, não existem parâmetros público no algoritmo RSA[1].

Note que os parâmetros públicos de um algoritmo sempre podem ser considerados como parte da chave pública, e, assim, ser utilizados para somente um par de chaves. Entretanto, a reutilização de parâmetros públicos pode trazer diversas vantagens. Se eles deixam de fazer parte da chave pública, esta então tem seu tamanho reduzido. Se eles são

[1] Com exceção talvez do expoente público que freqüentemente é reutilizado, mesmo que isso não traga vantagem alguma.

universalmente conhecidos, então eles podem ser embutidos diretamente na implementação dos algoritmos, ocasionando ganhos de performance. Além disso, eles podem ser estabelecidos por instituições confiáveis, o que, via de regra, permite que mais pessoas os testem, diminuindo as chances de algum furo de segurança mais acentuado.

Por motivos de simplicidade, o algoritmo aqui proposto presumirá que estes parâmetros fazem parte da chave pública. Caso os parâmetros públicos não façam parte da chave pública, necessariamente eles devem ser obtidos de alguma outra forma. Portanto, na prática, não importa se eles fazem parte ou não da chave. Assim, não será feita esta distinção.

O algoritmo proposto possui como parâmetro público um *twisted pair* de ordem prima, que é definido da seguinte forma:

Definição 5.1 *Um **twisted pair de ordem prima** é um* twisted pair $\mathcal{T}_{a,b,\beta}(\mathbb{F}_p)$ *como na definição 4.2 onde ambas as curvas do twist possuem um número primo de pontos. Denota-se um* twisted pair de ordem prima *por* ${\mathcal{T}'}_{a,b,\beta}(\mathbb{F}_p)$

Sabe-se que se uma curva elíptica sobre um corpo $\mathbb{F}_p$ tem traço de Frobenius t, então ela possui número de pontos $p + 1 - t$ (como visto no capítulo 3). Além disso, um *twist* desta curva possui exatamente $p + 1 + t$ pontos. O problema a ser resolvido pode ser enunciado da seguinte forma: encontrar uma curva elíptica sobre um corpo $\mathbb{F}_p$, com traço de Frobenius t, e cujos valores $p + 1 + t$ e $p + 1 - t$ sejam ambos primos.

Existe pelo menos uma maneira óbvia de se procurar um *twisted pair* de ordem prima, que é utilizando-se o algoritmo de *Schoof* para contagem de pontos de uma curva elíptica qualquer, apresentado na seção 3.4.1. Fixa-se um corpo $\mathbb{F}_p$ e escolhe-se os coeficientes $a, b \in \mathbb{F}_p$ de uma curva elíptica de forma aleatória. Utilizando o algoritmo de Schoof, calcula-se o número de pontos da curva. Se ele for primo, verifica-se a ordem do *twist*, que pode ser imediatamente calculado pois sabe-se que o somatório do número de pontos de uma curva com o do seu *twist* é exatamente $2p + 2$. Portanto, verifica-se se o valor

$$\#\mathcal{E}^t(\mathbb{F}_p) = 2p + 2 - \#\mathcal{E}(\mathbb{F}_p).$$

é primo, e, caso seja, o par de curvas foi encontrado.

Apesar de funcionar, é importante notar que este método é extremamente ineficiente. O problema principal é justamente o algoritmo de Schoof, que mesmo na sua versão com as modificações de *Elkies* e *Atkins* possui complexidade $O(n^5)$. Como é necessário executar o algoritmo diversas vezes, o procedimento complexo, mesmo que polinomial, pode levar um tempo muito grande para encontrar as curvas.

Uma forma muito mais eficiente de se encontrar *twisted pairs* de ordem prima é utilizar a teoria de Multiplicação Complexa de curvas elípticas. Utilizando-se métodos desta teoria, é possível se construir uma curva elíptica sobre um corpo $\mathbb{F}_p$ com traço de Frobenius, e portanto ordem, pré-determinado. Ao invés de se procurar curvas com uma ordem prima, o método permite procurar traços e corpos diretamente e, a partir de um traço e um corpo específicos, obter os coeficientes da curva.

A teoria compreende uma vasta área da matemática a qual somente uma pequena parte será vasculhada. Uma explicação bastante compreensiva sobre o método de Multiplicação Complexa (ou método CM simplificadamente), com enfoque em uma perspectiva computacional, pode ser encontrada em (COHEN, 1993). A próxima seção apresentará um algoritmo que utiliza o método para encontrar as curvas mas somente a teoria imprescindível para o entendimento do método será apresentado.

5.1.1 Encontrando *twisted pairs* de ordem prima

A construção de uma curva a partir do método CM inicia pela solução da seguinte equação:

$$t^2 = 4p - Dy^2 \tag{5.1}$$

Há quatro valores desconhecidos nesta equação. O valor t será o traço de Frobenius de uma curva elíptica sobre o corpo finito $\mathbb{F}_p$. Isso significa que para determinados valores p e t que possuam soluções D e y, existe uma curva elíptica com número de pontos $p+1-t$ e com twist de ordem $p+1+t$. Desta forma, basta encontrar os valores p e t que tornem o valor destas expressões primo.

Quanto aos outros valores, y pode ser qualquer inteiro. Entretanto D é um valor especial. Ele é um valor negativo, congruente a 0 ou 1 módulo 4, livre de quadrados[2] e é o discriminante do um corpo quadrático imaginário $\mathbb{Q}(\sqrt{D})$.

Por enquanto, considere D como uma constante na equação. Devido às características do valor D, não necessariamente existe uma solução p para qualquer valor t e vice versa. Portanto, uma estratégia para a solução da mesma é necessária. De fato, existem diversas formas de solucionar esta equação. Uma alternativa é notar que é possível visualizar a equação como uma equação de Pell. Uma equação de Pell é uma equação do tipo

$$1 = u^2 + Dv^2$$

e, para um valor D específico pode ser solucionada através do método de *frações contínuas* a partir de $\sqrt{D}$. Mais detalhes sobre este método de resolução de equações de Pell podem ser encontrados em (YAN, 2000). Entretanto, como para um D fixo a equação pode não possuir solução, será considerada uma outra abordagem.

A proposta é a de se modificar a equação 5.1 de forma que ela dependa somente de uma única variável x. Isso permite uma varredura incremental de possíveis valores x, e o pronto teste de primalidade dos valores $p+1-t$ e $p+1+t$. Caso estes não sejam primos, descarta-se este valor x e testa-se o próximo.

Implementa-se a idéia substituindo-se as variáveis p, t e y por polinômios que dependam da mesma variável x. Como y pode ser qualquer valor, ele pode ser substituído pelo polinômio constante 1. Já o valor t necessita ser um valor ímpar para que a ordem das curvas possa ser prima. Assim, ele pode ser substituído pelo polinômio quadrático

$$t(x) = 2x^2 - 2x + 1, \tag{5.2}$$

que sempre é ímpar para qualquer x. Finalmente, reescreve-se a equação 5.1 na forma

$$p = \frac{t^2 + Dy^2}{4} \tag{5.3}$$

e, substituindo y e t, obtém-se o polinômio

$$\begin{aligned} p(x) &= \frac{(2x^2 - 2x + 1)^2 + D}{4} && (5.4)\\ &= \frac{4x^4 - 8x^3 + 8x^2 - 4x + 1 + D}{4} && (5.5)\\ &= x^4 - 2x^3 + 2x^2 - x + \frac{1+D}{4}. && (5.6) \end{aligned}$$

[2] Isto é, não divisível por nenhum inteiro do tipo n^2.

Para que $p(x)$ seja inteiro para qualquer x, é necessário que a expressão

$$\frac{1+D}{4}$$

também o seja. Isso restringe efetivamente os valores de D para valores onde $|D| \equiv 3 \mod 4$. Além disso, esta expressão precisa fornecer valores ímpares, ou então $p(x)$ será par, e portanto nunca será primo. Forçando-se $|D| \equiv 3 \mod 8$, isto é obtido.

Encontrar a curva agora resume-se à simples tarefa de avaliar a expressão $p(x)$ para diferentes valores de x. Quando $p(x)$ for primo, verificar se as expressões $p(x)+1-t(x)$ e $p(x)+1+t(x)$ também são ambas primas. Simplificadamente, basta que, para um valor x, os polinômios

$$p(x) = x^4 - 2x^3 + 2x^2 - x + \frac{1+D}{4} \tag{5.7}$$

$$n(x) = x^4 - 2x^3 + 4x^2 - 3x + 2 + \frac{1+D}{4} \tag{5.8}$$

$$n'(x) = x^4 - 2x^3 + x + \frac{1+D}{4} \tag{5.9}$$

simultaneamente resultem em valores primos. Note que os polinômios esperam o valor positivo de D, ou seja, $|D|$.

Ao se encontrar um valor x que obedeça a estes requisitos, as curvas foram encontradas.

5.1.2 Calculando os coeficientes da curva

Neste ponto do processo de geração das curvas, são conhecidos três valores: o valor primo $p(x)$, o valor do traço de Frobenius $t(x)$ que identifica o *twisted pair* de ordem prima com ordens $n(x)$ e $n'(x)$ e o valor constante D proveniente da equação 5.6. Utilizando-se estes valores, é possível calcular-se os coeficientes das curvas elípticas sobre $\mathbb{F}_p$ que possuem estas ordens. Para tanto, é necessário voltar-se novamente para a teoria de multiplicação complexa de curvas elípticas.

Como dito anteriormente, D é o discriminante de um corpo quadrático imaginário. Em última instância, isso significa que D define unicamente um polinômio $H_D(x)$ chamado *polinômio de classe de Hilbert*, que será necessário para obter as curvas. O grau deste polinômio é o *número de classe* de uma ordem quadrática imaginária com discriminante D. Calcular este polinômio não é nem de perto uma tarefa trivial. Entretanto, como se está presumindo que D é um valor fixo, uma alternativa é pré-calcular os polinômios de alguns valores específicos de D e utilizá-los diretamente. A tabela 5.1 apresenta os polinômios de alguns valores D que possuem número de classe 1 e tem valores absolutos congruentes a 3 módulo 8 (e, portanto, podem ser utilizados no algoritmo da seção anterior).

Apesar de curta, o exame da tabela 5.1 torna evidente a tendência de que, quanto menor (ou mais negativo) o valor D, mais complexo é o polinômio correspondente. Essa característica de fato se confirma, de forma uniforme, para o decrescimento progressivo dos valores de D (com poucas exceções). Quanto mais negativo for o valor D, maior tende a ser não só grau do polinômio de Hilbert correspondente, mas também dos seus coeficientes. Para se ter uma idéia de quão rápido é esse crescimento, o polinômio que corresponde ao discriminante $D = -4195587$ (sete dígitos) possui grau 328 e seus coeficientes, em representação ASCII, somam 132Kbytes. Já o polinômio de discriminante

Tabela 5.1: Polinômios de Hilbert para alguns discriminantes D.

D	$H_D(x)$
-11	$x + 32768$
-19	$x + 884736$
-43	$x + 884736000$
-67	$x + 147197952000$
-163	$x + 262537412640768000$

$D = -16397831$ (oito dígitos) possui grau 3831, com coeficientes que superam em muito a casa dos megabytes. Atualmente, o limite prático da computação destes polinômios encontra-se em discriminantes da ordem de $D = -10^{10}$. Conclui-se, então, que a pré-computação destes polinômios talvez seja, de fato, a melhor estratégia a ser utilizada.

Dado o polinômio $H_D(x)$ correspondente ao valor D que foi utilizado nas equação que determinou os valores p e t, a teoria de multiplicação complexa diz que a raiz de $H_D(x)$ módulo p é a j-invariante do *twisted pair* que possui ordens respectivamente $p + 1 + t$ e $p + 1 - t$. No capítulo anterior foi apresentada a definição de j-invariante e, também, o fato de que uma curva e seu *twist* possuem mesma j-invariante. Utiliza-se esta propriedade para calcular os coeficientes das curvas elípticas correspondentes. Dado uma j-invariante (a partir da raiz de $H_D(x)$ módulo p) e um não resíduo quadrático $\beta \in \mathbb{F}_p$, o *twisted pair* é calculado em dois passos. Inicialmente, obtém-se a constante c utilizando a expressão

$$c = \frac{j}{1728 - j} \mod p \tag{5.10}$$

$$\tag{5.11}$$

onde j é a j-invariante. Em seguida as equações das curvas são determinadas, utilizando as expressões

$$\mathcal{E}(\mathbb{F}_p) : y^2 = x^3 + 3cx + 2c \mod p, \tag{5.12}$$

$$\mathcal{E}^t(\mathbb{F}_p) : y^2 = x^3 + 3c\beta^2 x + 2c\beta^3 \mod p. \tag{5.13}$$

Note que o *twist* é calculado como explicado no capítulo anterior, simplesmente multiplicando os coeficientes da primeira curva pelo quadrado e pelo cubo do não-resíduo.

Uma consideração importante acerca desse método é o fato de que ele não permite dizer diretamente qual a ordem de cada uma das curvas $\mathcal{E}(\mathbb{F}_p)$ e $\mathcal{E}^t(\mathbb{F}_p)$. Sabe-se, por construção, que uma delas possui ordem $p + 1 - t$ e a outra possui ordem $p + 1 + t$, mas não se sabe qual possui qual. Neste momento, o fato de que o conjunto de pontos de ambas as curvas formam um grupo torna-se particularmente útil. Como visto anteriormente, o produto escalar de um elemento de um grupo qualquer pela ordem do mesmo resulta na identidade do grupo. Dado que $p + 1 - t$ é necessariamente diferente de $p + 1 + t$, então o produto escalar de um destes valores sobre qualquer ponto de cada uma das curvas vai resultar no ponto no infinito em uma delas, e em um ponto diferente do infinito na outra. Este procedimento simples então, é utilizado para esta determinação.

Considerando-se que o polinômio de Hilbert foi pré-computado, a parte mais trabalhosa do cálculo dos coeficientes das curvas é calcular a raiz do polinômio sobre o corpo $\mathbb{F}_p$. Apesar de existirem fórmulas fechadas para extração de raízes de polinômios até o grau 4, a partir do grau 5 sabe-se que estas fórmulas não existem, e a solução é utilizar algoritmos que não são muito simples. Um exemplo de algoritmo genérico para a extração de raízes de polinômios pode ser encontrado em (COHEN, 1993).

Por outro lado, calcular raízes de polinômios de grau 1 é trivial. Se o polinômio é dado por $ax + b = 0$ a raiz deste polinômio é $-ba^{-1} \mod p$. No caso dos polinômios mencionados na tabela 5.1, cujo coeficiente a é 1, nem o cálculo do inverso de um número é necessário, bastando-se determinar $-b \mod p$.

5.1.3 Compressão dos parâmetros públicos

Considerando-se polinômios de Hilbert pré-computados e a utilização de polinômios de grau 1, os procedimentos anteriores são extremamente simples. O passo mais lento do processo como um todo é encontrar o valor x que define o corpo finito e torna as ordens do *twisted pair* primas. Excluindo-se este passo, todos os passos que sobram do algoritmo são *eficientes* e *determinísticos*. Isto é, de posse do valor x e do valor D utilizado no polinômio 5.6, é possível se determinar os valores p, t, as ordens e as curvas e estes serão sempre os mesmos.

Além disso, dado que as curvas possuem ordem prima, qualquer ponto de cada uma das curvas é elemento gerador das mesmas, como demonstrado no teorema 3.23. Portanto pode-se, utilizando-se um algoritmo fixo, encontrar deterministicamente sempre os mesmos geradores. Dado um *twisted pair* de ordem prima ${\mathcal{T}'}_{a,b,\beta}(\mathbb{F}_p)$, isto pode ser feito considerando-se as propriedades do *twist* de uma curva elíptica.

Sabe-se que, para qualquer valor $x \in \mathbb{F}_p$, se x não é a coordenada de um ponto da curva elíptica $\mathcal{E}(\mathbb{F}_p)$ então $x\beta \in \mathbb{F}_p$ é uma coordenada do seu *twist* $\mathcal{E}^t(\mathbb{F}_p)$. Isso implica que a coordenada $x = 0$ está somente em uma das curvas, mais especificamente, na curva onde o termo independente da equação é um resíduo quadrático. O termo independente da primeira curva é b e do *twist* é $b\beta^3$, portanto, se b é um resíduo quadrático, então os pontos $(0, \pm\sqrt{b})$ estão na primeira curva, e, caso contrário, $b\beta^3$ é um resíduo e os pontos $(0, \pm\sqrt{b\beta^3})$ se encontram no *twist*. Pode-se estabelecer então que o gerador da curva que possui pontos com coordenada $x = 0$ é o ponto $(0, y)$ onde y é a raiz positiva do termo independente da equação.

Definido o ponto gerador de uma das curvas, para se encontrar um gerador da outra curva, basta percorrer seqüencialmente os valores de coordenada x começando por $x = 1$ e, para cada um, verificar se existe um ponto na curva com esta coordenada. Devido à distribuição de resíduo quadráticos (e conseqüentemente se a equação $y = \sqrt{x^3 + ax + b} \mod p$ possui solução), o número esperado de tentativas é dois (já que metade dos valores do corpo são resíduos). Apesar deste procedimento ser em essência exponencial, não se tem notícia de algum um caso em que este procedimento levasse muito tempo. Portanto, pode-se considerar que ele é prático.

Ambos os procedimentos combinados gerarão, deterministicamente, exatamente os mesmos pontos. Assim, pode-se considerar que um *twisted pair* de ordem prima automaticamente define os seus próprios pontos geradores.

O último valor que resta determinar-se para que de fato x e D definam deterministicamente todos os parâmetros públicos do algoritmo é o não-resíduo quadrático β utilizado para definir o *twist* da curva. Apesar de, no caso geral, não existir um algoritmo polinomial determinístico para se encontrar um não-resíduo quadrático módulo p, pode-se resolver o problema de forma trivial utilizando-se o seguinte teorema acerca do símbolo de Legendre (conforme definição 2.10). O objetivo é encontrar deterministicamente um valor a tal que $\left(\frac{a}{p}\right) = -1$. Assim, o seguinte teorema torna-se útil (mais detalhes em (STINSON, 1995)).

Teorema 5.2 *Seja p um valor primo. Então*

$$\left(\frac{a}{p}\right) = a^{\left(\frac{p-1}{2}\right)} \mod p$$

Agora, considere que $p \equiv 3 \mod 4$. Isso significa que $\frac{p-1}{2}$ é um valor ímpar. Elevar o valor -1 (ou $p-1 \mod p$) a uma potência ímpar resulta no próprio valor -1. Isso implica no seguinte corolário, cuja prova é óbvia e portanto será omitida.

Corolário 5.3 *Seja $p \equiv 3 \mod 4$ um valor primo. Então $p-1$ é um não-resíduo quadrático módulo p.*

Portanto, caso se considere somente corpos onde $\mathbb{F}_p$ onde $p \equiv 3 \mod 4$, então $p-1$ será um não resíduo quadrático. Além disso, existe a seguinte vantagem de se utilizar $p-1$ como não resíduo, que é o fato de que $(p-1)^2 = 1$ e $(p-1)^3 = p-1$.

Juntando-se todas estas idéias desta seção, chega-se ao seguinte resultado:

Lema 5.4 *Dado um valor negativo $D \in \{-11, -19, -43, -67, -163\}$, e um valor x que torne os polinômios $n(x)$ e $n'(x)$ definidos nas equações 5.15 e 5.16 em valores primos e que torne o polinômio $p(x)$ da equação 5.14 um valor primo congruente a 3 módulo 4. Então D e x definem deterministicamente o* twisted pair *de ordem prima $\mathcal{T}'_{a,b,\beta}(\mathbb{F}_p)$, o que implica uma função determinística entre as tuplas*

$$\langle x, D\rangle \Longrightarrow \langle p, a, b, a^t, b^t, n, n^t, G, G^t\rangle$$

onde F_p *é um corpo, a e b são os coeficientes de uma curva elíptica sobre este corpo, n é sua ordem e G é seu ponto gerador, e os componentes do* twist *são os mesmo identificados pelo t sobrescrito.*

Prova Considere o algoritmo 5. Todos os seus passos são a explicitação das propriedades discutidas nas seções anteriores. Claramente ele é determinístico e calcula todos os parâmetros do *twisted pair* de ordem prima a partir dos valores x e D. □

O algoritmo 5 pode facilmente ser estendido para suportar valores D com número de classe 2, já que raízes de polinômios de grau 2 têm solução fechada. Entretanto, o fato de uma equação deste tipo possuir mais de uma raiz faz com que seja necessária utilização consistente de uma delas para que as curvas e os geradores sejam sempre os mesmos. Mesmo que a segunda raiz de uma equação quadrática indique outro par de curvas com os mesmos número de pontos, os pontos geradores e as chaves públicas são relativos a um único par, e não necessariamente estarão neste outro par. Assim, uma regra como, por exemplo, a utilização sempre da raiz com menor valor absoluto, faz-se necessária.

Finalmente, o algoritmo de descompressão sugere o seguinte resultado:

Teorema 5.5 *Qualquer twisted pair de ordem prima $\mathcal{T}'_{a,b,\beta}(\mathbb{F}_p)$ encontrado utilizando-se o algoritmo deste capítulo pode ser representado sem ambigüidade pelos valores x e D utilizados para encontrá-lo. Denota-se um* ***twisted pair de ordem prima comprimido*** *por*

$$\mathcal{T}_{x,D}.$$

Algoritmo 5 $ParametrosTwistedPair(x, D)$

1: $p := x^4 - 2x^3 + 2x^2 - x + \dfrac{1+|D|}{4}$;
2: $o_1 := x^4 - 2x^3 + 4x^2 - 3x + 2 + \dfrac{1+|D|}{4}$;
3: $o_2 := x^4 - 2x^3 + x + \dfrac{1+|D|}{4}$;
4: **if** $(D = -11)$ **then** $j := p - 32768 \mod p$;
5: **if** $(D = -19)$ **then** $j := p - 884736 \mod p$;
6: **if** $(D = -43)$ **then** $j := p - 884736000 \mod p$;
7: **if** $(D = -67)$ **then** $j := p - 147197952000 \mod p$;
8: **if** $(D = -163)$ **then** $j := p - 262537412640768000 \mod p$;
9: $c := \dfrac{j}{1728 - j} \mod p$;
10: $a := 3c \mod p$;
11: $a^t := a \mod p$;
12: $b := 2c \mod p$;
13: $b^t := p - b \mod p$;
14: **if** $\left(\dfrac{b}{p}\right) = 1$ **then**
15: $G := (0, +\sqrt{b})$
16: $x := 1$;
17: **while** $\left(\dfrac{x^3 + a^t x + b^t}{p}\right) = -1$ **do**
18: $x := x + 1$;
19: **end while**
20: $G^t := (x, +\sqrt{x^3 + a^t x + b^t})$;
21: **else**
22: $G^t := (0, +\sqrt{b^t})$;
23: $x := 1$;
24: **while** $\left(\dfrac{x^3 + ax + b}{p}\right) = -1$ **do**
25: $x := x + 1$;
26: **end while**
27: $G := (x, +\sqrt{x^3 + ax + b})$;
28: **end if**
29: $P := ProdEscalar(p, a, b, G, o_1)$;
30: **if** $(P = \infty)$ **then**
31: $n := o_1$;
32: $n^t := o_2$;
33: **else**
34: $n := o2$;
35: $n^t := o1$;
36: **end if**
37: **return** $\langle p, a, b, a^t, b^t, n, n^t, G, G^t \rangle$;

5.1.4 Um exemplo do algoritmo

A fim de clarificar o algoritmo, será fornecido um exemplo completo da obtenção dos parâmetros de um *twisted pair*. No exemplo será utilizado somente o discriminante $D = -43$. Para este valor de D os polinômios de busca são:

$$p(x) = x^4 - 2x^3 + 2x^2 - x + 11 \quad (5.14)$$
$$n(x) = x^4 - 2x^3 + 4x^2 - 3x + 13 \quad (5.15)$$
$$n'(x) = x^4 - 2x^3 + x + 11 \quad (5.16)$$

Uma rápida busca encontra $x = 332$, que torna $p(x) = 12076361567$, $n(x) = 12076581353$ e $n'(x) = 12076141783$ simultaneamente primos. Além disso, $12076361567 \equiv 3 \mod 4$. Utilizando-se o algoritmo da seção anterior, pode-se calcular os parâmetros que faltam. A j-invariante das curvas é dada pela raiz do polinômio de Hilbert para o respectivo valor de D, e portanto é dado por

$$j = 0 - 884736000 \mod 12076361567 = 11191625567.$$

Com o valor j, a constante intermediária c é calculada como

$$c = \frac{11191625567}{1728 - 11191625567} \mod 12076361567 = 6691706436$$

e as curvas serão

$$\mathcal{E}(\mathbb{F}_{12076361567}) : y^2 = x^3 + 7998757741x + 1307051305 \quad (5.17)$$
$$\mathcal{E}^t(\mathbb{F}_{12076361567}) : y^2 = x^3 + 7998757741x + 10769310262 \quad (5.18)$$

Obtidas as equações, resta obter os pontos geradores e determinar a ordem de cada curva. Inicialmente, verifica-se que

$$\left(\frac{1307051305}{12076361567}\right) = -1$$

e, portanto, o ponto com coordenada $x = 0$ está na curva $\mathcal{E}^t$, e é dado por

$$G^t = (0, +\sqrt{10769310262} \mod 12076361567) = (0, 4543926548).$$

A busca pelo outro gerador rapidamente verifica que a coordenada $x = 1$ está na curva $\mathcal{E}$ e, portanto,

$$G = (1, +\sqrt{1 + 7998757741 + 1307051305} \mod 12076361567) = (1, 1745803925).$$

Finalmente, pode-se utilizar o produto escalar para se determinar a ordem de cada curva. Utilizando-se a curva $\mathcal{E}$ e o valor de $n'(x)$ na função `ProdEscalar(p, a, b, G, n)` como definida anteriormente, obtém-se

$$ProdEscalar(12076361567, 7998757741, 1307051305, (1, 1745803925), 12076141783) = \infty.$$

Portanto,

$$n = n'(x) = 12076141783 \quad (5.19)$$
$$n^t = n(x) = 12076581353. \quad (5.20)$$

A quantidade de informação representada pelos valores x e D é bastante significativa. Como cada elemento (e inclusive cada coordenada dos pontos geradores) são elementos de $\mathbb{F}_p$, o número de bits utilizados para representar todas as informações é

$$\langle p, a, b, a^t, b^t, n, n^t, G, G^t \rangle = 11 \log p \text{ bits}$$

Neste exemplo específico, p possui 34 bits, o que significa um total de 374 bits. Toda a informação é univocamente representada pelo valor 332, com 9 bits, junto com o valor $D = -43$ (que sempre será negativo e portanto pode ser representado por $|D|$) que possui 6 bits. A razão de compressão, neste caso é

$$\frac{15 \text{ bits}}{374 \text{ bits}} \cong 0,04,$$

significando um fator de compressão em torno de 96%.

O exemplo em questão é puramente didático, pois uma curva elíptica sobre um corpo com ordem de 34 bits é considerada insegura nos padrões atuais. Um exemplo mais real é dado pelos valores $x = 1099511695761$ e $D = -43$. Neste caso,

$$p(x) = 146150199879853916111270831239682865870708736297 1 = 161 \text{ bits}$$

representando um nível de compressão próximo a 97% para todos os parâmetros. Este valor foi encontrado após 10 segundos de procura em um computador Athlon 1.3GHz. No mesmo computador, alguns experimentos empíricos demonstram que, em média, se encontra um par de curvas adequadas com este nível de segurança a cada 15 segundos de processamento. Para curvas sobre corpos de 200 bits, o tempo sobe para em torno de 18 segundos por curva, demonstrando que o método é muito eficaz na prática.

De fato, a tendência é que quanto maiores forem os parâmetros necessários, maior seja a compressão dos mesmos, dado que

$$x \cong \sqrt[4]{p},$$

relação que pode ser utilizada para guiar a própria busca pelas curvas. Isto é, para se obter um corpo com ordem em torno de n bits, deve-se testar valores x com $\frac{n}{4}$ bits.

5.1.5 Chaves pública e privada

As seções anteriores discutiram os procedimentos para se obter os parâmetros públicos, que podem ser considerados como parte da chave pública ou não. Entretanto, o cálculo do par de chaves de fato requer um último passo.

Sejam os parâmetros públicos a tupla indicada na seção anterior

$$\langle p, a, b, a^t, b^t, n, n^t, G, G^t \rangle.$$

A chave privada consiste de dois números s e s^t que devem ser escolhidos aleatoriamente dentre os intervalos

$$0 < s < n$$

e

$$0 < s^t < n^t.$$

Já a chave pública são dois pontos P e P^t, respectivamente na curva principal e no *twist* computados através do produto escalar de cada uma das chaves secretas pelo ponto gerador de cada uma das curvas elípticas. Mais especificamente, os pontos

$$P = ProdEscalar(p, a, b, G, s)$$

e

$$P^t = ProdEscalar(p, a^t, b^t, G^t, s^t).$$

A dificuldade de se calcular logaritmos no grupo de pontos das curvas garante que é impraticável calcula-se s a partir de P e G e s^t a partir de P^t e G^t. A chave pública completa, utilizando-se a compressão proposta, é representada pela tupla

$$\langle x, D, P, P^t \rangle. \tag{5.21}$$

5.2 Cifragem

Esta seção apresentará o algoritmo de cifragem, que talvez seja a parte mais importante do presente trabalho. Assim como o esquema probabilístico Blum-Goldwasser é baseado no gerador de bits pseudo-aleatórios criptograficamente seguro Blum-Blum-Schub (como apresentado na seção 2.4.2), a cifragem no algoritmo proposto utiliza o gerador de bits seguros baseado em curvas elípticas de Burton Kaliski, como descrito no capítulo 4. O texto cifrado é gerado a partir da operação de ou-exclusivo bit a bit entre a saída do gerador e o texto claro.

Como referência para o algoritmo gerador será utilizado o algoritmo 4, que possui a chamada de execução

$$Kaliski\_CSPRBG(p, a, b, \beta, o, G, G^t, s, l)$$

e cuja descrição pode ser revista na seção 4.3. Note que o não-resíduo quadrático β, no contexto apresentado na dissertação, será sempre $p - 1$, dado que $p \equiv 3 \mod 4$ foi considerado como restrição na determinação dos parâmetros públicos. Os parâmetros p, a, b, o e l são sempre os mesmos para a mesma chave pública, enquanto os parâmetros G, G^t e s serão sempre diferentes para qualquer operação de cifragem.

Necessariamente, a cifragem deve depender somente da chave pública descrita pela equação 5.21. Simplificadamente, a cifragem é feita calculando-se dois pontos, T e T^t, um sobre cada uma das curvas do *twisted pair* que é a chave pública, e mais um valor r. Como as curvas possuem ordem prima, necessariamente os pontos T e T^t serão geradores das curvas. Esses pontos são então utilizados no gerador de Kaliski, junto com a semente r para gerar uma seqüência de bits pseudo-aleatória segura. Se o gerador é seguro, então para mostrar que o algoritmo é seguro basta mostrar que os parâmetros utilizados no gerador não são dedutíveis a partir do texto cifrado.

A fim de facilitar o entendimento, será apresentado primeiro o algoritmo de cifragem e em seguida, uma explicação linha a linha do mesmo será apresentada. Para tanto, considere o algoritmo 6, que recebe como entrada uma chave pública como descrita anteriormente e uma mensagem $\mathcal{M}$ e retorna o texto cifrado $\mathcal{C}$ e um ponto W. Ele utiliza, também, a função χ definida pela equação 4.5, o algoritmo de descompressão `ParametrosTwistedPair(x, D)` e a função `Random(min, max)` que retorna um valor inteiro realmente aleatório no intervalo (min, max), escolhido utilizando-se uma distribuição uniforme entre todos os valores possíveis. É importante que o valor retornado por essa função seja realmente aleatório, preferencialmente criado utilizando-se algum método de entropia externo.

Passa-se, então, à análise linha à linha do algoritmo 6.

- **Passo 1.** A simples computação dos parâmetros públicos a partir dos valores comprimidos, como explicado anteriormente.

Algoritmo 6 $Cifragem(x, D, P, P^t, \mathcal{M})$

1: $\langle p, a, b, a^t, b^t, n, n^t, G, G^t\rangle := ParametrosTwistedPair(x, D)$;
2: $j := Random(0, 2p + 2)$;
3: **if** $(j < n)$ **then**
4: $\quad W := ProdEscalar(p, a, b, G, j)$;
5: $\quad T := ProdEscalar(p, a, b, P, j)$;
6: $\quad k := \left\lfloor \frac{\chi(T) * n^t}{2p+1} \right\rfloor$;
7: $\quad T^t := ProdEscalar(p, a^t, b^t, P^t, k)$;
8: $\quad r := \chi(T^t)$;
9: **else**
10: $\quad W := ProdEscalar(p, a^t, b^t, G^t, j - n)$;
11: $\quad T^t := ProdEscalar(p, a^t, b^t, P^t, j - n)$;
12: $\quad k := \left\lfloor \frac{\chi(T^t) * n}{2p+1} \right\rfloor$;
13: $\quad T := ProdEscalar(p, a, b, P, k)$;
14: $\quad r := \chi(T)$;
15: **end if**
16: $l := TamanhoEmBits(M)$;
17: $\mathcal{C}' := Kaliski\_CSPRBG(p, a, b, p - 1, n, T, T^t, r, l)$;
18: $\mathcal{C} := \mathcal{C}'$ xor $\mathcal{M}$;
19: **return** $\langle \mathcal{C}, W\rangle$;

- **Passo 2.** A variável j se torna um valor aleatório no intervalo fechado $[1, 2p + 1]$. Para valores práticos de p, a probabilidade de que duas instâncias de cifragem utilizem o mesmo valor j é desprezível.

- **Passo 3.** Testa se j é maior ou menor que a ordem da curva $\mathcal{E}$, dada pelo valor n. Caso seja, são executados os passos 4 a 8, caso contrário são executados os passos 10 a 14.

- **Passo 4.** Nesta linha é calculado o ponto W, que se encontrará sobre a curva $\mathcal{E}$, e será o produto escalar de j pelo gerador da curva G. Como j é um valor uniformemente escolhido e menor que n, então o ponto W pode ser qualquer ponto da curva. De fato, dado que G é um gerador e j pode assumir qualquer valor entre 1 e n, então

$$\{jG | 0 < j < n\} = \mathcal{E} - \{\infty\}.$$

- **Passo 5.** O ponto T calculado neste passo é dado por

$$T = jP$$

onde P é a chave pública sobre a curva $\mathcal{E}$. Como P é um gerador (como qualquer outro ponto), então T pode ser qualquer ponto da curva, da mesma forma que W na análise do passo 4.

- **Passos 6, 7.** No próximo capítulo, quando for feita a análise de segurança do algoritmo, será apresentada uma análise mais profunda desta expressão. Por enquanto,

assuma que a expressão

$$0 < k = \left\lfloor \frac{\chi(T) * n^t}{2p+1} \right\rfloor < n^t$$

é um valor uniforme no intervalo $(0, n^t)$. Isso significa que, na linha 7, T^t é um ponto de $\mathcal{E}^t$ com probabilidade uniforme pois, se k é uniforme, então $T^t = kP^t$ também é.

- **Passo 8.** Neste passo, calcula-se o valor r que é utilizado como semente para o gerador de Kaliski. Como também será explicado no próximo capítulo, como r resulta da aplicação da função χ sobre o ponto T^t, e portanto será um valor uniforme no intervalo $[1, 2p + 1]$.

- **Passos 10-14.** Os passos 10 a 14 basicamente repetem os passos 4 a 8, mas com as seguintes diferenças. Agora, sabe-se que $j \geq n$. Como $j < 2p + 2$, então necessariamente

  $$j < n + n^t \Longrightarrow j - n < n^t.$$

  Como j é escolhido uniformemente, então $j - n$ é um valor uniforme menor que a ordem da curva $\mathcal{E}^t$. Os passos 10 e 11 repetem os passos 4 e 5, calculando W e T^t sobre $\mathcal{E}^t$ uniformemente. O passo 12 gera um valor uniforme no intervalo $(0, n)$ e o 13 gera o ponto T sobre a curva $\mathcal{E}$ a partir de k. Finalmente, no passo 14, a semente r é gerada a partir de T .

- **Passo 16.** A variável l recebe o tamanho, em bits, da mensagem $\mathcal{M}$.

- **Passo 17.** É armazenado em $\mathcal{C}'$ a seqüência de bits pseudo-aleatórios de tamanho l gerada utilizando-se o gerador de Kaliki com parâmetros dados pelas curvas, pelos pontos geradores T e T^t e pela semente r.

- **Passo 18.** É executada a operação de ou-exclusivo entre a mensagem $\mathcal{M}$ e os bits pseudo-aleatórios $\mathcal{C}$ calculados no passo anterior, gerando o texto cifrado $\mathcal{C}$.

- **Passo 19.** O algoritmo encerra retornando o ponto W e o texto cifrado $\mathcal{C}$.

Pela descrição do algoritmo de cifragem, verifica-se que, para cada valor j uniformemente selecionado a cada cifragem, uma *tupla de cifragem* $\langle T, T^t, r\rangle$ diferente é utilizada. Como será visto no próximo capítulo, o fato de se utilizar pontos geradores diferentes para cada curva significa que o algoritmo `Kaliski_CSPRBG` é um algoritmo completamente diferente, mesmo que fosse utilizada a mesma semente r. Isso garante que, na prática não ocorrerão duas cifragens iguais.

5.3 Decifragem

O algoritmo de decifragem é um algoritmo que recebe como parâmetros as chaves privada e pública do destinatário, o ponto W gerado pelo algoritmo de cifragem e o texto cifrado $\mathcal{C}$ correspondente. O algoritmo, então, retorna a mensagem $\mathcal{M}$ originalmente cifrada. A primeira coisa a ser considerada é o fato de que o ponto W pode estar tanto na curva $\mathcal{E}$ quanto na curva $\mathcal{E}^t$, o que depende unicamente do valor j utilizado na cifragem e que é imprevisível, por definição. A determinação de em que curva W se encontra é feita utilizando-se as propriedades dos *twists*.

Seja W_x a coordenada x do ponto W e a e b os coeficientes da curva $\mathcal{E}$. Então,

$$W \in \mathcal{E} \longleftrightarrow \left(\frac{W_x^3 + aW_x + b}{p}\right) = 1,$$

ou seja, o ponto estará nesta curva se e somente se a expressão $W_x^3 + aW_x + b$ for um resíduo quadrático módulo p (e, portanto, o símbolo Jacobi relativo a p for 1). Note que desde que $b \neq 0$, o que no método de cálculo de parâmetros nunca acontece, então não existem pontos com coordenada x compartilhados entre as curvas. Neste caso, necessariamente, a coordenada W_x só tornará a expressão um resíduo quadrático para uma das curvas.

Determinada a origem do ponto W, a decifragem prossegue da seguinte forma. Considere que $W \in \mathcal{E}$. Isso significa que, no algoritmo de cifragem o valor j era menor que a ordem da $\mathcal{E}$ e, portanto, a expressão

$$T = jP = sjG = jsG = sW$$

é verdadeira. Neste caso, para obter o ponto T, basta calcular o produto escalar da chave privada s pelo ponto M sobre a curva $\mathcal{E}$. A partir daí a decifragem segue exatamente como a cifragem, dado que os parâmetros T^t e r utilizados no gerador dependem somente de T e a operação de ou-exclusivo no fim da algoritmo é simétrica.

A mesma análise é válida caso se tenha que $W \in \mathcal{E}^t$. isso significa que no algoritmo de cifragem, $j > n$ e, consequentemente,

$$T^t = (j-n)P^t = s^t(j-n)G^t = (j-n)s^tG^t = s^tW.$$

Aqui, a diferença é que a chave a ser utilizada é s^t, mas o algoritmo funciona exatamente da mesma forma. O algoritmo 7 é a formalização do procedimento de decifragem, que implementa toda a análise feita.

Algoritmo 7 $Decifragem(x, D, P, P^t, s, s^t, \mathcal{C}, W)$

1: $\langle p, a, b, a^t, b^t, n, n^t, G, G^t \rangle := ParametrosTwistedPair(x, D)$;
2: **if** $\left(\frac{W_x^3 + aW_x + b}{p}\right) = 1$ **then**
3: $\quad T := ProdEscalar(p, a, b, W, s)$;
4: $\quad k := \left\lfloor \frac{\chi(T) * n^t}{2p+1} \right\rfloor$;
5: $\quad T^t := ProdEscalar(p, a^t, b^t, P^t, k)$;
6: $\quad r := \chi(T^t)$;
7: **else**
8: $\quad T^t := ProdEscalar(p, a^t, b^t, P^t, s_t)$;
9: $\quad k := \left\lfloor \frac{\chi(T^t) * n}{2p+1} \right\rfloor$;
10: $\quad T := ProdEscalar(p, a, b, P, k)$;
11: $\quad r := \chi(T)$;
12: **end if**
13: $l := TamanhoEmBits(C)$;
14: $\mathcal{M}' := Kaliski\_CSPRBG(p, a, b, p-1, n, T, T^t, r, l)$;
15: $\mathcal{M} := \mathcal{C}$ xor $\mathcal{M}'$;
16: **return** $\langle \mathcal{M} \rangle$;

6 ANÁLISE DE SEGURANÇA DO ALGORITMO

Neste capítulo, será apresentada a análise de segurança do algoritmo proposto. Existem duas questões a serem abordadas, que serão examinadas individualmente.

A primeira questão diz respeito à segurança de uma instância de cifragem. A abordagem utilizada para mostrar que o algoritmo é seguro será a redução do problema de se obter alguma vantagem para a obtenção do texto claro a partir do texto cifrado para o problema de se calcular o logaritmo discreto sobre o grupo formado pelo conjunto de pontos de uma curva elíptica de ordem prima.

A segunda questão diz respeito à segurança semântica do algoritmo. Relativamente a um adversário polinomial, mostrar que é impossível recuperar o texto claro e mostrar que é impossível obter *qualquer* informação parcial sobre o texto claro são provas com requisitos distintos. Para se provar que o algoritmo é semanticamente seguro, será utilizada a abordagem de demonstrar que o algoritmo apresenta as características da definição 2.20 onde nenhum adversário polinomial consegue distinguir duas instâncias de cifragem entre si.

As próximas seções apresentam estas discussões e fornecem as provas de que o algoritmo possui as características apontadas. A última seção deste capítulo apresenta as características de um novo cenário de segurança e estende o algoritmo para que ele seja seguro neste cenário. Para tanto é apresentada uma nova função de resumo criptográfico (*hash*) cuja segurança também é redutível ao ECDLP assim como todo o resto do algoritmo.

6.1 Segurança de uma instância de cifragem

Esta seção se dedicará à segurança de uma instância de cifragem pelo algoritmo, isto é, dados um texto cifrado e a chave pública utilizada, pretende-se demonstrar que recuperar o texto claro significa resolver o ECDLP. Inicialmente, revisita-se as premissas de segurança que são assumidas neste trabalho. A primeira definição formal a ser feita é a do próprio ECDLP, mas no contexto do algoritmo proposto.

Definição 6.1 *Sejam P e $Q \neq \infty$ dois pontos da curva elíptica $\mathcal{E}(\mathbb{F}_p)$ com ordem prima n. O ECDLP é encontrar o único valor escalar $r \leq n$ que torna a equação*

$$P = rQ$$

verdadeira.

Sobre o ECDLP é feita a seguinte conjectura de segurança.

Conjectura 6.2 *O ECDLP é insolúvel por qualquer adversário polinomial.*

É necessário definir também um outro problema (já apresentado informalmente) chamado *Diffie-Hellman* (em homenagem aos seus criadores Whitfield Diffie e Martin Hellman). O problema Diffie-Hellman tradicional foi originalmente definido sobre o grupo multiplicativo $\mathbb{F}_p^*$. Porém, por ser baseado em uma estrutura de grupo, ele pode ser facilmente transposto para o grupo de pontos de uma curva elíptica, sendo neste caso chamado de *Elliptic Curve Diffie-Hellman Problem*, ou ECDHP. É importante não confundir o protocolo ECDH apresentado no capítulo 3 com o problema ECDHP, apesar do fato de que a segurança do protocolo é baseada na intratabilidade do problema.

Definição 6.3 *Sejam P e aP e bP três pontos da curva elíptica $\mathcal{E}$ com ordem prima n tal que os valores a e b não sejam conhecidos. O ECDHP é calcular o ponto abP sobre $\mathcal{E}$.*

É trivial visualizar que um algoritmo que solucione o ECDLP pode ser utilizado para solucionar o ECDHP. Bastaria calcula-se a a partir de aP (o que é o ECDLP) e calcular o produto escalar de a sobre o ponto bP, obtendo como resultado o ponto abP. Este procedimento prova a redução

$$ECDHP \Longrightarrow ECDLP$$

e significa que o ECDHP é, no máximo, tão difícil de se solucionar quanto o ECDLP.

Um problema em aberto até hoje é a prova da redução oposta. Isto é, como utilizar um algoritmo para o ECDHP de forma a se solucionar o ECDLP. Em (MAURER; WOLF, 1996) é apresentada uma discussão extensa sobre o assunto, considerando-se exclusivamente grupos multiplicativos. O fato é que, apesar de esta proposição não estar provada, a seguinte conjectura é aceita, sem restrições, pela comunidade acadêmica.

Conjectura 6.4 *O ECDLP e o ECDHP são problemas equivalentes.*

A proposição 6.4 implica que qualquer solução polinomial para o ECDHP é também uma solução para o ECDLP. Isso significa que a proposição 6.2 se aplica também ao ECDHP indistintamente.

Também é necessário se considerar a segurança do gerador de bits pseudo-aleatórios desenvolvido por Kaliski. Porém, a prova de que o gerador é seguro é extensa demais para ser explicitada nesta dissertação. Para uma apresentação detalhada desta prova, é sugerida a análise da tese de Kaliski (KALISKI, 1988). Para fins desta dissertação, será considerado como verdadeiro o seguinte teorema.

Teorema 6.5 *Seja ${\mathcal{T}'}_{a,b,\beta}(\mathbb{F}_p)$ um twisted pair de ordem prima. Sejam os pontos G e G^t pontos uniformemente escolhidos dentre os pontos das curvas do twisted pair e seja $s \in [0, 2p+1]$ um valor uniformemente escolhido. Então qualquer vantagem de distinguir a saída do algoritmo*

$$Kaliski\_CSPRBG(p, a, b, \beta, n, G, G^t, s, l),$$

de uma seqüência de bits aleatória de comprimento l implica a mesma vantagem para solucionar o $ECDLP$.

A análise do algoritmo propriamente dita, pode ser dividida em duas partes, que correspondem respectivamente a cada um dos resultados da comparação da linha 3 no algoritmo de cifragem 6. Como sugerido no algoritmo de decifragem, é trivial descobrir, a partir do texto cifrado (que é composto por $\langle \mathcal{C}, W \rangle$), qual foi o resultado da comparação na execução da cifragem. Pode-se então separar a análise nos casos onde a comparação é verdadeira e falsa, mais especificamente, se o valor j escolhido é maior ou menor que n.

Consideremos, inicialmente, $j < n$. Dado que o texto cifrado é gerado através da operação de ou-exclusivo com a saída do gerador pseudo-aleatório, descobrir qualquer padrão no texto cifrado $\mathcal{C}$ é equivalente a solucionar o ECDLP. Isso significa que, qualquer vantagem para se reverter o algoritmo de cifragem implica na aproximação da tupla $\langle T, T^t, r \rangle$ que foi utilizada como semente para gerar a seqüência.

Se $j < n$, então, o algoritmo é revertido se o ponto T é obtido. Considere, então, os seguintes lemas.

Lema 6.6 *Seja $j \in [0, 2p+1]$ e $s \in [0, n-1]$ a chave privada relativa à curva $\mathcal{E}$. Se j e s são uniformemente selecionados e $j < n$, então a probabilidade de que W e T sejam o mesmo ponto é desprezível.*

Prova Por definição, todos os pontos são geradores e n é a ordem de qualquer ponto diferente ∞. O ponto W é

$$W = jG$$

e o ponto T é

$$T = jP = sjG,$$

e isso significa que $W = T$ se e somente se

$$j \equiv js \mod n.$$

Como ambos j e s são menores que n, e n é primo, esta congruência é verdadeira somente quando $s = 1$ ou $j = 0$ (o que não ocorre), então, a probabilidade de que $W = T$ é

$$\frac{1}{n} = \frac{1}{p + 1 - 2\sqrt{p}} = \Omega(p^{-1}),$$

que decresce mais rápido que qualquer polinômio em $\log p$. □

O lema 6.6 garante que o texto cifrado é decifrado sem a chave secreta somente em um número desprezível de casos. De fato, essa possibilidade pode ser excluída completamente modificando-se o algoritmo para garantir que s (e também s^t) seja diferente de 1.

Para continuar, considere o próximo lema.

Lema 6.7 *Seja n um valor primo, e dois valores $0 < i_1 < n$ e $0 < i_2 < n$. Então a congruência*

$$i_1 i_2 \mod n$$

apresenta uma distribuição uniforme para todos os valores entre 1 e $n - 1$.

Prova A prova é por indução. Claramente, se fixarmos um dos valores em 1, a distribuição do resultado será uniforme entre 1 e $n-1$. Considere valor $k = i_1$ fixo, e varie o i_2 entre 1 e $n-1$. Se

$$k \Longrightarrow (k, 2k, 3k, \ldots, (n-1)k) \mod n$$

resulta distintamente em todos os valores entre 1 e $n-1$ então, para $k+1$ se tem

$$(k+1) \Longrightarrow (k+1, 2(k+1), 3(k+1), \ldots, (n-1)(k+1)) \mod n$$
$$(k+1) \Longrightarrow (k+1, 2k+2, 3k+3, \ldots, (n-1)k + (n-1)) \mod n,$$

o que é simplesmente os próprios valores $(k, 2k, 3k, \ldots, (n-1)k)$ acrescidos de $1, 2, \ldots, n-1$. Assim, o valor que era 1 passa a ser 2, o que era 2, passa a ser 3 e assim sucessivamente. Portanto, esta continuará sendo uma distribuição uniforme entre os valores 1 e $n-1$. Como para todos os valores de i_1 ela é uniforme, então toda a distribuição é uniforme. □

Lema 6.8 *T é um ponto qualquer da curva $\mathcal{E}$ com a mesma probabilidade para cada ponto.*

Prova O índice de T na base G é dado pela congruência

$$sj \mod n.$$

Se j e s são valores uniformemente escolhidos e menores que n, então, pelo lema 6.7 o índice de T será um valor uniforme no intervalo $[1, n-1]$ o que resulta em qualquer ponto de $\mathcal{E}$ com distribuição uniforme. □

O lema 6.8 implica que, como T é qualquer ponto, então não existe nenhuma forma de distinguir T do conjunto de pontos da curva.

Lema 6.9 *Seja $\mathcal{P} = \{\chi(P) | \forall P \in \mathcal{E}\}$ o conjunto de valores geradores pela função χ quando aplicados sobre os pontos de $\mathcal{E}$ e seja $\mathcal{A} = \{\chi(P) | \forall P \in \mathcal{E} \cup \mathcal{E}^t\}$. Então, a distribuição dos valores do conjunto $\mathcal{P}$ é uniforme sobre o conjunto $[0, 2p+1]$ com alta probabilidade.*

Prova Por definição, $\mathcal{A} = [0, 2p+1]$. A pior distribuição ocorre quando $\mathcal{E}$ é a curva que possui o menor número de pontos, o que representa, no mínimo, $p + 1 - 2\sqrt{p}$ pontos. O primeiro fato a ser considerada é que, mesmo nesse caso, o número de valores que os pontos da curva representam é aproximadamente metade dos valores do intervalo, ou seja,

$$\frac{|\mathcal{P}|}{|\mathcal{A}|} = \frac{p + 1 - 2\sqrt{p}}{2p+2} \cong \frac{1}{2}.$$

Em (PERALTA, 1992), René Peralta demonstra que a distribuição de resíduos quadráticos módulo um número primo p é praticamente uniforme dentre todos os valores menores que p, ou seja, a cada dois valores menores que p quaisquer, em média, um deles será um resíduo quadrático e o outro não. Isso significa que, de cada dois valores $x \in [0, p-1]$ um deles tornará a expressão $x^3 + ax + b \mod p$ um resíduo quadrático e isso implica que as coordenadas x dos pontos da curva $\mathcal{E}$ estão uniformemente distribuídas entre os valores do intervalo $[0, p-1]$.

Relativamente a $\mathcal{E}$, o valor da função χ é $2x$ e $2x+1$ para os pontos $(x, \pm y) \in \mathcal{E}$. Como os valores x são uniformemente distribuídos sobre $[0, p-1]$ então os valores $2x$ e $2x+1$, mais o ponto no infinito que representa o limite superior do intervalo, estão uniformemente distribuídos no intervalo $[0, 2p+1]$. □

Lema 6.10 *Se a chave secreta $0 < s < n$ é uniformemente escolhida, então o ponto T^t é um ponto qualquer da curva $\mathcal{E}^t$ com distribuição quase uniforme para todos os pontos.*

Prova Pelo lema 6.8, o ponto T é um ponto qualquer da curva $\mathcal{E}$. Portanto, os possíveis valores de $\chi(T)$ são os que os pontos de $\mathcal{E}$ geram. Pelo lema 6.9 estes valores estão uniformemente distribuídos no intervalo $[0, 2p + 1]$ com alta probabilidade. Considere, então, a expressão

$$k = \left(\left\lfloor \frac{\chi(T) * n^t}{2p + 1} \right\rfloor \right).$$

Esta expressão multiplica a razão entre $\frac{\chi(T)}{2p+1}$, que será um valor uniforme no intervalo contínuo entre $[0, 1]$, pela ordem da curva $\mathcal{E}^t$. Assim, o resultado da expressão é um valor uniforme no intervalo $[0, n^t - 1]$ com alta probabilidade. O erro desta distribuição corresponde à diferença entre os traços de Frobenius das duas curvas que é, no máximo, $n - n^t = 4\sqrt{p}$, e, portanto, desprezível.

O índice de T^t na base G^t será dado pela congruência $ks^t \mod n^t$. Como k e s^t são uniformes no intervalo $[0, n^t - 1]$, então, pelo mesmo argumento apresentado no lema 6.8, T^t é um ponto qualquer da curva $\mathcal{E}^t$ com distribuição uniforme.

Utilizando os lemas apresentados, é possível provar a segurança do algoritmo de cifragem para o caso em que $j < n$, isto é, calcular a tupla $\langle T, T^t, r\rangle$ é equivalente a solucionar o ECDLP. Isto é demonstrado no seguinte teorema.

Teorema 6.11 *Se $j < n$, $0 < s < n$ and $0 < s^t < n^t$ são uniformemente escolhidos, então, obter a tupla $\langle T, T^t, r\rangle$ a partir de um texto cifrado $\langle \mathcal{C}, W\rangle$ é equivalente a solucionar o ECDLP.*

Prova Devido a primalidade das curvas e ao fato dos valores j e s serem uniformes, W e P são pontos quaisquer da curva $\mathcal{E}$. Então, obter T a partir de W e P é o mesmo que solucionar o ECDHP, que é equivalente ao ECDLP pela proposição 6.4.

Como T^t é um ponto qualquer da curva $\mathcal{E}^t$ com probabilidade praticamente uniforme, então, pelo mesmo argumento do lema 6.9 os valores da semente $r = \chi(T^t)$ estão uniformemente distribuídos no intervalo $[0, 2p + 1]$. Dado que $r = \chi(T^t)$, qualquer vantagem de se calcular r implica a mesma vantagem de obter o ponto T^t. O índice de T^t na base G^t é dado pela congruência $ks^t \mod n^t$. Como k é uniforme em $[0, n^t - 1]$, conforme a prova do lema 6.10, então aproximar o valor de $ks^t \mod n^t$ implica aproximar o valor k. O valor k depende unicamente de T, e, por conseqüência, aproximar k é tão difícil quanto obter T, o que já foi estabelecido ser tão difícil quanto o ECDLP. Portanto, calcular cada um dos valores da tupla $\langle T, T^t, r\rangle$ apresenta a mesma dificuldade de se solucionar o ECDLP. □

O teorema 6.11 é um dos dois principais resultado acerca da segurança do algoritmo. O segundo resultado é referente à segurança semântica, e será apresentado na próxima seção. Note que todas as provas presumem $j < n$. Mesmo assim, o caso $j \geq n$ é análogo e as provas são praticamente as mesmas, com a seguinte diferença: agora, $j - n$ é um valor uniforme no intervalo $[0, n^t]$. Isso significa que W e T^t são pontos uniformemente distribuídos em $\mathcal{E}^t$ e assim por diante. Portanto, exatamente da mesma forma, se $j \geq n$, então obter a tupla $\langle T, T^t, r\rangle$ é tão difícil quanto solucionar o ECDLP.

6.2 Segurança semântica

Nesta seção será demonstrado que o esquema possui segurança semântica. Como já apresentado, isso pode ser feito demonstrando que quaisquer duas instâncias de cifragem são indistinguíveis entre si por um adversário polinomial. No algoritmo proposto, isto só é possível devido ao elemento aleatório que a variável j introduz à cifragem. Inicialmente considere o seguinte lema que diz respeito à geradores de grupos.

Lema 6.12 *Sejam G_1 e G_2 dois geradores distintos de um grupo $\mathcal{G}$ com ordem prima n. Então os pontos das seqüências*

$$(G_1, 2G_1, 3G_1, \ldots, (n-1)G_1)$$

e

$$(G_2, 2G_2, 3G_2, \ldots, (n-1)G_2)$$

são diferentes para todos os índices menores que n, ou seja,

$$(G_1 \neq G_2, 2G_1 \neq 2G_2, 3G_1 \neq 3G_2, \ldots, (n-1)G_1 \neq (n-1)G_2).$$

Prova O índice 1 das seqüências é diferente, pois, por definição $G_1 \neq G_2$. Para visualizar porque o lema é verdadeiro sempre que n é primo, considere que o conjunto $\mathcal{G} = \{A, B, C, D, \infty\}$ forme um grupo de ordem 5. Suponha que a seqüência de índices do elemento A seja

$$(A, 2A, 3A, 4A, 5A) = (A, B, C, D, \infty).$$

Isso significa que $B = 2A$. Portanto, a seqüência de índices gerada por B pode ser dada por

$$\begin{aligned}(B, 2B, 3B, 4B, 5B) &= ((2A), 2(2A), 3(2A), 4(2A), 5(2A)) \\ &= (2A, 4A, 6A, 8A, 10A) \\ &= (2A, 4A, 1A, 3A, 5A) \\ &= (B, D, A, C, \infty)\end{aligned}$$

e, similarmente, a seqüência do elemento C é

$$\begin{aligned}(C, 2C, 3C, 4C, 5C) &= ((3A), 2(3A), 3(3A), 4(3A), 5(3A)) \\ &= (3A, 1A, 4A, 2A, 5A) \\ &= (C, A, D, B, \infty)\end{aligned}$$

Isso significa que a seqüência de qualquer gerador pode ser dada toda com respeito a um único gerador, como

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{ccccc} \times 1 & \times 2 & \times 3 & \times 4 & \times 5 \end{array} \\ \begin{pmatrix} A & 2A & 3A & 4A & 5A \\ 2A & 2(2A) & 3(2A) & 4(2A) & 5(2A) \\ 3A & 2(3A) & 3(3A) & 4(3A) & 5(3A) \\ 4A & 2(4A) & 3(4A) & 4(4A) & 5(4A) \end{pmatrix} \end{array}$$

sendo a última coluna completamente preenchida com o elemento identidade, pois é multiplicado pela ordem do grupo.

Como regra geral, cada elemento pode ser definido em função de um elemento gerador A. Considera-se uma matriz $n \times n$ e faz-se a primeira linha dela igual à seqüência de pontos gerados pelos índices de A. A segunda linha começa com o elemento correspondente ao segundo índice de A e assim sucessivamente. Ao se preencher esta matriz, nota-se que as seqüências dos elementos são dadas pelo elemento A com índice igual ao produto da linha i com a coluna j para cada célula, tomada módulo n que é a ordem do grupo. Esta matriz $n \times n$ é explicitada como a seguir:

$$\begin{array}{c} \\ 1 \\ 2 \\ \vdots \\ n-1 \\ n \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{pmatrix} 1A & (1 \times 2 \mod n)A & \dots & (1 \times n \mod n)A \\ 2A & (2 \times 2 \mod n)A & \dots & (2 \times n \mod n)A \\ \vdots & \vdots & \ddots & \vdots \\ (n-1)A & ((n-1) \times 2 \mod n)A & \dots & ((n-1) \times n \mod n)A \\ nA & (n \times 2 \mod n)A & \dots & (n \times n \mod n)A \end{pmatrix} \end{array}$$

Como a ordem do grupo é prima, todos os elementos diferentes da identidade são geradores. Assim, necessariamente, cada uma das linhas desta matriz vai possuir cada um dos elementos de $\mathcal{G}$ exatamente uma vez. Note, também, que cada linha é igual à coluna de mesmo índice transposta, isto é, a linha $i = k$ é idêntica à coluna $j = k$ transposta, para todos os valores $1 \leq k < n$. Como todas as linhas contém todos os n elementos, então, todas as colunas também possuem todos os elementos diferentes da identidade. O fato de que nenhuma coluna repete nenhum elemento, implica, então, que dois geradores distintos, quando multiplicados pelo mesmo índice menor que n, nunca geram como resultado o mesmo elemento. □

Utilizando-se o lema 6.12, é possível provar-se a segurança semântica do algoritmo proposto, como demonstrado no seguinte teorema.

Teorema 6.13 *O algoritmo é semanticamente seguro.*

Prova Como cada curva forma um grupo de ordem prima, então as chaves públicas P e P^t são geradores dos grupos de cada curva. Isso significa que cada valor $j \in [0, 2p+1]$ resultará em uma tupla de cifragem distinta $\langle T, T^t, r \rangle$. Pelo lema 6.12, a seqüência gerada por cada gerador é distinta e, para qualquer escalar fixo k o produto escalar kT é um ponto distinto da curva $\mathcal{E}$ para cada ponto T. Isso significa que cada tupla $\langle T, T^t, r \rangle$ instancia um algoritmo gerador de bits pseudo-aleatórios completamente diferente do instanciado por qualquer outra tupla $\langle T', T^{t'}, r' \rangle$, sem deixar de ser criptograficamente seguros. Assim, a probabilidade de que o texto cifrado $\langle \mathcal{C}, W \rangle$ seja o mesmo para uma mesma mensagem $\mathcal{M}$ é

$$\frac{1}{2p+2,}$$

que decresce mais rápido que qualquer polinômio em $\log p$. Portanto, a dificuldade de um adversário polinomial distinguir duas instâncias de cifragem entre si é equivalente à dificuldade de decifrar cada uma das instâncias e comparar os respectivos textos claros entre si. Mas isso é equivalente a solucionar o ECDLP, que é insolúvel por qualquer adversário polinomial. Portanto, nenhum adversário polinomial consegue distinguir duas instâncias de cifragem entre si e o algoritmo é semanticamente seguro. □

6.3 Estendendo o algoritmo: adversários ativos

Até o momento, as provas de segurança do algoritmo foram dadas relativamente a um adversário polinomial que é *passivo*, como um circuito polinomial. Entretanto, o algoritmo claramente não é resistente contra um *adversário ativo*, ou, mais genericamente, contra um *ataque de texto cifrado adaptativo*. Um exemplo claro desta fragilidade aparece no caso em que o atacante conhece o conteúdo do texto cifrado $\mathcal{C}$. Como a operação de ou-exclusivo é linear, é trivial alterar o conteúdo do texto claro correspondente para qualquer outro valor sem dificuldade alguma.

Garantir que o algoritmo seja seguro neste contexto pode, em alguns casos, ser mais importante do que inicialmente pode parecer. Por exemplo, o esquema *Blum-Goldwasser* (apresentado na seção 2.4.2) é tão frágil contra um ataque de texto cifrado adaptativo, que ele pode acabar entregando a própria chave secreta como resultado. Isto acontece devido ao fato de que, se n é um número composto por dois primos congruentes a $3 \mod 4$, então existem exatamente 4 raízes para cada resíduo quadrático, respectivamente $\pm x$ e $\pm y$, e alguma combinação $MDC(\pm x \pm y, n)$ retorna um dos fatores de n.

Neste contexto, considere esta nova definição de segurança:

Definição 6.14 *Seja $P(x)$ um polinômio. Um algoritmo de chave pública é dito* não maleável *se, dado um texto cifrado $\mathcal{C}$ um adversário polinomial não consegue gerar outro texto cifrado válido $\mathcal{C}'$ onde o texto claro correspondente a $\mathcal{C}$ seja relacionado de alguma forma conhecida com o texto claro correspondente ao texto cifrado $\mathcal{C}'$.*

Note a crucial inserção da palavra *válido* no contexto da definição 6.14. Ela sugere algum tipo de verificação intrínseca no algoritmo de forma a se detectar mudanças propositais no texto cifrado, e que um adversário polinomial não consiga subverter. Baseado nesta definição, será proposta uma modificação no algoritmo para que o resultado final seja um algoritmo semanticamente seguro e não maleável.

Considere a seguinte definição de função de *hash* criptograficamente segura.

Definição 6.15 *Uma **função de hash**, ou **resumo criptográfico**, $H : \{0,1\}^* \longrightarrow \{0,1\}^k$ é uma função que mapeia strings binárias de tamanho arbitrário em strings binárias de tamanho específico k. Ela é dita* criptograficamente segura *se*

1. *dada a string m, um adversário polinomial não consegue encontrar uma mensagem m' tal que $H(m) = H(m')$,*
2. *um adversário polinomial não consegue encontrar duas mensagens m_1 e m_2 tal que $H(m_1) = H(m_2)$.*

Devido ao chamado *paradoxo do aniversário* (MENEZES; OORSCHOT; VANSTONE, 1996), a complexidade do problema 1 é naturalmente maior que a complexidade do problema 2. A complexidade esperada do problema 1, chamado de *ataque de pré-imagem* é 2^k, enquanto a do problema 2, chamado de *ataque de colisão*, é $\sqrt{2^k}$. Mesmo assim, ambos são exponenciais e, portanto, insolúveis por um adversário polinomial.

Um dos objetivos colocados como premissa inicial para o trabalho é que o algoritmo fosse dependente somente da segurança provida por curvas elípticas. Portanto, propõe-se uma nova função de hash, baseada na dificuldade de se computar logaritmos no grupo de pontos de uma curva elíptica de ordem prima. A proposta é uma adequação para curvas elípticas da função de *hash* segura Chaum-van Heijst-Pfitzmann (CHAUM; HEIJST; PFITZMANN, 1992) que é baseada na dificuldade de se calcular logaritmos discretos em um grupo multiplicativo $\mathbb{F}_p^*$.

Teorema 6.16 *Sejam P e $Q = sP$ dois pontos da curva elíptica $\mathcal{E}$ com ordem prima n. Encontrar uma colisão na função $\mathcal{H} : [0, n-1] \times [0, n-1] \rightarrow \mathcal{E}$ dada por*

$$\mathcal{H}(x_1, x_2) = x_1 P + x_2 Q$$

é equivalente a calcular o logaritmo discreto de Q na base P.

Prova Considere que se consiga encontrar uma colisão na função, dada por

$$\mathcal{H}(x_1, x_2) = \mathcal{H}(x_3, x_4).$$

Isso significa que

$$x_1 P + x_2 Q = x_3 P + x_4 Q$$

e, portanto

$$\begin{aligned} x_1 P - x_3 P &= x_4 Q - x_2 Q \\ (x_1 - x_3) P &= (x_4 - x_2) Q \end{aligned}$$

Se $(x_1 - x_3)P$ e $(x_4 - x_2)Q$ são o mesmo ponto, então

$$(x_1 - x_3)^{-1} * (x_4 - x_2) * s \mod n = 1$$

e, portanto o logaritmo discreto de Q na base P é dado por,

$$\log_P Q = (x_1 - x_3) * (x_4 - x_2)^{-1} \mod n$$

□

A prova é relativamente simples porque a curva possui ordem prima. Entretanto ela é trivialmente extensível para qualquer curva onde a ordem possui pelo menos um fator primo grande, o que é um pré-requisito para qualquer curva elíptica utilizada para fins criptográficos. Note, também, que a função é resistente a ataques de pré-imagem, como mostra o seguinte teorema.

Teorema 6.17 *Seja $\mathcal{H}$ a função de hash apresentada no teorema 6.16. Se $\mathcal{H}$ é livre de colisões então $\mathcal{H}$ é resistente a ataques de pré-imagem.*

Prova Seja H o hash de algum par de valores (x_1, x_2). Um ataque de pré-imagem é bem sucedido encontrando dois outros valores (x_3, x_4) tal que $\mathcal{H}(x_3, x_4) = H$. Isso significa que

$$\begin{aligned} H = x_1 P + x_2 Q &= x_3 P + x_4 Q = \mathcal{H}(x_3, x_4) \\ x_1 P - x_3 P &= x_4 Q - x_2 Q \\ (x_1 - x_3) P &= (x_4 - x_2) Q \end{aligned}$$

e, conseqüentemente

$$\log_P Q = (x_1 - x_3)(x_4 - x_2)^{-1} \mod n.$$

Dado que o ECDLP é difícil, então, a função é resistente a ataques de pré-imagem. □

Em alguns casos, como a seguir, pode não ser interessante que o resultado da função de hash seja um ponto de uma curva. Portanto, o resultado final da função $\mathcal{H}$ pode ser mapeado em uma seqüência de bits utilizando a função χ. Assim, o resultado final do *hash* ocupa $\lceil \log 2p + 1 \rceil = \log p + 1$ bits.

Um fator importante é a questão de relativo a quais pontos o *hash* é gerado. O *hash* só é seguro quando o logaritmo de Q na base P não é conhecido. Desta forma, no contexto do algoritmo sendo proposto, o *hash* $\mathcal{H}$ pode ser gerado com base na curva $\mathcal{E}$ sobre os pontos G e P que definem a chave pública do destinatário. Já que a chave secreta sobre essa curva, que é o valor s tal que $P = sG$, não é conhecido pelo atacante (caso contrário todo o algoritmo seria inseguro) então o valor do *hash* será seguro. Convenciona-se, então, o cálculo do *hash* como à estes pontos.

A função de *hash* $\mathcal{H}$, como foi definida, mapeia $2 \log n$ bits em um ponto de uma curva elíptica, que é representável em $\log p + 1$ bits. Dado que n é a ordem de uma das curvas, sabe-se que o menor valor possível para n é $n = p + 1 - 2\sqrt{p}$. Isso significa que a maior seqüência binária que não ultrapassa o valor n deve possuir, no máximo, $\log p - 2$ bits. Isso significa que a função faz uma redução, sem colisão de

$$2(\log p - 2) = 2 \log p - 4 \text{ bits } \implies \log p + 1 \text{ bits },$$

presumindo-se que $\log p > 4$. Entretanto, uma função de *hash* genérica deve ser capaz de mapear uma quantidade arbitrária de bits em uma seqüência de tamanho fixo, sem permitir colisões. Para isto é necessário estender a função $\mathcal{H}$ de forma que uma colisão em uma seqüência de aplicações necessariamente implique uma colisão na função base, garantindo, então, que a extensão também é livre de colisões.

Assim, define-se a função estendida $\mathcal{H}^*$, na forma do algoritmo 8, que utiliza a função $\mathcal{H}$. Esta função divide a mensagem $\mathcal{M}$ cujo *hash* será calculado em blocos menores, de exatamente $\log p - 6$ bits, representados seqüencialmente pelos blocos $m_1 \ldots m_k$, onde k é o número total de blocos. O valor d que aparece no algoritmo é a diferença entre o tamanho que cada um dos blocos tem e o tamanho que o último bloco possui, devido a $\mathcal{M}$ não ter um tamanho múltiplo de $\log p - 6$ e, portanto, $d = \log p - 6 - |m_k|$. Já o valor t é o tamanho da saída da função H, que é $\log p + 1$. As expressões 0^d e 0^{t+1} significam simplesmente uma seqüência binária de zeros de tamanho indicado pelos expoentes e o símbolo || é a concatenação de seqüências binárias.

Algoritmo 8 $\mathcal{H}^*(m_1 \ldots m_k, t, d)$

1: **for** $i := 1$ to $k - 1$ **do**
2: $\quad y_i := m_i$;
3: **end for**
4: $y_k := m_k || 0^d$
5: Seja y_{k+1} a representação binária do valor d;
6: $g_1 := \mathcal{H}(0^{t+1} || y_1)$;
7: **for** $i := 1$ to k **do**
8: $\quad g_{i+1} := \mathcal{H}(g_i || 1 || y_{i+1})$;
9: **end for**
10: **return** g_{k+1};

O algoritmo atribui às variáveis y_i o conteúdo dos blocos m_i, onde o último bloco é completado com bits zero até $\log p - 4$ bits. Um bloco extra y_{k+1} é criado a partir da representação binária da diferença d. Após isso, ele procede a uma seqüência recursiva

de chamadas à função $\mathcal{H}$. A cada passo, ele reutiliza como entrada para a função o valor do *hash* da iteração anterior. Note que a função $\mathcal{H}$ é invocada com a concatenação dos parâmetros $g_i||1||y_{i+1}$, o que representa $\log p + 1 + 1 + \log p - 6 = 2\log p - 4$ bits, exatamente o tamanho máximo da entrada da função de compressão $\mathcal{H}$ (isto é, dois valores de $\log p - 2$ bits).

Em última instância, essa recursão na função garante que uma colisão na função $\mathcal{H}^*$ implica necessariamente uma colisão na função $\mathcal{H}$. A prova deste fato é relativamente simples, porém longa e será omitida. Seus detalhes podem ser encontrados em (STINSON, 1995).

Dada a função de hash $\mathcal{H}^*$ como definida no algoritmo 8, pode-se modificar os algoritmos 6 e 7 para que eles apresentem não maleabilidade de acordo com a definição 6.14.

O algoritmo de cifragem sofre as seguintes modificações. Seja $l = TamanhoEmBits(\mathcal{M})$ e seja $k = \lceil \log 2p + 1 \rceil$ o tamanho do *hash* gerado. Calcula-se a tupla de cifragem $\langle T, T^t, r \rangle$ da mesma forma que no algoritmo original. Ao executar o algoritmo gerador de Kaliski, ao invés de l bits, retorna-se $l + 2k$ bits. Seja ϑ igual aos os últimos k bits da seqüência pseudo-aleatória gerada. Calcula-se o *hash* $h = \mathcal{H}^*(\mathcal{M}||\vartheta)$, onde || significa a concatenação de seqüências. Faça $\mathcal{M}' := \mathcal{M}||h$ e faça o ou-exclusivo de $\mathcal{M}'$ com os primeiros $l + k$ bits pseudo-aleatórios retornados, gerando finalmente $\langle \mathcal{C}, W \rangle$.

A decifragem é modificada da seguinte forma. Assim como a cifragem, ela é idêntica até a geração da seqüência pseudo-aleatória, retornando, neste caso, $l + k$ bits (dado que $\mathcal{C}$ já é maior que $\mathcal{M}$ por k bits). Faça ϑ igual aos últimos k bits da seqüência. Faça o ou-exclusivo dos primeiros l bits da seqüência com $\mathcal{C}$ retornando $\mathcal{M}'$ que, na verdade, é $\mathcal{M}||h$. Aceitar o texto cifrado como válido se e somente se $\mathcal{H}^*(\mathcal{M}||\vartheta) = h$.

Como h é um valor de *hash* assinado pela seqüência pseudo-aleatória que é definida unicamente por T, e ambos são protegidos pelo ECDLP, então é impossível para um adversário polinomial modificar $\mathcal{M}$ e ao mesmo tempo produzir um texto cifrado válido. Portanto esta modificação define um algoritmo de criptografia *não maleável* e conseqüentemente seguro contra um ataque de texto cifrado adaptativo.

7 CONSIDERAÇÕES FINAIS

Este trabalho apresentou o desenvolvimento de um algoritmo de criptografia de chave pública baseado na teoria de curvas elípticas. Como demonstrado, o algoritmo é semanticamente seguro dado que o cálculo do logaritmo discreto no grupo formado pelos pontos de uma curva elíptica adequadamente escolhida é um problema difícil.

Foi apresentada, também, uma nova função de resumo criptográfico, ou *hash*, baseada na função *Chaum-van Heijst-Pfitzmann*. Esta versão modificada, entretanto, tem sua segurança baseada no logaritmo em curvas elípticas de ordem prima, e não no logaritmo em um grupo multiplicativo. Utilizando esta função, o algoritmo principal é estendido a fim de apresentar a característica de não maleabilidade e, portanto, ser seguro contra um ataque de texto cifrado adaptativo, considerado o pior cenário de segurança para um algoritmo de chave pública.

Este capítulo final propõe-se a apresentar algumas discussões não abordadas nos demais capítulos, principalmente, no que diz respeito às decisões tomadas ao longo do desenvolvimento do algoritmo. Escolheu-se por apresentar estas questões neste último capítulo pelo caráter mais empírico das discussões.

Quanto ao par de chaves

A apresentação de qualquer algoritmo de chave pública deve se dedicar não só aos algoritmos de cifragem e decifragem, mas também à obtenção do par de chaves que ele utiliza. O algoritmo proposto utiliza parâmetros públicos que provavelmente não são compartilhados por nenhum outro algoritmo conhecido no momento: um *twisted pair de ordem prima*. Apesar de ser relativamente simples encontrar-se um *twisted pair*, procurar um em que ambas as curvas possuam ordem prima é uma tarefa que deve ser muito bem investigada, pois a sua busca por métodos óbvios revela-se extremamente ineficiente.

O método apresentado para tal é baseado na teoria de multiplicação complexa, e somente seus aspectos práticos foram apresentados. O algoritmo apresentado, entretanto, nem de longe é a única alternativa para este fim. Mesmo que, como vantagem, o método permita a compressão dos parâmetros das curvas, ainda assim algumas características dele podem chamar a atenção. Todo o método se baseia na idéia de modificar a equação 5.1 de forma a facilitar a busca das curvas. Em especial, três escolhas feitas requerem explicações.

A primeira questão diz respeito à variável y da equação. Optou-se por fixar o valor de y como 1, efetivamente eliminando-o da equação. O efeito mais óbvio desta escolha é que, os valores onde um valor y diferente de 1 resultaria em um par de curvas adequado são simplesmente ignorados. Por outro lado, variar o valor de y anularia a regra de que valores $D \equiv 3 \mod 8$ resultam em valores inteiros no polinômio. Dado que a simplicidade dos

polinômios foi considerada como mais importante, considerou-se a exclusão de y como a melhor escolha. Mesmo assim, seria interessante a investigação de um método similar à esse, mas onde o valor de y fosse diferente, ou então variasse.

A segunda questão que merece uma discussão é a escolha do traço $t(x)$, como na equação 5.2. De fato, esta é uma escolha de compromisso entre a granularidade da varredura e o nível de compressão possível nos parâmetros públicos. Claramente, uma equação cúbica, ou mesmo uma equação linear, satisfariam os requisitos. Porém, uma equação linear faria com que a compressão dos parâmetros fosse muito menor para um mesmo nível de segurança e uma equação cúbica faria com que, para cada valor testado na equação, muitos valores primos fossem pulados, possivelmente aumentando o tempo de busca. Opta-se, então, pelo polinômio quadrático que é um meio termo entre ambos.

A terceira questão diz respeito aos valores D sugeridos. Apesar de em momento algum isto ser um pré-requisito, efetivamente sugere-se o uso de curvas elípticas construídas utilizando-se polinômios de Hilbert com números de classe baixo, em especial 1 e 2. O método de compressão dos parâmetros públicos, particularmente, necessita desta restrição.

Longe de ser consenso na comunidade acadêmica, existem pesquisadores que acreditam que se deve evitar curvas elípticas deste tipo, preferindo-se, ao menos, curvas obtidas com discriminantes com número de classe igual a 100. Por exemplo, o governo da Alemanha apresenta como um requisito para a utilização de curvas elípticas para assinaturas digitais oficiais, que estas curvas sejam construídas utilizando-se valores D com número de classe de, no mínimo, 200.

A questão colocada é que, mesmo que alguns pesquisadores acreditem que isto possa vir a representar uma fraqueza das curvas, atualmente ninguém conseguiu sugerir nem de longe como um ataque destes seria possível. E mesmo que algum dia um ataque destes venha a ocorrer, provavelmente todo o método de criação de curvas via multiplicação complexa será afetado, não só algumas curvas com números de classe baixos. Desta forma, não há motivo para, em especial, não se aproveitar da simplicidade deste tipo de curva em prol de um sentimento sem fundamento algum. Por este motivo, isto não é levado em conta pelo algoritmo.

Quanto à eficiência do algoritmo

Talvez o maior defeito do algoritmo proposto é o fato de depender do algoritmo gerador de bits pseudo-aleatórios de Kaliski. Enquanto algoritmos baseados em fatoração conseguem retornar $O(n)$ bits pseudo-aleatórios por iteração, este gerador não ultrapassa $O(\log n)$ bits. Isso acontece pois o próprio fato do logaritmo sobre curvas elípticas ser mais difícil que a fatoração faz com que seja mais difícil provar que é seguro retornar mais de $\log n$ bits, mesmo que isso seja, intuitivamente, contraditório.

Esta deficiência em particular faz com que o algoritmo seja, em média, mais lento que algoritmos similares baseados em outros problemas. Entretanto, o fato de utilizar curvas elípticas faz com que as operações básicas sejam, em última instância, mais rápidas, fazendo com que esta diferença não seja tão grande quanto poderia ser à primeira vista. Como uma vantagem direta disto, ele apresenta muito menos requisitos de memória, o que em alguns ambientes, como *smart cards* é muito mais restritivo do que velocidade de processamento propriamente dita.

Quanto à extensão do algoritmo

Esta sugestão não foi imediatamente incluída como parte do algoritmo original pois ela depende da existência de uma função de *hash* e, portanto, é menos eficiente que a versão original do algoritmo. Dependendo do comprimento da mensagem a ser cifrada, diversos cálculos extras são necessários apenas para o cômputo do *hash*. Em alguns casos, este esforço extra pode dobrar o tempo de execução de uma cifragem ou decifragem.

Outro motivo para a separação da extensão é o fato do algoritmo estendido ser praticamente igual ao algoritmo original. A única diferença é a inclusão do *hash* do conteúdo cifrado, de forma a garantir a sua integridade contra mudanças propositais. Conclui-se, portanto, que seria uma escolha didática não misturar detalhes que podem não ser misturados. Desta forma, optou-se por apresentar detalhadamente o algoritmo sem esta extensão e, somente após, indicar as mudanças necessárias.

Ainda assim, é importante ressaltar que, provavelmente, o algoritmo na sua forma estendida deva ser privilegiado no uso prático. Isto, pois o cenário de atacantes ativos é um cenário extremamente comum, e o uso de um algoritmo de chave pública que não resista a ataques deste tipo é considerado, de certa forma, arriscado.

REFERÊNCIAS

BLAKE, I. F.; SEROUSSI, G.; SMART, N. P. **Elliptic Curves in Cryptography**. Cambridge, UK: Cambridge University Press, 1999. (London Mathematical Society Lecture Notes Series, 265).

BLUM, L.; BLUM, M.; SHUB, M. Comparison of two pseudo-random number generators. In: CRYPTO 1982, Santa Barbara. **Advances in Cryptology:** proceedings of CRYPTO'82. New York: Plenum Press, 1983. p.61–78. Disponível em: <http://www.informatik.uni-trier.de/~ley/db/conf/crypto/>. Acesso em: abr. 2006.

BLUM, L.; BLUM, M.; SHUB, M. A simple unpredictable pseudorandom number generator. **Journal of the ACM**, New York, v.15, n.2, p.364–383, 1986.

BLUM, M.; GOLDWASSER, S. An efficient probabilistic public key encryption scheme which hides all partial information. In: CRYPTO 1984, Santa Barbara. **Advances in Cryptology:** proceedings of CRYPTO'84. Berlin: Springer-Verlag, 1985. p.289–302. (Lecture Notes in Computer Science, v.196).

BLUM, M.; MICALI, S. How to generate cryptographically strong sequences of pseudo-random bits. **SIAM Journal on Computing**, Philadelphia, v.13, n.4, p.850–864, 1984.

BONEH, D.; FRANKLIN, M. K. Identity-Based Encryption from the Weil Pairing. In: ANNUAL INTERNATIONAL CRYPTOLOGY CONFERENCE, 21., 2001, Santa Barbara. **Advances in Cryptology:** proceedings of CRYPTO 2001. Berlin: Springer-Verlag, 2001. p.213–229. (Lecture Notes in Computer Science, v.2139).

BOYAR, J. Inferring sequences produced by pseudo-random number generators. **Journal of the ACM**, New York, v.36, n.1, p.129–141, 1989.

CHAUM, D.; HEIJST, E. van; PFITZMANN, B. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In: ANNUAL INTERNATIONAL CRYPTOLOGY CONFERENCE, 11., 1991, Santa Barbara. **Advances in Cryptology:** proceedings of CRYPTO'91. Berlin: Springer-Verlag, 1992. p.470–484. (Lecture Notes in Computer Science, v.576).

COHEN, H. **A course in computational algebraic number theory**. Berlin: Springer-Verlag, 1993. (Graduate Texts in Mathematics 138).

DEDIC, N.; REYZIN, L.; VADHAN, S. An Improved Pseudorandom Generator Bases on Hardness of Factoring. In: INTERNATIONAL ASSOCIATION FOR CRYPTOLOGIC

RESEARCH. **Cryptology ePrint Archive**. [S.l.], 2002. (Report 2002/131). Disponível em: <http://eprint.iacr.org/2002/131.pdf>. Acesso em: abr. 2006.

DIFFIE, W.; HELLMAN, M. New directions in cryptography. **IEEE Transactions on Information Theory**, New York, v.22, n.6, p.644-654, Nov. 1976.

GOLDREICH, O.; LEVIN, L. A. A hard-core predicate for all one-way functions. In: ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, STOC, 21., 1989, Seatle. **Proceedings...** New York: ACM Press, 1989. p.25–32.

GOLDREICH, O.; MEYER, B. Computational Indistinguishability - Algorithms vs. Circuits. In: ELECTRONIC COLLOQUIUM ON COMPUTATIONAL COMPLEXITY. **ECCC reports**. [S.l.]: 1996. (TR96-067). Disponível em: <http://eccc.hpi-web.de/eccc-reports/1996/TR96-067/Paper.pdf>. Acesso em: abr. 2006.

GOLDREICH, O.; ROSEN, V. On the security of modular exponentiation with application to the construction of pseudorandom generators. In: INTERNATIONAL ASSOCIATION FOR CRYPTOLOGIC RESEARCH. **Cryptology ePrint Archive**. [S.l.], 2000. (Report 2000/064). Disponível em: <http://eprint.iacr.org/2000/064.ps>. Acesso em: abr. 2006.

GOLDWASSER, S.; MICALI, S. Probabilistic encryption. **Journal of Computer and System Sciences**, New York, v.28, n.2, p.270–299, 1984.

HASTAD, J.; SCHRIFT, A. W.; SHAMIR, A. The discrete logarithm modulo a composite hides O(n) bits. **Journal of Computer and System Sciences**, New York, v.47, n.3, p.376–404, 1993.

KALISKI, B. S. **Elliptic curves and cryptography**: a pseudorandom bit generator and other tools. 1988. Tese (Doutorado em Ciência da Computação) — MIT, Cambridge, Mass., 1988.

KNUTH, D. E. **Seminumerical Algorithms**. 2nd ed. Reading, Mass.: Addison-Wesley, 1973. (The Art of Computer Programming, v.2).

KOBLITZ, N. Elliptic Curve Cryptosystems. **Mathematics of Computation**, Washington, v.48, p.203–209, 1987.

KOBLITZ, N.; MENEZES, A. J.; WU, Y.-H.; ZUCCHERATO, R. J. **Algebraic aspects of cryptography**. New York: Springer-Verlag, 1998.

KOLMOGOROV, A. N. Three approaches to the concept of the amount of information. **Problems on Information Transmission**, New York, v.1, n.1, 1965.

LIDL, R.; NIEDERREITER, H. **Introduction to Finite Fields and Their Applications**. Cambridge, UK: Cambridge University Press, 1994.

LONG, D. L.; WIGDERSON, A. How discreet is the discrete log? In: ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, STOC, 15., 1983, Boston. **Proceedings...** New York: ACM Press, 1983. p.413–420.

MAURER, U. M.; WOLF, S. Diffie-Hellman Oracles. In: ANNUAL INTERNATIONAL CRYPTOLOGY CONFERENCE, 16., 1996, Santa Barbara. **Advances in Cryptology:** proceedings of CRYPTO'96. Berlin: Springer-Verlag, 1996. p. 268-282. (Lecture Notes in Computer Science, v.1109).

MENEZES, A. **Elliptic Curve Public Key Cryptosystems**. [S.l.]: Kluwer Academic Publishers, 1993.

MENEZES, A. J.; VANSTONE, S.; OKAMOTO, T. Reducing elliptic curve logarithms to logarithms in a finite field. **IEEE Transactions on Information Theory**, New York, v.39, n.5, p.1639-1646, 1993.

MENEZES, A.; OORSCHOT, P. V.; VANSTONE, S. **Handbook of Applied Cryptography**. Boca Raton: CRC Press, 1996. (Discrete Mathematics and its Applications, 6)

MICALI, S.; RACKO, C.; SLOAN, R. Notions of security of public-key cryptosystems. **SIAM Journal on Computing**, Philadelphia, v.17, n.2, p.412–426, 1988.

MICALI, S.; SCHNORR, C. P. Efficient, perfect polynomial random number generators. **Journal of Cryptology**, New York, v.3, n.3, p.157–172, 1991.

MILLER, V. S. Use of elliptic curves in cryptography. In: CRYPTO 1985, Santa Barbara. **Advances in Cryptology:** proceedings of CRYPTO'85. Berlin: Springer-Verlag, 1986. p.417–426. (Lecture Notes in Computer Science, v.218).

OKAMOTO, T.; FUJISAKI, E.; MORITA, H. PSEC: provably secure elliptic curve encryption scheme. In: IEEE STANDARD ASSOCIATION. The P1363 Working Group. **IEEE P1363: Asymmetric Encryption**. [S.l.]: 1999. Disponível em: <http://grouper.ieee.org/groups/1363/P1363a/contributions/psec.pdf>. Acesso em: abr. 2006.

PERALTA, R. Simultaneous security of bits in the discrete log. In: WORKSHOP ON THE THEORY AND APPLICATION OF CRYPTOGRAPHIC TECHNIQUES, 1985, Linz. **Advances in Cryptology:** proceedings of EUROCRYPT'85. Berlin: Springer-Verlag, 1986. p.62–72. (Lecture Notes on Computer Science, v.219).

PERALTA, R. On the distribution of quadratic residues and nonresidues modulo a prime number. **Mathematics of Computation**, Washington, v.58, n.197, p.433–440, 1992.

PLUMSTEAD, J. B. Inferring a sequence generated by a linear congruence. In: CRYPTO 1982, Santa Barbara. **Advances in Cryptology:** proceedings of CRYPTO'82. New York: Plenum Press, 1983. p.317–319. Disponível em: <http://www.informatik.uni-trier.de/~ley/db/conf/crypto/>. Acesso em: abr. 2006.

SCHNEIER, B. **Applied Cryptography**. New York: John Wiley & Sons, 1996.

SCHNORR, C. P. Security Of Almost ALL Discrete Log Bits. In: ELECTRONIC COLLOQUIUM ON COMPUTATIONAL COMPLEXITY. **ECCC reports**. [S.l.]: 1998. (TR98-033). Disponível em: <http://eccc.hpi-web.de/eccc-reports/1998/TR98-033/Paper.pdf>. Acesso em: abr. 2006.

SHAMIR, A. Identity-based cryptosystems and signature schemes. In: CRYPTO 1984, Santa Barbara. **Advances in Cryptology:** proceedings of CRYPTO'84. Berlin: Springer-Verlag, 1985. p.47–53. (Lecture Notes in Computer Science, v.196).

SHOUP, V. Lower Bounds for Discrete Logarithms and Related Problems. In: ANNUAL INTERNATIONAL CONFERENCE ON THE THEORY AND APPLICATIONS OF CRYPTOGRAPHIC TECHNIQUES, 1997, Konstanz. **Advances in Cryptology:** proceedings of EUROCRYPT'97. Berlin: Springer-Verlag, 1997. p.256–266. (Lecture Notes in Computer Science, v.1233).

SILVERMAN, J. H. **Arithmetic of Elliptic Curves**. New York: Springer, 1986.

SILVERMAN, J. H.; TATE, J. **Rational Points on Elliptic Curves**. Berlin: Springer-Verlag, 1992. (Undergraduate Texts in Mathematics).

SMART, N. P. The discrete logarithm problem on elliptic curves of trace one. **Journal of Cryptology**, New York, v.12, n.3, p.193–196, 1999.

STINSON, D. R. **Cryptography**: theory and practice. Boca Raton: CRC Press, 1995. (Discrete Mathematics and its Applications, 36).

VAZIRANI, U. V.; VAZIRANI, V. V. Efficient and secure pseudo-random number generation. In: CRYPTO 1984, Santa Barbara. **Advances in Cryptology:** proceedings of CRYPTO'84. Berlin: Springer-Verlag, 1985. p.193–202. (Lecture Notes in Computer Science, v.196).

WANG, X.; YU, H. How to Break MD5 and Other Hash Functions. In: ANNUAL INTERNATIONAL CONFERENCE ON THE THEORY AND APPLICATIONS OF CRYPTOGRAPHIC TECHNIQUES, 24., 2005, Aarhus. **Advances in Cryptology:** proceedings of EUROCRYPT'05. Berlin: Springer-Verlag, 2005. p.19–35. (Lecture Notes in Computer Science, v.3494).

YAN, S. Y. **Number theory for computing**. 2nd ed. Berlin: Springer-Verlag, 2000.

YAO, A. C. Theory and applications of trapdoor functions. In: ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 21., 1982. **Proceedings...** New York: IEEE, 1982. p.80–91.