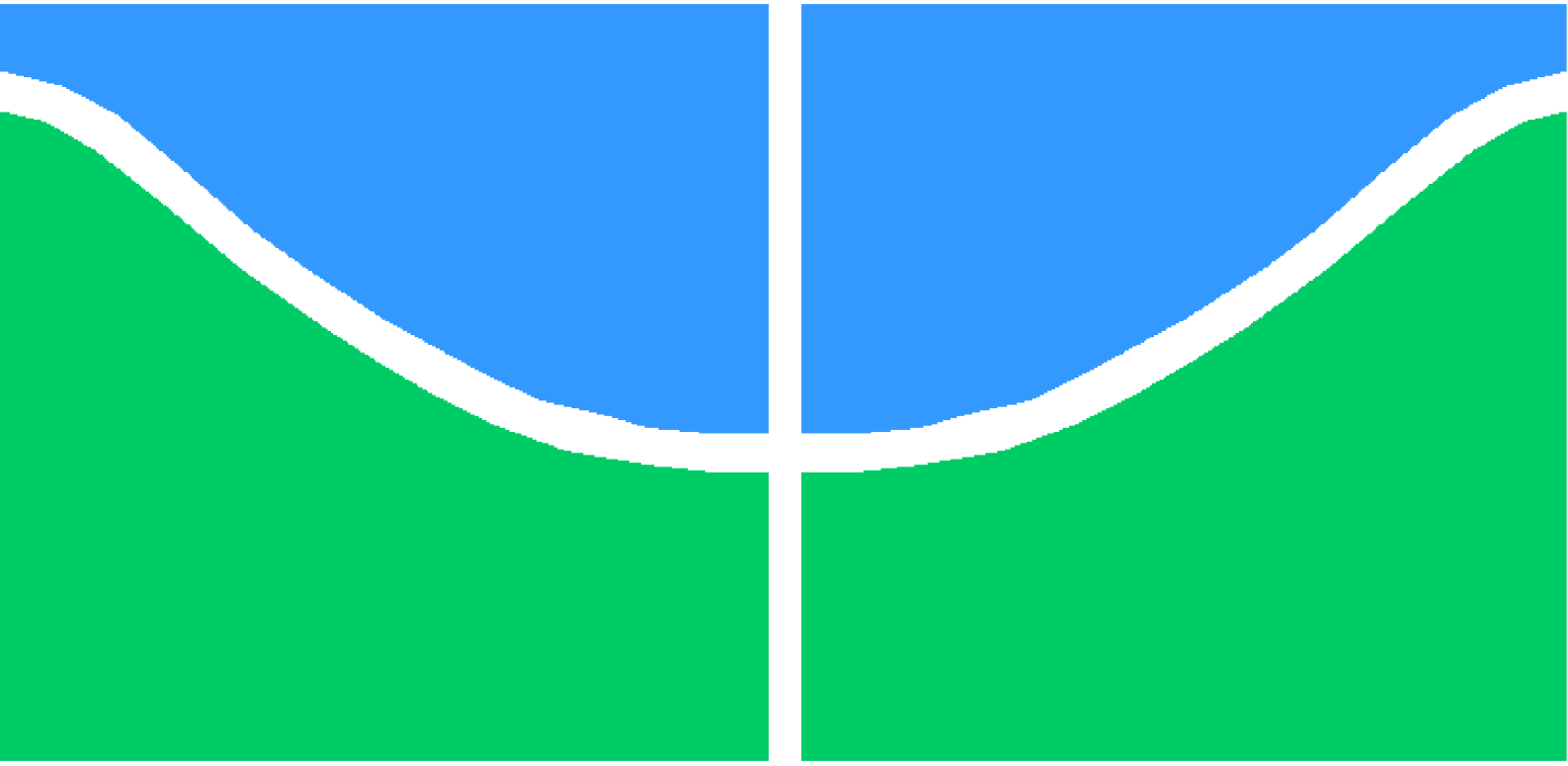

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Implementação e análise de perfomance do algoritmo Pollard-rho para ataque à segurança de criptografia de curvas elípticas

Autores: Rodrigo Santana Gonçalves e Carlos Alberto Teixeira Junior

Orientador: Prof. Dr. Luiz Augusto Laranjeira

Brasília, DF

2016

Rodrigo Santana Gonçalves e Carlos Alberto Teixeira Junior

Implementação e análise de perfomance do algoritmo Pollard-rho para ataque à segurança de criptografia de curvas elípticas

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Luiz Augusto Laranjeira

Coorientador: Prof. Dr. Edson Alves Costa Júnior

Brasília, DF

2016

Rodrigo Santana Gonçalves e Carlos Alberto Teixeira Junior

Implementação e análise de perfomance do algoritmo Pollard-rho para ataque à segurança de criptografia de curvas elípticas/ Rodrigo Santana Gonçalves e Carlos Alberto Teixeira Junior. – Brasília, DF, 2016-

85 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Luiz Augusto Laranjeira

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. Curva elíptica. 2. Pollard-rho. I. Prof. Dr. Luiz Augusto Laranjeira. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Implementação e análise de perfomance do algoritmo Pollard-rho para ataque à segurança de criptografia de curvas elípticas

CDU 02:141:005.6

Rodrigo Santana Gonçalves e Carlos Alberto Teixeira Junior

Implementação e análise de perfomance do algoritmo Pollard-rho para ataque à segurança de criptografia de curvas elípticas

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 7 de julho de 2016:

Prof. Dr. Luiz Augusto Laranjeira
Orientador

Prof. Tiago Alves
Convidado 1

Prof. Vinícius Rispoli
Convidado 2

Brasília, DF
2016

Agradecimentos

Eu, Rodrigo Gonçalves, agradeço a Deus acima de tudo, que me deu a oportunidade de estudar em uma universidade federal, que me sustentou durante toda a minha caminhada, e me deu forças para vencer os desafios da vida. Agradeço à minha família que sempre apoiou meus sonhos e objetivos, proporcionando um suporte emocional indispensável, em especial aos meus pais Robertson Gonçalves e Claudevânia de Santana, pois eles foram fundamentais para minha formação acadêmica. Agradeço aos demais amigos e companheiros que também proveram um suporte emocional e intelectual importantes para que eu chegasse até aqui.

Eu, Carlos Alberto, agradeço a Deus acima de tudo e à minha família por todo o esforço que fizeram para me ajudar durante o curso e por todo o incentivo.

Agradecemos aos orientadores, professor Luiz Laranjeira e professor Edson Alves pelos seus suportes. Além de todo corpo acadêmico da Faculdade UnB Gama, em especial os professores de Engenharia de Software por terem contribuído com a nossa formação acadêmica ao longo dos cinco anos de curso nos tornando profissionais qualificados no mercado de trabalho.

Resumo

O avanço da tecnologia nos últimos anos tem causado um grande impacto para a humanidade. Hoje em dia é praticamente inconcebível que exista alguma área que não utilize a tecnologia de alguma forma. Com esse crescimento tecnológico, torna-se imprescindível a criação de meios para garantir a segurança das pessoas. A criptografia tem importância central nesse contexto, pois evita que dados confidenciais sejam descobertos por pessoas mal-intencionadas. Porém, com o avanço da tecnologia, os sistemas criptográficos existentes demandam cada vez mais processamento e complexidade para se manterem seguros. Daí, é necessário buscar alternativas que garantam a segurança dos indivíduos no meio digital e que utilizem menos recursos computacionais. Uma dessas alternativas é a utilização de curvas elípticas em criptografia. Essa alternativa oferece a mesma segurança que a criptografia baseada no RSA, porém, consumindo bem menos recursos computacionais. A segurança da criptografia de curvas elípticas é baseada na dificuldade em resolver o problema do logaritmo discreto. No entanto, existem alguns algoritmos computacionais que resolvem esse problema, podendo comprometer a segurança do sistema criptográfico. O objetivo deste trabalho é explorar os principais conceitos que são empregados na interessante estrutura matemática de curvas elípticas e implementar um desses algoritmos de ataque, a saber, o algoritmo de Pollard-rho.

Palavras-chaves: Criptografia. Criptoanálise. Curva elíptica. Problema do logaritmo discreto. Pollard-rho.

Abstract

The advancement of technology in recent years has caused a great impact on humanity. Nowadays it is almost inconceivable that there is any area that does not use the technology in some way. With this technological growth, it is essential to create means to ensure the safety of persons. Encryption has central importance in this context because it prevents confidential data from being disclosed by malicious people. But with the advancement in technology, the existing cryptographic systems require increasingly complex processing and to stay safe. Hence, it is necessary to seek alternatives to ensure the safety of individuals in the digital environment and using less computing resources. One such alternative is the use of elliptic curves in cryptography. This alternative offers the same security that encryption based on RSA, however, consuming far less computing resources. The security of elliptic curve cryptography is based on the difficulty in solving the discrete logarithm problem. However, there are some computational algorithms that solve this problem, which may compromise the security of cryptographic system. The objective of this study is to explore the key concepts that are used in interesting mathematical structure of elliptic curves and implementation of these attack algorithms, namely the Pollard-rho algorithm.

Key-words: latex. abntex. text editoration.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AES	Advanced Encryption System
DSA	Digital Signature Algorithm
EC	Elliptic Curve
ECC	Elliptic Curve Criptography
ECDLP	Elliptic Curve Discrete Logarithm Problem
GCD	Greatest Common Divisor
NIST	National Institute for Standards and Technology
RSA	Rivest-Shamir-Adleman cryptosystem

Lista de símbolos

$\in$	Pertence
$\mathcal{O}$	Ponto no infinito
λ	Letra grega lambda
μ	Letra grega mu
π	Letra grega pi
ψ	Letra grega psi
ρ	Letra grega rho
θ	Letra grega theta

Sumário

Introdução

Contextualização

A maior parte dos produtos que utilizam a criptografia de chave pública para criptografia e assinaturas digitais utiliza RSA. O tamanho da chave para o uso seguro do RSA tem aumentado nos últimos anos, e isso gerou uma carga de processamento maior sobre as aplicações usando RSA. Esse peso tem ramificações, especialmente para sites de comércio eletrônico, que realizam grandes quantidades de transações seguras. Recentemente, um sistema concorrente começou a desafiar o RSA; criptografia de curva elíptica (ECC - *Elliptic Curve Cryptography*). O ECC já está aparecendo em esforços de padronização, como o IEEE P1363 *Standard for Public-Key Cryptography*. (LEE, 2011)

O atrativo principal do ECC, em comparação com o RSA, é que ele oferece igual segurança com um tamanho de chave muito menor reduzindo assim a carga de processamento. (STALLINGS, 2011) Em 2003, a principal empresa ligada à ECC (CERTICOM) promoveu um teste para verificação de segurança do criptossistema baseado em curvas elípticas, que foi atacado por 10.000 computadores do tipo Pentium durante 540 dias seguidos. Nesse episódio foi quebrado um sistema com chave de 109 bits, utilizando um ataque conhecido como ataque aniversário (*birthday attack*). Atualmente, o tamanho mínimo de chave recomendado pelo NIST para se obter um bom nível de segurança em ECC é de, pelo menos, 163 bits; enquanto para RSA este tamanho é de 2048 bits. (SANGALLI, 2011)

Por outro lado, embora a teoria do ECC já exista há algum tempo, só nos últimos 20 anos esses produtos começaram a aparecer, e tem havido interesse criptoanalítico contínuo em encontrar algum ponto fraco. (STALLINGS, 2011) Por conseguinte, RSA é mais comumente utilizado do que o ECC por usar estruturas matemáticas mais simples. Isso se deve ao fato de se definir alguns parâmetros antes de executar o algoritmo de cifração/decifração, sendo estes parâmetros não comuns em outros criptossistemas. Devemos definir inicialmente um corpo finito e, em seguida, definir a curva elíptica para que possamos gerar um grupo abeliano, composto por pontos da curva elíptica, sobre o qual as operações serão definidas. (SANGALLI, 2011)

Objetivos

1. **Objetivo Geral**

   Objetivo deste trabalho consiste em realizar pesquisas acerca da criptografia de

curvas elípticas e implementar o algoritmo Pollard-rho - que se propõe a resolver o problema matemático no qual está baseada a segurança deste sistema criptográfico - e suas variações, comparando o tempo de execução destas variações entre si.

2. **Objetivos Específicos**

   a) Apresentar conceitos matemáticos importantes sobre curvas elípticas e sua aplicação na criptografia.

   b) Demonstrar o problema do logaritmo discreto e algoritmos para resolvê-lo no contexto de ECC.

   c) Implementar o algoritmo Pollard-rho e suas variações: com único processador, com múltiplos processadores e utilizando uma versão paralela.

   d) Utilizar os algoritmos implementados para solucionar o problema do logaritmo discreto para curvas elípticas mais simples, que não tenham utilização prática para criptografia de curvas elípticas. Serão utilizadas curvas elípticas mais simples por dois motivos:

      - Para validar o trabalho em tempo hábil, ou seja, deseja-se obter um resultado verificável em um prazo que esteja de acordo com o calendário acadêmico da UnB para o primeiro semestre de 2016. Pois a literatura pesquisada faz referências sobre o tempo necessário para resolver o problema do logaritmo discreto para curvas de interesse criptográfico (ver Seção 2.1.3), sendo este impraticável para o escopo deste trabalho.
      - A literatura pesquisada (ver Seção 2.1.3) cita a dificuldade e o poder computacional necessário para solucionar tal problema com curvas muito complexas, fazendo alusão inclusive à experimentos já realizados utilizando-se milhares de computadores. Como não dispomos do mesmo poder computacional, optamos por resolver o problema para curvas mais simples.

   e) Apresentar as comparações de resultados práticos dos desempenhos dos algoritmos implementados.

1 Fundamentação Teórica

Neste capítulo serão mostrados os conceitos matemáticos necessários para fundamentar a busca dos objetivos propostos.

1.1 Aritmética Modular

O **teorema de Eudoxius** afirma que, dados dois inteiros a e b com $b \neq 0$, então, ou a é múltiplo de b, ou se encontra entre dois múltiplos consecutivos de b, assim, existe um inteiro q tal que (SANTOS, 2014)

$$qb \leq a < (q+1)b \quad \text{para } b > 0 \tag{1}$$

$$qb \leq a < (q-1)b \quad \text{para } b < 0 \tag{2}$$

O **teorema da divisão** diz que: dados dois inteiros a, b, com $b > 0$, ao dividir a por b, obtém-se um único par de inteiros q, r que obedecem ao seguinte relacionamento:

$$a = qb + r \qquad 0 \leq r < b;\ q = \lfloor a/b \rfloor \tag{3}$$

em que $\lfloor x \rfloor$ é o maior inteiro menor ou igual a x e q, r são chamados respectivamente de *quociente* e *resto* ou *resíduo* da divisão de a por b (SANTOS, 2014). A demonstração será feita utilizando o teorema de Eudoxius descrito acima. Dessa forma, existe um inteiro q que satisfaz a Eq. (1), o que implica que $0 \leq a - qb$ e que $a - qb < b$. Definindo $r = a - qb$, teremos demonstrado a existência de q e r. A unicidade de q e r pode ser demonstrada por contradição. Suponhamos que exista outro par de inteiros q_1 e r_1 que satisfaçam a equação

$$a = q_1 b + r_1 \tag{4}$$

então $(qb + r) - (q_1 b - r_1) = 0$, assim temos que $b(q - q_1) = r_1 - r$, ou seja, $b|(r_1 - r)$. Mas isso só é possível se $r_1 - r = 0$ pois $r < b$ e $r_1 < b$, dessa forma temos que $r_1 = r$ e assim fica também demonstrada a unicidade de q e r. (SANTOS, 2014)

A aritmética modular trata da relação entre os inteiros e seus resíduos quando são divididos por algum inteiro positivo denominado **módulo** (LEWINTER; MAYER, 2015). Para exemplificar melhor, considere a sequência de números naturais $S = 3, 4, 5, \ldots$ e o módulo sendo $b = 3$, é possível reescrever qualquer número dessa sequência como $3q$, $3q + 1$ ou $3q + 2$. Dessa forma, qualquer número da sequência se encaixa em uma dessas três classes. Analogamente, se o módulo for n, então haverá n **classes de resto mod n**.

A aritmética modular estuda essa relação entre cada inteiro da sequência S e sua classe de resto mod n (LEWINTER; MAYER, 2015).

Quando dois inteiros pertencem à mesma classe de resto, diz-se que estes são *congruentes mod k*. Uma definição mais formal de congruência é: dados dois inteiros a e b, diz-se que a é *congruente* a b módulo m se m divide a diferença $(a-b)$, ou seja, $m \mid (a-b)$. A relação de congruência entre a e b é representada pela notação abaixo.

$$a \equiv b \pmod{m} \tag{5}$$

Consequentemente, se m não divide a diferença $(a-b)$, então a é *incongruente* a b *mod* m, e representa-se por $a \not\equiv b \pmod{m}$ (SANTOS, 2014).

Seja um inteiro positivo n e a, b, c inteiros quaisquer, a congruência tem as seguintes propriedades (STALLINGS, 2011):

1. $a \equiv b \pmod{n}$ se $n|(a-b)$.
2. $a \equiv b \pmod{n}$ implica $b \equiv a \pmod{n}$.
3. $a \equiv b \pmod{n}$ e $b \equiv c \pmod{n}$ implica $a \equiv c \pmod{n}$.

Por definição, o operador $\pmod{n}$ mapeia todos os inteiros para o conjunto de inteiros $\{0, 1, ..., (n-1)\}$. Daí pode-se realizar operações aritméticas dentro dos limites desse conjunto, técnica conhecida como aritmética modular. A aritmética modular exibe as seguintes propriedades (STALLINGS, 2011):

1. $[a \pmod{n} + b \pmod{n}] \pmod{n} = (a+b) \pmod{n}$
2. $[a \pmod{n} - b \pmod{n}] \pmod{n} = (a-b) \pmod{n}$
3. $[a \pmod{n} \times b \pmod{n}] \pmod{n} = (a \times b) \pmod{n}$

Antes de tratar a divisão módulo n, é importante fixar o conceito de **máximo divisor comum**(MDC) entre dois números inteiros. O MDC entre dois inteiros a e b é o maior inteiro que divide simultaneamente a e b.

A divisão módulo n não está definida em todos os casos. Seja d um número inteiro que divide a e b, verifica-se que a relação apresentada na Eq. (6), em que $\text{mdc}(n, d)$ é o máximo divisor comum entre n e d.

$$\frac{a}{d} \equiv \frac{b}{d} \left(\text{mod } \frac{n}{\text{mdc}(n,d)} \right) \tag{6}$$

O inverso multiplicativo de a módulo n é o número inteiro b que satisfaz a operação $a \times b = 1 \pmod{n}$, e somente pode ser obtido quando existe $(a, n) = 1$ (isto é, a e n são

coprimos). O valor do inverso multiplicativo b pode ser obtido através do Algoritmo de Euclides Estendido (HALIM, 2013).

Outra operação que pode ser definida é a exponenciação modular, que calcula o resto de um número inteiro b quando elevado a um número inteiro k e é dividido por um inteiro positivo m. Assim, b é a base, k o expoente e m o módulo da exponenciação modular, que pode ser expressa como na Eq. (7).

$$c \equiv b^k \pmod{m} \tag{7}$$

1.2 Relações de Equivalência

Em matemática, relação é o estudo de como os elementos de um conjunto E, ou de dois conjuntos distintos, se relacionam entre si (DOMINGUES; IEZZI, 2003). O conceito de relação pode ser compreendido facilmente partindo-se da definição de outro conceito, o de *produto cartesiano*.

Sejam dois conjuntos não vazios E e F, o **produto cartesiano** entre esses dois conjuntos, representado por $E \times F$, é o conjunto de todos os pares ordenados (x, y), tais que $x \in E$ e $y \in F$.

$$E \times F = (x, y) | x \in E \text{ e } y \in F \tag{8}$$

Sabendo-se o conceito de produto cartesiano, define-se *relação binária* de E em F como qualquer subconjunto de $E \times F$, ou seja, uma relação $\mathcal{R}$ une um $x \in E$ a um $y \in F$ em um par ordenado (x, y), e dizemos que x *se relaciona com* y *mediante* $\mathcal{R}$, que pode ser representado por $x\mathcal{R}y$ (DOMINGUES; IEZZI, 2003). Lembrando-se que, uma relação $\mathcal{R}$ pode ser definida somente sobre um conjunto E, ou seja, a relação $\mathcal{R}$ é um subconjunto do produto cartesiano $E \times E$.

Quando definida sobre um conjunto E, algumas propriedades que a relação $\mathcal{R}$ pode atender são:

- **Reflexiva**: a relação $\mathcal{R}$ é reflexiva se todo elemento $x \in E$ se relaciona consigo mesmo, ou seja, $x\mathcal{R}x$. Por exemplo, para $E = \{a, b, c\}$, a relação

  $$\mathcal{R} = \{(a, a), (b, b), (c, c), (a, b), (a, c)\}$$

  é reflexiva, pois para cada elemento $x \in E$, existe um par ordenado (x, x) em $\mathcal{R}$ (DOMINGUES; IEZZI, 2003).

- **Simétrica**: a relação $\mathcal{R}$ é simétrica se é válido afirmar que $y\mathcal{R}x$ sempre que $x\mathcal{R}y$, com $x, y \in E$. Por exemplo, a relação

$$\mathcal{R} = \{(a,a),(a,b),(b,a),(c,c)\}$$

é simétrica sobre $E = \{a,b,c\}$, pois é facilmente verificável que $y\mathcal{R}x$ sempre que $x\mathcal{R}y$ (DOMINGUES; IEZZI, 2003).

- **Transitiva**: a relação $\mathcal{R}$ é transitiva se é válido afirmar que $x\mathcal{R}z$ sempre que $x\mathcal{R}y$ e $y\mathcal{R}z$. Por exemplo, a relação

$$\mathcal{R} = \{(a,b),(b,c),(a,c),(c,c)\}$$

é transitiva sobre $E = \{a,b,c\}$, pois pode-se verificar que $a\mathcal{R}b$ e $b\mathcal{R}c$ implica em $a\mathcal{R}c$ (DOMINGUES; IEZZI, 2003).

- **Antissimétrica**: a propriedade antissimétrica de uma relação $\mathcal{R}$, como o nome já diz, é o contrário da propriedade simétrica, ou seja, se o par ordenado $(x,y) \in \mathcal{R}$, então $(y,x) \notin \mathcal{R}$, exceto se $x = y$. Por exemplo, a relação

$$\mathcal{R} = \{(a,a),(a,b),(b,c),(c,c)\}$$

é antissimétrica sobre $E = \{a,b,c\}$, pois para cada par $(x,y) \in \mathcal{R}$, com $x \neq y$, o par $(y,x) \notin \mathcal{R}$.

Uma relação que satisfaz as propriedades reflexiva, simétrica e transitiva citadas acima, é chamada de **relação de equivalência** (DOMINGUES; IEZZI, 2003).

Seja o conjunto E e a relação de equivalência $\mathcal{R}$ sobre E, para cada elemento $a \in E$ existe uma **classe de equivalência** representada por $\overline{a}$ e constituída por todos os elementos em E que estão relacionados com a mediante a relação $\mathcal{R}$

$$\overline{a} = \{x \in E | x\mathcal{R}a\}$$

e diz-se que $\overline{a}$ é a classe de equivalência determinada por a, módulo $\mathcal{R}$ (DOMINGUES; IEZZI, 2003). Por exemplo, na relação de equivalência

$$\mathcal{R} = \{(a,a),(b,b),(c,c),(d,d),(a,b),(b,a),(b,c),(c,b),(a,c),(c,a)\}$$

sobre o conjunto $E = \{a,b,c,d\}$, a classe de equivalência determinada por a módulo $\mathcal{R}$ é dada por

$$\overline{a} = \{a,b,c\}$$

pois apenas os elementos a, b e c estão relacionados com a mediante $\mathcal{R}$.

O conjunto formado pelas classes de equivalência módulo $\mathcal{R}$ é chamado de **conjunto-quociente** de E por $\mathcal{R}$ e indicado por $E/\mathcal{R}$ (DOMINGUES; IEZZI, 2003). Por exemplo, para o exemplo anterior, as classes de equivalência são

$$\overline{a} = \{a, b, c\}, \quad \overline{b} = \{b, a, c\}, \quad \overline{c} = \{c, a, b\}, \quad \overline{d} = \{d\}$$

logo, o conjunto-quociente de E por $\mathcal{R}$ é dado por

$$E/\mathcal{R} = \{\{a, b, c\}, \{d\}\}$$

Um partição de um conjunto não vazio E é formada pela classe $\mathcal{F}$ dos seus subconjuntos não vazios tal que sejam satisfeitas as propriedades (DOMINGUES; IEZZI, 2003):

1. Para quaisquer dois membros da classe $\mathcal{F}$, ou eles são iguais, ou são disjuntos, ou seja, não possuem elementos em comum.
2. A união dos membros da classe $\mathcal{F}$ é igual ao conjunto E.

É importante notar que a união das classes de equivalência é igual ao próprio conjunto E, isto porque uma relação de equivalência sobre um conjunto E faz com que $E/\mathcal{R}$ seja uma partição do conjunto E (DOMINGUES; IEZZI, 2003).

1.3 Estruturas Algébricas

Antes de iniciar os estudos em aritmética de curvas elípticas, é importante fixar alguns conceitos de álgebra. De acordo com Hefez, estruturas algébricas são modelos abstratos para tratar em bloco várias situações matemáticas concretas, em que determinados conjuntos definem operações com propriedades semelhantes (HEFEZ; VILLELA, 2008).

1.3.1 Grupos

Um **grupo** $(G, *)$ é composto por um conjunto G e uma operação binária $*$ sobre os elementos desse conjunto tal que os axiomas abaixo sejam satisfeitos (GILBERT; NICHOLSON, 2004).

1. O conjunto G é **fechado** para a operação $*$

   $a * b \in G$ para todo $a, b \in G$.

2. A operação $*$ é **associativa**

   $(a * b) * c = a * (b * c)$ para todo $a, b, c \in G$.

3. Existe um **elemento identidade** $e \in G$ tal que para todo elemento $a \in G, a*e = a$.

4. Existe um **elemento inverso**[1] a^{-1} para todo elemento $a \in G$ tal que $a^{-1} * a = e$ (elemento identidade).

Se a operação é **comutativa**, ou seja, se $a * b = b * a$ para todo $a, b \in G$, então o grupo é denominado **abeliano** (ou **comutativo**) em homenagem ao matemático Niels Abel (GILBERT; NICHOLSON, 2004). Exemplos de grupos abelianos são $(\mathbb{Z}, +)$, $(\mathbb{R}, +)$, ambos com identidade 0 e com infinitos elementos.

A **ordem do grupo** é dada pela quantidade de elementos que este possui (COUTINHO, 2014). Assim, um grupo que possui infinitos elementos, por exemplo, o grupo $(\mathbb{Z}, +)$, diz-se que possui ordem infinita, já os grupos que possuem uma quantidade finita de elementos, diz-se possuir ordem finita. Para este trabalho, os grupos de maior interesse são aqueles que possuem ordem finita.

1.3.2 Subgrupos

Seja $(G, *)$ um grupo, e H um subconjunto não vazio de G. Se $(H, *)$ satisfaz todos os axiomas de grupo, então diz-se que $(H, *)$ é um subgrupo de $(G, *)$ (COUTINHO, 2014), ou seja

1. Para todo $a, b \in H$, $a * b \in H$.

2. O elemento neutro de H é o mesmo elemento neutro de G.

3. Existe um inverso a' para todo $a \in H$.

Todo grupo possui pelo menos dois subgrupos. Esses dois subgrupos são chamados de **subgrupos triviais**. Um deles é o subgrupo formado unicamente pelo elemento neutro, ou seja, por e, pois esse grupo, apesar de formado por um conjunto composto somente pelo elemento neutro, satisfaz todos os axiomas de grupo para a operação dada. O segundo subgrupo trivial de um grupo $(G, *)$ é ele mesmo (DOMINGUES; IEZZI, 2003).

1.3.3 Teorema de Lagrange

Um teorema de grande importância para o estudo da criptografia é o teorema de Lagrange, no qual estabelece que, se G é um grupo finito e H é um subgrupo de G, então

[1] Para um grupo é necessário que um elemento a tenha um elemento *simétrico*, denotado genericamente por a'. Se a operação do grupo for a multiplicação, então o elemento simétrico é chamado de *inverso* de a, e é representado por a^{-1}. Se a operação do grupo for a soma, então o elemento simétrico é chamado de *negativo* de a, e é representado por $-a$ (DOMINGUES; IEZZI, 2003). Para manter a simplicidade da notação nesse texto, o simétrico sempre será equivalente ao inverso, mesmo que a operação do grupo seja a soma.

a ordem de H sempre divide a ordem de G (SHOUP, 2005). Para a demostração desse teorema, além dos conceitos já apresentados, faz-se necessário apresentar o conceito de classes laterais e algumas propriedades que podem ser extraídas a partir destas.

1.3.3.1 Classes Laterais

Seja um grupo G, um subgrupo H de G, e um elemento $a \in G$, chama-se **classe lateral à direita**, módulo H, determinada por a, qualquer subconjunto de G formado pelos elementos Ha tais que $Ha = \{ha \mid h \in H\}$ (GILBERT; NICHOLSON, 2004). Por exemplo, considere um grupo multiplicativo formado pelo conjunto $G = \{1, -1, i, -i\}$, agora considere o subconjunto $H = \{1, -1\}$ (DOMINGUES; IEZZI, 2003). As classes laterais à direita são dadas por

$$\begin{aligned} 1 \cdot H &= \{1 \cdot 1,\ 1 \cdot -1\} = \{1, -1\} \\ -1 \cdot H &= \{-1 \cdot 1,\ -1 \cdot -1\} = \{-1, 1\} \\ i \cdot H &= \{i \cdot 1,\ i \cdot -1\} = \{i, -i\} \\ -i \cdot H &= \{-i \cdot 1,\ -i \cdot -1\} = \{-i, i\} \end{aligned}$$

De forma análoga, as **classes laterais à esquerda** são definidas pelos subconjuntos $aH = \{ah \mid h \in H\}$. Assim, se o grupo for um grupo abeliano, então $aH = Ha$ (DOMINGUES; IEZZI, 2003). O conjunto-quociente de G por H, nesse caso, é dado por

$$G/H = \{\{1H\}, \{iH\}\}$$

O **índice de** H **em** G é a quantidade de classes laterais distintas que compõe o conjunto-quociente, sendo denotado por $(G : H)$. No caso acima, o índice é igual a dois, pois o conjunto quociente é formado pelas classes laterais $\{1H\}$ e $\{iH\}$ (DOMINGUES; IEZZI, 2003).

Pode-se perceber que todas as classes laterais tem a mesma cardinalidade, ou seja, possuem a mesma quantidade de elementos, que é a mesma quantidade de elementos que possui o conjunto H (SHOUP, 2005).

1.3.3.2 Demonstração do Teorema de Lagrange

Seja um grupo G e um subgrupo H, suponha que o índice de H em G seja dado por $(G : H) = r$ e que o conjunto-quociente seja $G/H = \{a_1H, a_2H, \ldots, a_rH\}$. Sabe-se, de acordo com a Seção 1.2, que a união de todos os elementos de G/H é igual ao próprio conjunto G e também, pela seção anterior, que cada classe lateral que compõe G/H possui cardinalidade igual a do subconjunto H. Dessa forma, a ordem de G é igual a soma das ordens das classes laterais, o que pode ser escrito da seguinte forma

$$o(G) = o(H) + o(H) + \ldots + o(H)$$

em que $o(G)$ indica a ordem do conjunto G. Como $(G : H) = r$, então pode-se escrever

$$o(G) = (G : H) \cdot o(H) = r \cdot o(H)$$

e portanto, $o(H) \mid o(G)$, ou seja, a ordem do subgrupo H divide a ordem do grupo G (DOMINGUES; IEZZI, 2003).

1.3.4 Subgrupos cíclicos

Seja o grupo finito $(G, *)$ e um elemento $a \in G$. É possível realizar a operação $*$ do grupo repetidas vezes sobre o elemento a, obtendo

$$a^k = a * a * \ldots * a \quad \text{(k vezes)}$$

que também é um elemento do grupo finito $(G, *)$, pois, pela definição de grupo, o conjunto G deve ser fechado para a operação $*$. Pode-se dizer que essa é a k-ésima *potência* de a (assim, $a * a = a^2$ mesmo que a operação $*$ do grupo não seja equivalente à multiplicação). Assim, é possível criar um conjunto com as potências de a, obtendo

$$H = \{e, a, a^2, a^3, \ldots\}$$

que é um subconjunto de G. Como o conjunto G é um conjunto finito, então o conjunto H também é finito e existem inteiros positivos $n > m$, tais que $a^m = a^n$, por exemplo, se $m = 0$ temos que $a^m = a^0 = e$, que é o elemento neutro, e também existe um inteiro $n > 0$ tal que $a^n = e$.

Considere que o inverso de a seja a^{-1}, multiplicando ambos os lados desta equação por $(a^{-1})^m$, obtém-se $a^{m-m} = a^{n-m} = e$, que é o elemento neutro (COUTINHO, 2014). Daí pode-se afirmar que

1. Dado um elemento $a \in G$, existe um inteiro positivo k tal que $a^k = e$.
2. Se $a^k = e$, então o inverso de a é dado por a^{k-1}, pois $a * a^{k-1} = a^k = e$. Por ser uma potência de a, o inverso de a também pertence ao conjunto H.

A **ordem do elemento** $a \in G$ é o menor inteiro positivo k tal que $a^k = e$. Como o elemento a gera o conjunto H, diz-se que a é um **gerador** do conjunto H, e consequentemente, a ordem do conjunto H é igual à ordem de a. Quando um grupo é formado por um elemento gerador, diz-se que tal grupo é um *grupo cíclico* (COUTINHO, 2014).

Pelo teorema de Lagrange, é possível concluir que a ordem de H divide a ordem de G. Uma importante consequência dessa afirmação é que, se um grupo G possui ordem p prima, então seus subgrupos devem ter ordem 1 ou p. Porém, como um subgrupo H

formado pelas potências de um elemento a deve possuir pelo menos o elemento neutro e e um gerador a, ou seja, deve possuir pelo menos dois elementos, então todos os subgrupos de G, com exceção do subgrupo trivial formado pelo elemento neutro, devem ter ordem p. Em consequência disso, qualquer elemento $b \in G$ é um gerador do grupo e todo grupo de ordem prima é um grupo cíclico, embora nem todo grupo cíclico possua ordem prima. Por exemplo, o grupo formado formado pelo conjunto $G = \{1, 2, 3, 4\}(\text{mod } 5)$ e a operação de multiplicação têm ordem $k = 4$, porém admite-se o elemento 2 como gerador (COUTINHO, 2014).

1.3.5 Homomorfismos de grupos

Sejam os grupos $(G, \cdot)$ e $(H, \cdot)$, uma função $f : G \to H$ é chamada de *homomorfismo* se $f(a \cdot b) = f(a) \cdot f(b)$ em que $a, b \in G$. Se a operação dos grupos for diferente, por exemplo, $(G, \cdot)$ e $(H, \star)$, então a condição é dada por $f(a \cdot b) = f(a) \star f(b)$ (GILBERT; NICHOLSON, 2004).

Sendo $f : G \to H$ uma função de homomorfismo entre os grupos G e H, se a função for bijetora, então ela é denominada *isomorfismo*, e ainda, se os grupos G e H são iguais e a função é um isomorfismo, então ela é denominada *automorfismo* (SHOKRANIAN, 2010).

1.3.6 Corpos Finitos

Corpos finitos são estruturas algébricas compostas por um conjunto numérico $\mathbb{F}$ juntamente com duas operações, sendo uma delas a operação de soma (denotada por $+$) e a outra a operação de multiplicação (denotada por $\cdot$) em que são satisfeitas as seguintes propriedades: (HANKERSON; MENEZES; VANSTONE, 2004)

1. $(\mathbb{F}, +)$ é um grupo abeliano com elemento identidade sendo 0.
2. $(\mathbb{F}\backslash\{0\}, \cdot)$ é um grupo abeliano com elemento identidade sendo 1.
3. Para todo $a, b, c \in \mathbb{F}$, é válida a propriedade distributiva $(a + b) \cdot c = a \cdot c + b \cdot c$.

Se o conjunto $\mathbb{F}$ é finito, então o corpo é denominado *corpo finito*. A *ordem* de um corpo finito indica a quantidade de elementos que constituem o corpo finito. Existe um corpo finito de ordem q se e somente se q é igual a uma potência prima, ou seja, $q = p^m$, em que p é um número primo. O número p é chamado de característica do corpo finito. Se $m = 1$ então o corpo é denominado *corpo primo*, caso $m \geq 2$, o corpo é chamado de *corpo extenso* (HANKERSON; MENEZES; VANSTONE, 2004).

1.4 Criptografia

A criptografia é uma ciência que tem por objetivo estudar as formas de cifrar e decifrar a escrita, de modo a torná-la ininteligível para aqueles que não devem ter acesso à informação nela contida. Em Tecnologia da Informação, a criptografia é utilizada para cifrar não apenas a escrita, mas qualquer tipo de dados que necessitem de um tratamento mais restrito quanto ao seu acesso. Um Sistema de Criptografia define procedimentos para cifrar um texto (ou documento, ou qualquer tipo de dado), transformando-o em um texto cifrado (*ciphertext*) e decifrar o mesmo texto cifrado, obtendo-se novamente o texto original (*plaintext*), através de algoritmos. A primeira ação é denominada cifração e a segunda é chamada de decifração (PORTNOI, 2005).

A criptografia, em geral, fornece quatro serviços: (PORTNOI, 2005)

- *confidencialidade*: é a cifração dos dados de forma que somente seja inteligível aos receptores legítimos.
- *integridade*: durante a cifração dos dados, é necessário garantir que a informação não seja modificada pelo algoritmo de cifração, ou seja, a forma como os dados se apresentam deve ser modificada, mas sua informação deve permanecer a mesma.
- *autenticação*: é a garantia de que o emissor e o receptor são realmente quem dizem ser, ou seja, é a forma de verificar as identidades uns dos outros assim como a origem e o destino da informação. A autenticação pode ser obtida por meio de assinatura digital. Outra forma de obtê-la seria através de HMAC ou Cifração autenticada(*Authenticated Encryption*).
- *irretratabilidade*: impede que o emissor ou o receptor negue a mensagem transmitida, dessa forma, o emissor pode provar que o receptor recebeu a mensagem e o receptor pode provar que o emissor a enviou.

1.4.1 Criptografia simétrica

A criptografia simétrica, ou convencional, foi o primeiro tipo de criptografia utilizado. Baseia-se no uso de uma chave secreta para cifrar e decifrar a mensagem. É importante notar que a mesma chave utilizada para cifrar a mensagem é utilizada para decifrá-la. Dessa forma, é necessário que o emissor e o receptor saibam interpretar a chave secreta, assim, é necessário utilizar um canal seguro para compartilhar tal chave. Esse tipo de criptografia é bastante utilizado em transmissões de dados para as quais exista uma forma barata de estabelecer um canal seguro, tais como como mensagens enviadas de um computador para outro, nas comunicações entre duas máquinas ou no armazenamento da informação em um disco rígido, dentre outras (CAVALCANTE, 2015).

1.4.2 Criptografia assimétrica

A criptografia assimétrica, também conhecida como criptografia de chave pública, utiliza duas chaves distintas, uma para cifrar os dados (chamada de chave pública) e outra para decifrar os dados (chamada de chave privada). Neste caso a chave pública é divulgada enquanto que a chave privada permanece secreta. Assim, o emissor tem acesso à chave pública do receptor e utiliza esta para cifrar a mensagem. O receptor ao receber a mensagem, utiliza sua chave privada para decifrá-la. Esse tipo de criptografia não precisa de um canal seguro para compartilhamento da chave, pois a chave pública está disponível para qualquer um que queira enviar a mensagem para o receptor, porém, somente o receptor possui a chave privada (CAVALCANTE, 2015). Este sistema é capaz de prover a confidencialidade dos dados pois somente o proprietário da chave privada será capaz de decifrar a mensagem. Em algoritmos dessa natureza, um dado que é criptografado com uma chave só poderá ser decifrado com a utilização da outra chave e vice-versa.

1.5 Curvas Elípticas

1.5.1 Definição

Curvas elípticas têm esse nome porque são descritas por equações cúbicas, semelhantes às usadas para calcular a circunferência de uma elipse (STALLINGS, 2011). Uma curva elíptica E é um conjunto de soluções de uma equação cúbica definida sobre um corpo $\mathbb{F}$ é descrita da seguinte forma:

$$y^2 + axy + by = x^3 + cx^2 + dx + e \tag{9}$$

em que a, b, c, d e e são elementos do corpo $\mathbb{F}$ e x e y são as variáveis da equação e assumem valores dentro de $\mathbb{F}$. A notação utilizada para indicar que uma curva elíptica E está definida sobre um corpo $\mathbb{F}$ é $E(\mathbb{F})$, e o corpo $\mathbb{F}$ é chamado de corpo subjacente da curva (HANKERSON; MENEZES; VANSTONE, 2004). Equações deste tipo são conhecidas como *equações de Weierstrass*. Com frequência, no estudo de criptografia de curvas elípticas, costuma-se utilizar a forma reduzida de Weierstrass, descrita pela equação

$$y^2 = x^3 + ax + b \tag{10}$$

obtida através da mudança de variáveis

$$(x, y) \rightarrow \left(\frac{x - 3a^2 - 12c}{36}, \ \frac{y - 3ax}{216} + \frac{a^3 + 4ac - b}{24} \right) \tag{11}$$

e $\mathbb{F}$ com característica diferente de 2 e de 3 (HANKERSON; MENEZES; VANSTONE, 2004).

É imprescindível que os valores de a e b sejam escolhidos tais que $4a^3 + 27b^2 \neq 0$ (SEET, 2007). Essa condição garante que não existe ponto na curva que possua mais de

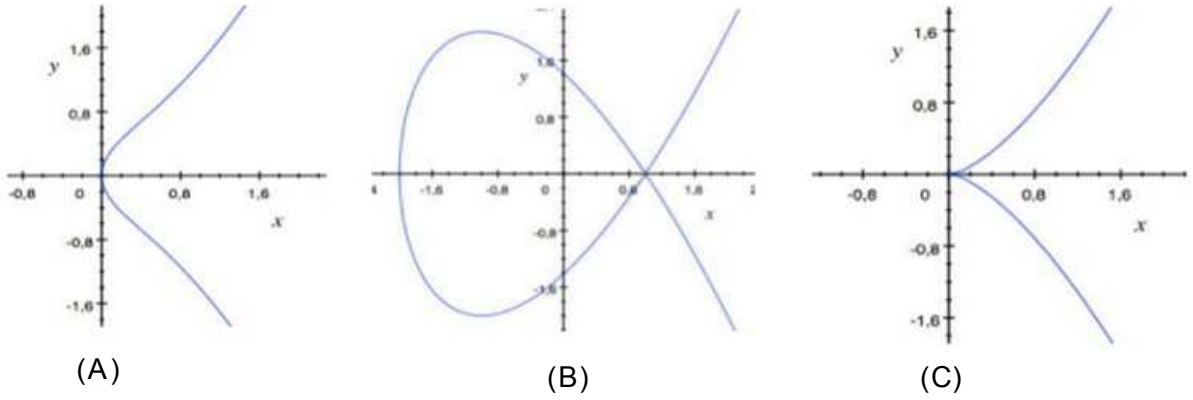

Figura 1: (A) Curva suave, (B) Curva singular e (C) Cúspide

uma reta tangente, ou seja, a curva é suave(Fig. (1)), não possui auto-interseções(curva singular) e nem cúspides (HANKERSON; MENEZES; VANSTONE, 2004).

O **conjunto dos pontos** que pertencem à curva elíptica E definida sobre um corpo $\mathbb{F}$ é descrito por:

$$E(\mathbb{F}) = \{(x, y) \in \mathbb{F} \times \mathbb{F} \mid y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\} \tag{12}$$

em que o ponto $\mathcal{O}$ é chamado de *ponto no infinito*. (SEET, 2007). Será visto na próxima seção que, juntamente com uma operação de soma dos pontos da curva elíptica, o conjunto $E(\mathbb{F})$ forma um grupo abeliano com o ponto no infinito sendo $\mathcal{O}$ o elemento identidade (HANKERSON; MENEZES; VANSTONE, 2004).

A Figura (2) apresenta alguns exemplos de curvas elípticas usando a forma normal da equação de Weierstrass.

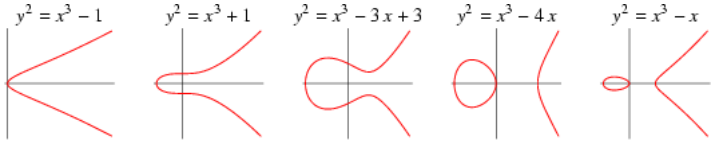

Figura 2: Exemplos de curvas elípticas

1.5.2 Leis de grupo para Curvas Elípticas

Seja o conjunto de pontos $E(\mathbb{F})$ da curva elíptica E definida sobre um corpo $\mathbb{F}$, é possível definir uma operação sobre esse conjunto de pontos, que será chamada de adição e indicada por $+$. Essa operação de adição e o conjunto de pontos $E(\mathbb{F})$ da curva elíptica formam um grupo abeliano (STALLINGS, 2011).

A forma mais clara de entender a operação de adição sobre curvas elípticas é geometricamente (ver Fig. (3)). Para somar dois pontos P e Q com coordenadas x diferentes, basta ligar uma linha reta entre eles e encontrar o terceiro ponto de interseção R. Pela natureza das curvas elípticas, pode-se observar que existe um único ponto R que é o ponto de interseção (a menos que os pontos P, Q possuam a mesma coordenada x e coordenadas y diferentes). Para formar uma estrutura de grupo, é preciso definir a adição sobre três pontos da seguinte forma: $P + Q = -R$. (STALLINGS, 2011)

Se as coordenadas x dos pontos P e Q são iguais e as coordenadas y são diferentes, então a reta que passa pelos dois pontos não intersecta a curva elíptica em nenhum outro ponto. Para definir a estrutura de grupo, considera-se que essa reta vertical intersecta a curva no ponto no infinito $\mathcal{O}$. (STALLINGS, 2011)

É possível calcular o dobro de um ponto P, ou seja, calcular a soma do ponto com ele mesmo. Para isso desenha-se a reta tangente à curva no ponto P. Essa linha intersecta a curva elíptica num segundo ponto, que será refletido para obter-se o ponto R, que é o resultado da soma. (HANKERSON; MENEZES; VANSTONE, 2004) Observe que esse caso se aplica quando se deseja somar os pontos P e Q em que $P = Q$.

Os axiomas de grupo são satisfeitos da seguinte forma:

1. Para quaisquer pontos $P, Q \in E(\mathbb{F})$, o ponto $P + Q \in E(\mathbb{F})$.
2. $\mathcal{O}$ é o elemento identidade. Logo, $P + \mathcal{O} = P$ para todo $P \in E(\mathbb{F})$.
3. Para qualquer ponto P, existe um ponto P' tal que $P + P' = \mathcal{O}$. O ponto P' é denominado *inverso aditivo* de P.
4. A adição é associativa. Logo, $P, Q, R \in E(\mathbb{F})$, observa-se que $(P + Q) + R = P + (Q + R)$
5. A adição é comutativa. Logo, $P, Q \in E(\mathbb{F})$, observa-se que $P + Q = Q + P$.

1.5.3 Curvas elípticas sobre corpo finito

A criptografia de curva elíptica utiliza curvas elípticas em que as variáveis e coeficientes são restritos a elementos de um corpo finito (STALLINGS, 2011), ou seja, sendo a curva elíptica E definida sobre o corpo finito $\mathbb{F}_p$, seus coeficientes assumem valores inteiros

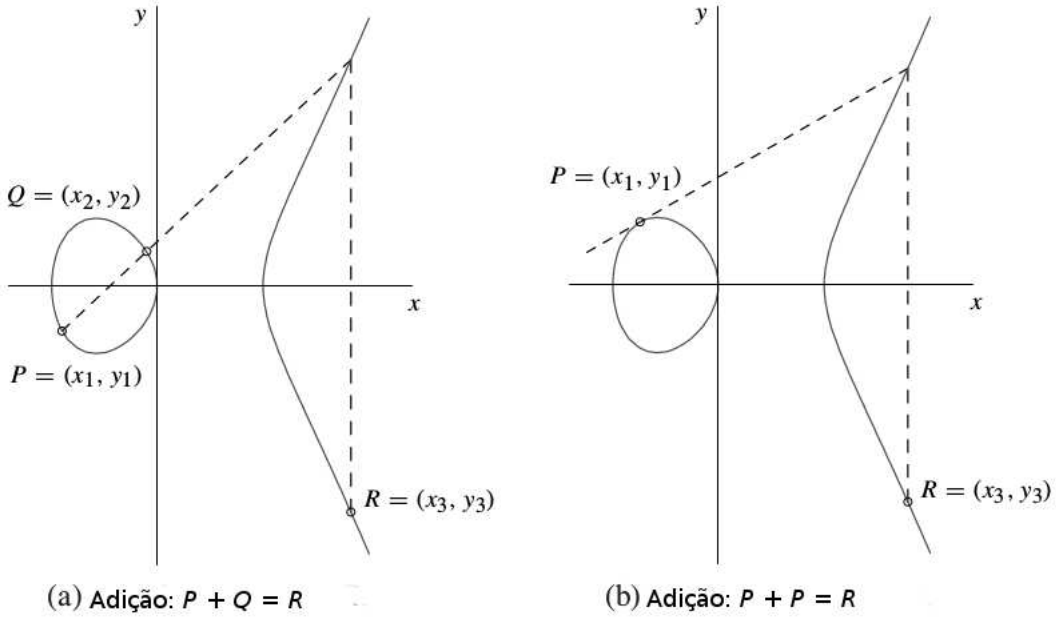

Figura 3: Soma de pontos em curvas elípticas

entre 0 e $p-1$, pois os cálculos são realizados módulo p. Quando definida sobre um corpo finito, a equação da curva deve indicar qual é esse corpo através da notação *mod*, como ocorre na Eq. (13), em que a curva está definida sobre um corpo finito $\mathbb{F}_p$.

$$y^2 \mod p = (x^3 + ax + b) \mod p \tag{13}$$

Pode-se mostrar que um grupo abeliano finito é definido com base no conjunto $E_p(a, b)$, desde que $(x^3 + ax + b) \mod p$ não tenha fatores repetidos. Isso é equivalente à condição

$$(4a^3 + 27b^2) \mod p \neq 0 \tag{14}$$

A Eq. (10) tem a mesma forma da Eq. (13). As regras para adição sobre $E_p(a, b)$ correspondem à técnica algébrica descrita para as curvas elípticas definidas sobre números reais. Para todos os pontos $P, Q \in E_p(a, b)$.

1. $P + \mathcal{O} = P$.
2. Se $P = (x_P, y_P)$, então $P + (x_P, -y_P) = \mathcal{O}$. O ponto $(x_P, -y_P)$ é o negativo de P, indicado como $-P$.

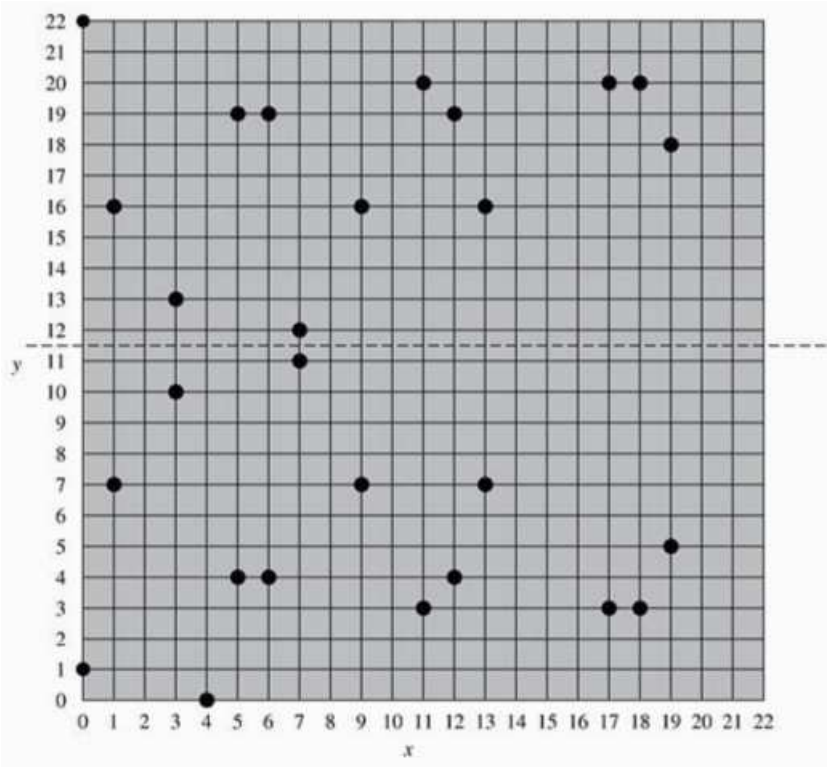

Figura 4: Curva elíptica $E_{23}(1,1)$

3. Se $P = (x_P, y_P)$ e $Q = (x_Q, y_Q)$ com $P \neq -Q$, então $R = P + Q = (x_R, y_R)$ é determinado pelas seguintes regras:

$$x_R = (\lambda^2 - x_P - x_Q) \mod p \tag{15}$$

$$y_R = (\lambda(x_P - x_R) - y_P) \mod p \tag{16}$$

em que

$$\lambda = \begin{cases} \left(\dfrac{y_Q - y_P}{x_Q - x_P}\right) & \mod p, \text{ se } P \neq Q \\ \\ \left(\dfrac{3x_P^2 + a}{2y_P}\right) & \mod p, \text{ se } P = Q \end{cases} \tag{17}$$

4. A multiplicação é definida como adição repetida; por exemplo, $4P = P+P+P+P$.

Por exemplo, para somar os pontos $P(6,19)$ e $(9,16)$, pertencentes à curva $E_{23}(1,1)$ descrita na Fig. (4), basta substituir os valores $x_P = 6$, $y_P = 19$, $x_Q = 9$, $y_Q = 16$, $a = 1$ e $p = 23$ nas Eqs. (15) , (15) e (17) para obter como resultado da soma o ponto $R = (9,7)$. Para multiplicar o ponto $P(6,19)$ por 2, ou seja, ao somá-lo com ele mesmo, obtém-se como resultado $2P = (13,16)$.

1.5.4 Quantidade de pontos em uma curva elíptica

Seja a curva elíptica E definida sobre o corpo finito $\mathbb{F}_p$. O conjunto de todos os pontos que pertencem à curva elíptica é representado por $E(\mathbb{F}_p)$ (ver 12). A quantidade de pontos que compõe esse conjunto é a **ordem de** E **sobre** $\mathbb{F}_p$ e será representada pela notação $\#E(\mathbb{F}_p)$. (HANKERSON; MENEZES; VANSTONE, 2004)

A ordem da curva elíptica é importante para o sistema criptográfico baseado em curvas elípticas, pois sua segurança depende dos fatores primos da ordem (ALVARADO, 2005). Se a ordem da curva elíptica pode ser fatorada em números primos pequenos, a sua segurança pode ser facilmente quebrada utilizando o algoritmo de Pohlig-Hellman, que não será abordado em profundidade nesse trabalho.

O teorema de Hasse fornece limites para a ordem de uma curva elíptica. Esse teorema determina que, sendo a curva elíptica E definida sobre $\mathbb{F}_p$, então

$$p + 1 - 2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p} \tag{18}$$

O **traço** de E sobre $\mathbb{F}_p$ é $t = 2\sqrt{p}$. Sabendo-se o traço da curva definida sobre $\mathbb{F}_p$, é possível calcular a sua ordem. O algoritmo de Schoof (ver B) calcula o traço de uma curva, permitindo saber a sua ordem.

1.5.5 Criptografia de curvas elípticas

Primeiro, selecione um inteiro grande p que seja primo e parâmetros da curva elíptica a e b de acordo com a Eq. (9) ou (10). Isso define o grupo de pontos $E_p(a, b)$. Em seguida escolha um ponto base $G \in E_p(a, b)$, cuja a ordem seja um valor muito grande n. $E_p(a, b)$ e G são os parâmetros do criptossistema, conhecidos por todos os participantes.

Um acordo de chaves entre os usuários **A** e **B** pode ser realizado da seguinte maneira: (STALLINGS, 2011)

1. **A** seleciona um inteiro $n_A < n$. Essa é chave privada de **A**, então **A** gera sua chave pública $P_A = n_A \times G$

2. Do mesmo modo, **B** seleciona um inteiro $n_B < n$, sendo essa a chave privada de **B**. E então **B** calcula e divulga sua chave pública $P_B = n_B \times G$.

3. **A** e **B** agora podem compartilhar o mesmo ponto da curva elíptica, que será chamado de segredo compartilhado entre **A** e **B**. **A** obtém o segredo compartilhado através do cálculo $K = n_A \times P_B$. Da mesma forma, **B** obtém o segredo compartilhado através do cálculo $K = n_B \times P_A$

É possível verificar que **A** e **B** chegam ao mesmo segredo compartilhado da seguinte forma

$$n_A \times P_B = n_A \times (n_B \times G) = n_B \times (n_A \times G) = n_B \times P_A$$

Para quebrar esse esquema, um atacante precisa utilizar a curva $E_p(a, b)$ e o ponto base G, que são conhecidos, e uma das chaves públicas P_A ou P_B para descobrir respectivamente uma das chaves privadas n_A ou n_B. Para isso, ele precisará encontrar um valor x que satisfaça $P_A = xG$, encontrando assim $n_A = x$, ou $P_B = xG$, encontrando $n_B = x$. Este é um problema computacionalmente difícil de resolver, e é conhecido como o *Problema do logarítmo discreto sobre curvas elípticas.*

2 Criptoanálise

Enquanto o campo da Criptografia procura desenvolver um sistema ou um protocolo que garanta a privacidade na troca de mensagens, o campo da Criptoanálise é o estudo de métodos para quebrar sistemas de criptografia.

Existem dois lados da Criptoanálise: a que procura criar uma criptografia segura e a que se propõe a quebrar a segurança de um criptossistema. Embora possam ser vistos como antagonistas, o conhecimento de que um criptossistema é suscetível à ataques pode ser uma vantagem. O princípio de Kerchoff's diz que a segurança de um criptossistema deve depender apenas do sigilo do espaço da chave K [1], e não do sigilo do algoritmo de criptografia. (SEET, 2007)

2.1 Criptoanálise no contexto de curvas elípticas

No caso de um criptossistema de curva elíptica, ela deverá depender do sigilo dos inteiros n_A e n_B, que são as chaves privadas de **A** e **B** mencionados na Seção 1.5.5. Se estes inteiros são facilmente inferidos a partir de execuções dos protocolos ou do próprio conhecimento da chave pública, então o criptossistema não é mais seguro. Esse tipo de descoberta é classificado como ruptura total, no qual o atacante deduz as chaves secretas. (KNUDSEN, 1998)

2.1.1 O problema do logaritmo discreto sobre curvas elípticas

Seja E uma curva elíptica sobre o corpo finito $\mathbb{F}_p$ e seja P e Q pontos em $E(\mathbb{F}_p)$. O problema do logaritmo discreto sobre curvas elípticas (ECDLP - *Elliptic Curve Discrete Logarithm Problem*) é o problema de encontrar um inteiro x tal que $Q = xP$. Pela analogia com o problema do logaritmo para $\mathbb{F}_p$, denotamos esse inteiro x por

$$xP = Q \tag{1}$$
$$x = \log_P(Q) \tag{2}$$

e chamamos x como logaritmo discreto elíptico de Q em relação a P. (HOFFSTEIN, 2008) Encontrar o valor de x neste contexto exige muito poder e tempo de processamento, sendo algo computacionalmente caro de se realizar. Um valor muito grande de x inviabiliza análise de força bruta.

[1] Uma chave de tamanho m oferece um espaço de 2^m.

2.1.2 Algoritmos conhecidos para resolver ECDLP

Ataques mais conhecidos sobre ECC têm complexidade exponencial. Essa afirmação é válida para curvas genéricas e exclui ataques em subclasses especiais, como curvas super-singular e anômalas. A solução para a Eq. (2) pode ser calculada usando as seguintes técnicas: (PELZL, 2006)

- Força-bruta: Este método adiciona sequencialmente o ponto $P \in E(\mathbb{F}_p)$ a ele mesmo. A cadeia de adição $P, 2P, 3P, 4P, ...$ acabará por chegar em Q descobrindo assim o valor de n, de acordo com a Eq. (1). No pior caso, este cálculo pode levar até $n-1$ passos em que n é da ordem de P. Desta forma, o ataque pode se tornar inviável para prática quando n é muito grande.

- *Baby Step Giant Step* (BSGS): O algoritmo BSGS é uma melhoria à busca por força-bruta. Para n na ordem de P, é necessário que o algoritmo utilize um quantidade de memória temporária de $\sqrt{n}$ e um adicional de, aproximadamente, $\sqrt{n}$ passos. No entanto, devido à sua complexidade de memória alta, BSGS acaba não sendo muito interessante.

- Pohlig-Hellman: trabalha com a fatoração do valor n da ordem da curva e utiliza o Teorema do Resto Chinês para o cálculo da série de equações lineares gerados com os fatores de n. Esta abordagem pode ser implementada em conjunto com Pollard's rho. No entanto, a escolha de uma curva cuja ordem é um número primo pode inibir esse tipo de ataque, pois não é possível fatorar números primos.

- Pollard's rho: O ataque Pollard's rho ou Pollard-rho foi proposto por J. Pollard em 1978. Este ataque consiste em um algoritmo baseado em colisão com base em um percurso aleatório num grupo cíclico, assim, pode ser aplicado ao grupo de pontos gerados por P numa curva elíptica. O percurso aleatório calcula uma trilha de pontos numa curva elíptica e eventualmente termina em um ciclo, revelando a solução para ECDLP. (POLLARD, 1978) Embora tendo um tempo de complexidade similar de $\sqrt{\pi n/2}$ comparado ao BSGS, Pollard-rho é superior devido aos seus desprezíveis requisitos de memória. Em combinação com a paralelização adequada, o método Pollard-rho é o mais rápido ataque conhecido contra ECC. (PELZL, 2006)

  Neste trabalho serão apresentados os métodos Pollard-rho original e suas variantes Pollard-rho com único processador (SPPR), Pollard-rho com multiprocessadores (MPPR) e Pollard-rho com automorfismo.

2.1.3 Quebrando ECC em um ano

De acordo com uma pesquisa (PELZL, 2006), definindo um limite de tempo específico, pode-se calcular o número de processadores necessários. Seguindo a tentativa de

Tabela 1: Número de processadores necessários para ataque de um ano

k	**Pentium M**	**XC3S1000**	**ASIC**
80	1	1	-
96	56	21	-
128	$4.86 \cdot 10^6$	$2.55 \cdot 10^6$	$2.05 \cdot 10^4$
160	$3.78 \cdot 10^{11}$	$3.1 \cdot 10^{11}$	$2.48 \cdot 10^9$

quebrar ECC com uma chave de tamanho k de bits em um tempo máximo de um ano (365 dias), é possível encontrar a quantidade de processadores necessários representadas pela Tab. (7).

Seriam necessários cerca de 378 bilhões de processadores de propósito geral para completar um ataque de sucesso em um ano em um ECC com chave de $k = 160$ bits. Com hardwares mais preparados utilizando diferentes arquiteturas, levaria menos tempo. No caso do XC3S1000 não se nota grandes diferenças, no entanto, utilizando ASIC é preciso cerca de 100 vezes menos processadores para completar a tarefa. Obviamente, a construção de um cenário tão grande é inviável atualmente.

3 Pollard-rho para resolver ECDLP

Em 1978, Pollard sugeriu o método Monte-Carlo[1] para resolver o problema do logaritmo discreto. Desde então, o método foi modificado para resolver o ECDLP. Como o algoritmo Pollard-rho é atualmente o algoritmo mais rápido para resolver o ECDLP, então a segurança do ECC depende da eficiência desse algoritmo. Teoricamente, se o algoritmo Pollard-rho é capaz de resolver o ECDLP eficientemente e em um tempo relativamente curto, então o criptossistema estará inseguro. (SEET, 2007)

A estratégia do algoritmo é produzir uma sequência de termos gerados aleatoriamente (c_k, d_k, R_k), em que R_k é um ponto na curva E e c_k e d_k estão em $\mathbb{F}_p$ sobre a qual a curva elíptica E está definida. Como $E(\mathbb{F}_p)$ é um grupo finito, a sequência eventualmente irá se tornar periódica e voltará para um termo anterior da sequência – tal ocorrência é chamada de *colisão*. Pelo paradoxo do aniversário [2], o número de iterações médio para que se obtenha uma colisão é de aproximadamente $\sqrt{\pi n/2} \approx 1.2533\sqrt{n}$. Usa-se essa periodicidade para resolver ECDLP. Como nem sempre a sequência volta para o primeiro termo, um diagrama da sequência parecerá com a letra grega ρ (ver Fig. (5)). Por este motivo esse método é chamado de Pollard-rho. Essa sequência de termos gerados é chamada de "percurso". (HANKERSON; MENEZES; VANSTONE, 2004)

Seja μ o tamanho da cauda e λ o tamanho do ciclo. Após um número finito de iterações, obtêm-se os termos $R_k = R_{k+\lambda}$, em que $k > \mu$ e $\lambda > 1$. Neste ponto, já deverá ter sido encontrado uma correspondência entre os termos e poderá empregar resultados de matemática discreta para resolver os problemas de logaritmo discreto gerado pelos elementos do conjunto finito.

A seguir serão apresentados o algoritmo de Pollard-rho e suas variações.

3.1 Pollard-rho original

Uma forma simplificada de encontrar uma colisão seria selecionar inteiros aleatórios $c, d \in [0, n-1]$ e armazenar as triplas $(c, d, cP+dQ)$ em uma tabela ordenada pelo terceiro

[1] Monte-Carlo é um método estatístico que se baseiam em amostragens aleatórias massivas para obter resultados numéricos, isto é, repetindo sucessivas simulações um elevado número de vezes, para calcular probabilidades de forma heurística.

[2] Suponha que uma urna tenha n bolas numeradas de 1 a n. As bolas são aleatoriamente retiradas uma de cada vez e colocadas de volta na urna. Portanto, o número esperado para que as bolas se repitam é de aproximadamente de $\sqrt{\pi n/2}$. Se $n = 365$ e as bolas representam diferentes dias do ano, então pode-se dizer que o número esperado de pessoas que tem de ser reunidas em uma sala para que pelo menos duas delas tenham nascido no mesmo dia é de aproximadamente $\sqrt{\pi 365/2} \approx 24$. Esse número é surpreendentemente pequeno e, consequentemente, daí vem a nomenclatura "paradoxo do aniversário". (HANKERSON; MENEZES; VANSTONE, 2004)

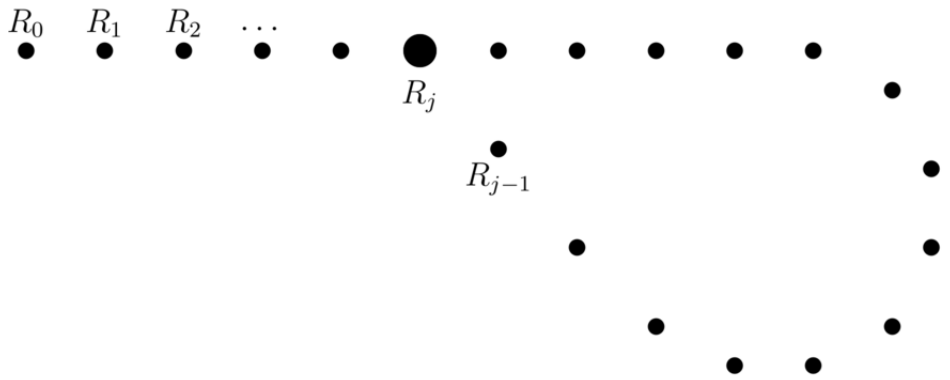

Figura 5: Diagrama da sequência produzida pelo algoritmo Pollard-rho

componente até que um ponto $cP+dQ$ seja obtido pela segunda vez. A desvantagem desse algoritmo é que a capacidade de armazenamento necessária para o cálculo é de $\sqrt{\pi n/2}$ triplas.

Com o algoritmo Pollard-rho, é possível encontrar os pares (c_m, d_m) e (c_{2m}, d_{2m}) aproximadamente no mesmo tempo que o algoritmo citado acima, mas tem desprezíveis requisitos de armazenamento. A ideia é definir uma função de iteração $f : \langle P\rangle \to \langle P\rangle$ de modo que $X \in \langle P\rangle$ e $a, b \in [0, n-1]$ com $X = aP + bQ$. Além disso, f deve ter características de uma função aleatória. (HANKERSON; MENEZES; VANSTONE, 2004)

Seja um grupo $G = E(\mathbb{F}_p)$, tal que a ordem de $\#E(\mathbb{F}_p) = |G| = n$, e P e $Q \in E(\mathbb{F}_p)$ tal que $Q = xP$ em G. O objetivo é calcular x. Segue os passos do algoritmo original proposto por Pollard:

1. G é particionado em 3 conjuntos S_1, S_2, S_3 de aproximadamente do mesmo tamanho, em que $\mathcal{O} \notin S_2$.

2. Definir uma função de iteração $f : R \to R$ de um percurso aleatório:

$$R_{k+1} = f(R_k) = \begin{cases} Q + R_k, & R_k \in S_1 \\ 2R_k, & R_k \in S_2 \\ P + R_k, & R_k \in S_3 \end{cases} \tag{1}$$

3. Seja $R_k = c_k P + d_k Q$, e portanto

$$c_{k+1} = \begin{cases} c_k, & R_k \in S_1 \\ 2c_k \pmod n, & R_k \in S_2 \\ c_k + 1, & R_k \in S_3 \end{cases} \tag{2}$$

e

$$d_{k+1} = \begin{cases} d_k + 1, & R_k \in S_1 \\ 2d_k \pmod{n}, & R_k \in S_2 \\ d_k, & R_k \in S_3 \end{cases} \tag{3}$$

4. Os termos iniciais são: $R_0 = P, c_0 = 1, d_0 = 0$ e os pares gerados (R_k, R_{2k}) até encontrar uma correspondência $R_m = R_{2m}$, para qualquer m.

5. Uma vez encontrados, calcule:

$$R_m = c_m P + d_m Q$$
$$R_{2_m} = c_{2m} P + d_{2m} Q$$

6. Com isso, é possível calcular x:

$$x = \frac{c_{2m} - c_m}{d_m - d_{2m}} \pmod{n} \tag{4}$$

O termo $(d_m - d_{2m})^{-1} \pmod{n}$ em (4) somente é possível calcular quando o $\gcd(d_m - d_{2m}, n) = 1$, ou seja, quando d_m e d_{2m} são coprimos entre si. Caso contrário, o algoritmo retorna um valor indefinido.

3.2 Pollard-rho com único processador

A função de iteração proposta por Pollard no algoritmo descrito na Seção 3.1 particionava o conjunto $G = E(\mathbb{F}_p)$ em $L = 3$ subconjuntos S_1, S_2, S_3. Alguns estudos demonstram que essa função de iteração proposta por Pollard não é suficientemente aleatória, e consequentemente, demora para encontrar uma colisão. Uma alternativa encontrada foi particionar o conjunto $G = E(\mathbb{F}_p)$ em vários subconjuntos S_L, por exemplo, $L = 100$, em vez de apenas três, como fora proposto por Pollard. (LAPORTA; PIZZIRANI, 2014).

Seja $\{S_1, S_2, ..., S_L\}$ uma partição aleatória de $\langle P \rangle$ em uma quantidade L de conjuntos de aproximadamente do mesmo tamanho. Então um ponto $X \in \langle P \rangle$ pode ser designado a S_j se os cinco *bits* menos significantes da coordenada x de X representam o inteiro $j - 1$. Desta forma, define-se $j = H(X)$ se $X \in S_j$, no qual H é uma função de partição. Finalmente, seja $c_j, d_j \in [0, n-1]$ para $1 \leq j \leq L$. Então $f : \langle P \rangle \to \langle P \rangle$ é definido por

$$f(X) = X + c_j P + d_j Q, \text{ em que } j = H(X)$$

O pseudoalgoritmo está descrito em anexo (A).

3.2.1 Algoritmo Busca-Ciclos de Floyd

Pollard-rho é uma variante do algoritmo busca-ciclos de Floyd (*Floyd's cycle-finding*), que é importante para detectar a ocorrência de um ciclo em sequências arbitrárias, seja em estruturas de dados ou geradas por sequências pseudo-aleatórias e grafos. Este algoritmo, também é chamado de algoritmo "lebre e tartaruga", no qual se utiliza apenas dois ponteiros que se movem através da sequência com diferentes velocidades.

Para encontrar $R_i = R_j$, o algoritmo calcula as triplas (c_i, d_i, R_i) e (c_{2i}, d_{2i}, R_{2i}) até que $R_i = R_{2i}$. Para cada iteração, calcula-se $R_{i+1} = f(R_i)$ e $R_{2(i+1)} = f(f(R_{2i}))$, o que significa que este algoritmo utiliza uma quantidade mínima de armazenamento. O algoritmo busca-ciclos de Floyd é baseado na seguinte ideia. (WANG; ZHANG, 2011) (BAI; BRENT, 2008)

Teorema 1 ((KNUTH, 1997)). Para uma sequência periódica $\{R_0, R_1, R_2, ...\}$, existe um $i > 0$ tal que $R_i = R_{2i}$ e o menor valor de i que pertença ao intervalo $\mu \leq i \leq \mu+\lambda$. μ e λ são o pré-período (cauda) e o período (ciclo) da sequência R_i, respectivamente.

O melhor caso desse algoritmo são necessários μ iterações e o pior caso $\mu + \lambda$ iterações. Sob a suposição de que $f : G \to G$ se comporta como uma função verdadeiramente aleatória, o número esperado de iterações antes que se encontre uma colisão é de $\sqrt{\pi^5|G|/288} \approx 1.03\sqrt{|G|}$ (BAI; BRENT, 2008). O ponto chave para esse algoritmo é que precisa-se de três operações de grupo e uma comparação para cada iteração, o que o torna ineficiente.

3.3 Pollard-rho multiprocessadores

Suponha agora que M processadores estão disponíveis para resolver uma instância de ECDLP. Uma abordagem normal seria de executar o algoritmo Pollard-rho independente para cada processador (com diferentes pontos X_0 aleatórios escolhidos inicialmente) até que algum dos processadores finalize. Uma análise cuidadosa mostra que o número esperado de operações de curva elíptica executadas por cada processador até que algum finalize é de $3\sqrt{n/M}$. Assim a aceleração esperada é dada pelo fator $\sqrt{M}$. (HANKERSON; MENEZES; VANSTONE, 2004)

Van Oorschot e Wiener propuseram uma variante do algoritmo Pollard-rho que produz um fator de aceleração M quando M processadores são empregados. A ideia é permitir que as sequências $\{X_i\}_{i\geq 0}$ geradas pelos vários processadores possam colidir umas

com as outras. Ou seja, cada processador escolhe aleatoriamente seu próprio ponto inicial X_0, mas todos processadores utilizam a mesma função de iteração f para calcular os pontos subsequentes X_i. Desta forma, se as sequências de dois processadores diferentes colidirem entre si, como ilustrado na Fig. (6), as duas sequências serão idênticas daquele ponto em diante. A sequência gerada pelos processadores 3 e 4 se colidem em X. O algoritmo informa a colisão em Y, o primeiro ponto distinto subsequente (OORSCHOT; WIENER, 1996).

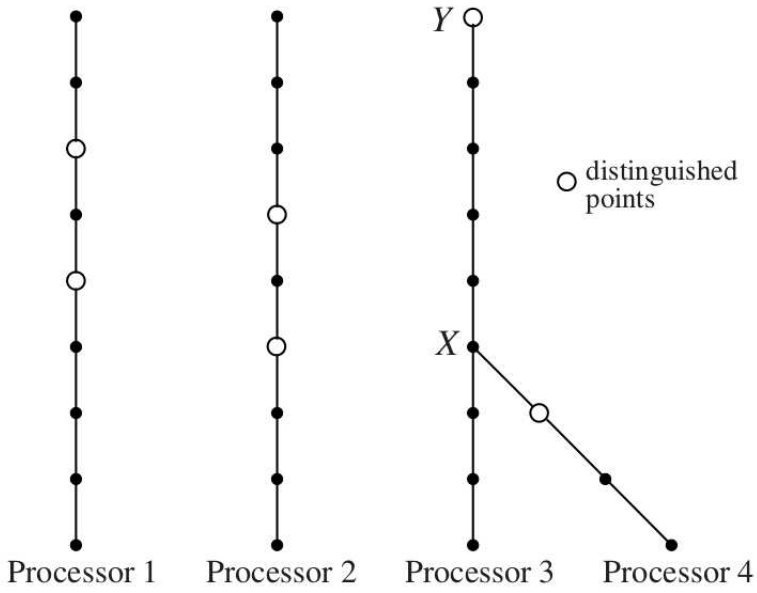

Figura 6: A sequência gerada pelo algoritmo Pollard-rho paralelizado.

O algoritmo busca-ciclos de Floyd – descrita na Seção 3.2.1 – encontra uma colisão na sequência gerada por um único processador. A seguinte estratégia possibilita uma procura eficiente de uma colisão nas sequências geradas por diferentes processadores. Uma *propriedade distintiva* facilmente testável de pontos é selecionada. Por exemplo, um ponto deve ser *distinto* se os primeiros t *bits* de sua coordenada x são iguais a zero. Seja θ a proporção em $\langle P \rangle$ tendo essa propriedade distintiva. Sempre que um processador encontra um ponto distinto, ele transmite o ponto a um servidor central que o armazena em uma lista ordenada. Quando o servidor recebe o mesmo ponto distinto pela segunda vez, ele calcula o logaritmo discreto desejado pela Eq. (4) e termina a execução de todos os processadores. O número esperado de passos por processador antes de uma colisão ocorrer é de $(\sqrt{\pi n/2})/M$. Um ponto distinto subsequente é esperado para após $1/\theta$ passos. Consequentemente o número médio (dado um número muito grande de tentativas) de operações de curva elíptica desempenhadas por cada processador antes de uma colisão de

pontos distintos é de

$$\frac{1}{M}\sqrt{\frac{\pi n}{2}} + \frac{1}{\theta} \tag{5}$$

A sua versão paralelizada do algoritmo Pollard-rho obtém um aumento de velocidade que é linear em relação à quantidade de processadores empregados. Uma observação que deve ser feita é que os processadores não tem de comunicar-se entre si, e além disso tem uma comunicação limitada com o servidor central. Portanto, o espaço total necessário no servidor pode ser controlado com uma seleção cautelosa da propriedade distintiva.

O pseudoalgoritmo está descrito em anexo (B).

3.3.1 Propriedade distintiva

Atualmente, o método do ponto distinto é o algoritmo mais eficiente para detectar um ciclo pseudo-aleatório quando n é grande. Para quebrar ECC2K-130, por exemplo, (BAILEY L. BATINA, 2009) define a propriedade distintiva como o *Hamming weight*[3] de uma representação-base normal da coordenada x do ponto que seja menor ou igual a 34. Note que este tipo de definição permite uma rápida verificação para a propriedade distintiva.

3.4 Pollard-rho com automorfismo

Uma maneira descrita por Hankerson, Menezes e Vanstone para acelerar o algoritmo Pollard-rho é fazendo uso de automorfismo.(HANKERSON; MENEZES; VANSTONE, 2004)

Seja o grupo de pontos da curva elíptica $E(\mathbb{F}_q)$. Considere o ponto P de ordem n e o subgrupo gerado por esse ponto, ou seja, $\langle P\rangle$. Seja o automorfismo $\psi : \langle P\rangle \to \langle P\rangle$. A ordem da função ψ é o menor inteiro positivo t tal que $\psi^t(R) = R$ para todo ponto $R \in \langle P\rangle$. Por exemplo, a função $f(x) = -x$ tem ordem $t = 2$, pois $f(f(x)) = x$.

É possível definir uma relação de equivalência $R_1 \sim R_2$ se e somente se $R_1 = \psi^j(R_2)$ para algum $j \in [0, t-1]$. Daí cria-se a classe de equivalência $[R]$, que é descrita por

$$[R] = \{R, \psi(R), \psi^2(R), \ldots, \psi^{l-1}(R)\}$$

em que l é o menor divisor positivo da ordem t da função ψ, tal que $\psi^l(R) = R$.

O método de Pollard-rho com automorfismo consiste em utilizar uma função iterativa f que seja definida nas classes de equivalência. Para isso, será definido um repre-

[3] *Hamming weight* de uma *string* é o número de símbolos que são diferentes do símbolo-zero do alfabeto utilizado. No caso de uma *string* binária, é a quantidade de *bits* que são iguais a 1.

sentante $\overline{R}$ para cada classe de equivalência $[R]$ e uma função g tal que

$$g(R) = f(\overline{R})$$

Sendo conhecido um inteiro $\lambda \in [0, n-1]$ tal que $\psi(P) = \lambda P$ e os inteiros a, b tal que $X = aP + bQ$, então pode-se calcular $\overline{X} = \overline{a}P + \overline{b}Q$ eficientemente por $\overline{a} = \lambda^j a \bmod n$ e $\overline{b} = \lambda^j b \bmod n$.

A função $g(R)$ é utilizada como função de iteração para o algoritmo Pollard-rho paralelizado. Essa modificação garante ao algoritmo uma melhoria no tempo de execução do Pollard-rho paralelizado, dessa forma, a Eq. (5) passa a ser

$$\frac{1}{M}\sqrt{\frac{\pi n}{2t}} + \frac{1}{\theta}$$

ou seja, o uso da função $g(R)$ acarreta uma melhoria no tempo de execução do algoritmo por um fator de $\sqrt{t}$. (HANKERSON; MENEZES; VANSTONE, 2004)

4 Metodologia

Metodologias específicas colaboram para estabelecer diretrizes e boas práticas na condução do trabalho, conferindo padronização, noções de pesquisa científica, dentre outras contribuições. (WOHLIN, 2000)

Com o intuito de guiar a pesquisa de forma adequada, esse capítulo aborda sobre os diversos tipos de metodologias de pesquisa, de modo a definir qual se adequa melhor ao projeto.

4.1 Classificação da pesquisa

A seguir serão apresentados os grupos de classificação de pesquisa, quanto à natureza da pesquisa, abordagem do problema, objetivos de uma pesquisa e procedimentos técnicos.

- Natureza da pesquisa:
  - **Pesquisa básica:** Possui o objetivo de gerar novos conhecimentos para ciência. Neste tipo de pesquisa não é obrigatório que o conhecimento gere um uso prático (TAFNER; SILVA, 2007).
  - **Pesquisa Aplicada:** Visa gerar uma maior compreensão para assuntos práticos dirigidos à solução de problemas específicos (TAFNER; SILVA, 2007).
- Abordagem do problema:
  - **Pesquisa Quantitativa:** O estudo quantitativo considera que tudo pode ser quantificável, ou seja, o estudo que pode ser traduzido em números e requer uso de técnicas estatísticas para sua análise (TRAVASSOS, 2002).
  - **Pesquisa Qualitativa:** O estudo qualitativo é descritivo, sendo assim, não pode ser analisada de forma mensurável e sim indutivamente (TRAVASSOS, 2002).
- Objetivos de uma pesquisa:
  - **Pesquisa Exploratória:** é utilizada pelo pesquisador para se familiarizar com um assunto pouco explorado. Ao decorrer ou no final da pesquisa exploratória, o pesquisador poderá estar apto para formular hipóteses (GIL, 2008).

  - **Pesquisa Descritiva:** é usada quando se tem um conhecimento do assunto e se quer descrever um fenômeno. Hipóteses podem ser formuladas com base em conhecimentos prévios, procurando confirmá-las ou negá-las (GIL, 2008).
  - **Pesquisa Explicativa:** é classificada com base em procedimentos técnicos, podendo ser quantitativa ou qualitativa. A pesquisa quantitativa traduz em números os estudos realizados e se utiliza técnicas estatísticas para comprovar os fatos (GIL, 2008).

- Procedimentos técnicos:

  - **Pesquisa bibliográfica:** é realizada a partir do levantamento de referências teóricas sobre o tema. De forma geral, toda pesquisa inicia-se como uma pesquisa bibliográfica, mas há aquelas que dependem exclusivamente desse tipo de pesquisa. Em essência, a conclusão desse tipo de pesquisa é uma compilação das publicações referentes ao tema (TAFNER; SILVA, 2007).
  - **Pesquisa documental:** semelhante à pesquisa bibliográfica, diferencia-se pela natureza das fontes. Tem como base documentos sem tratamento analítico, como tabelas, cartas, fotos, pinturas, dentre outros (GIL, 2008).
  - **Pesquisa experimental:** seleciona grupos de assuntos coincidentes e submete-os a tratamentos diferentes, com isso, ocorre uma verificação se há variáveis estranhas e checa-se as diferenças observadas nas respostas e se são estatisticamente significantes (TAFNER; SILVA, 2007).
  - **Pesquisa *ex-post facto*:** a tradução literal é "de um fato passado", ou seja, a pesquisa *ex-post facto* é realizada após a ocorrência de variáveis no objeto de estudo. A principal característica deste tipo de pesquisa é o fato de os dados serem coletados após a ocorrência dos eventos. Assim, investiga possíveis relações de causa e efeito entre um determinado fato identificado pelo pesquisador e um fenômeno que ocorre posteriormente (GIL, 2008).
  - **Levantamento (*survey*):** é uma investigação realizada em retrospecto, que em seguida, mediante análise quantitativa, chega às conclusões correspondentes aos dados coletados. O levantamento feito com informações de todos os integrantes do universo da pesquisa origina um censo (TRAVASSOS, 2002).
  - **Estudo de caso:** o estudo de caso busca o aprofundamento nas questões propostas, estudando um único grupo ou comunidade, utilizando mais a observação direta do que a interrogação, captando as explicações e interpretações do que ocorre no grupo. No estudo de campo, o pesquisador está inserido no grupo, para poder entender melhor as regras, os costumes e as convenções (TRAVASSOS, 2002).

4.2 Metodologia de pesquisa

Analisando-se o tema proposto, observa-se uma natureza de pesquisa aplicada. Este trabalho de conclusão de curso visa a implementação do algoritmo Pollard-rho e suas variações, carregando consigo diversos conhecimentos de assuntos práticos e gerando uma solução para um problema específico, no caso, o problema do logaritmo discreto no contexto de curvas elípticas.

Considerando os objetivos de estudo deste trabalho, foi incorporada uma abordagem explicativa afim de apresentar valores quantitativos que demonstrem a relação entre os algoritmos implementados e suas diferenças, além de uma análise dos dados e resultados utilizando de técnicas com o intuito de comprovar fatos inerentes à segurança de criptografia de curvas elípticas.

Dessa forma, este trabalho classifica-se como uma pesquisa:

- **Quanto à natureza**, este trabalho classifica-se como uma **pesquisa aplicada**.
- **Quanto aos objetivos da pesquisa**, este trabalho pode ser classificado como uma **pesquisa explicativa**.
- **Quanto à abordagem do problema**, este trabalho pode ser classificado como uma **pesquisa quantitativa**.
- **Quanto aos procedimentos técnicos**, foram utilizados a **pesquisa bibliográfica** e a **pesquisa experimental**.

4.3 Atividades de projeto

Os seguintes pacotes de trabalho, não necessariamente nessa ordem, são fundamentais para compor este projeto de monografia:

- **Pesquisa:** Levantamento e pesquisa acerca das propriedades matemáticas da curva elíptica e potenciais algoritmos que desafiam sua segurança.
- **Levantamento Bibliográfico:** Levantamento bibliográfico sobre os aspectos teóricos e práticos que envolvem a segurança das curvas elípticas.
- **Implementação:** Implementação do algoritmo Pollard-rho e suas variações.
- **Execução:** Execução e ajustes dos algoritmos, assim como os testes de suas execuções.
- **Parte escrita:** Escrita da parte textual do trabalho.

4.4 Ferramentas

A seguir serão apresentadas as ferramentas utilizadas e o ambiente de desenvolvimento, desde as configurações físicas do computador ao software utilizado.

As especificações técnicas do computador utilizado no desenvolvimento são descritas na Tab. (2), a listagem dos software empregados no ambiente de desenvolvimento são descritas na Tab. (3) e, por fim, as bibliotecas e ferramentas de apoio são descritas na Tab. (4).

Tabela 2: Configuração de hardware do computador de desenvolvimento.

Item	**Especificação**
Descrição	Servidor 64 bits
Processador	Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz (8 cores)
Memória	16 GiB SODIMM DDR3 Síncrono 1600 MHz (0,6 ns)
Armazenamento	ATA Disk 100 MB

Tabela 3: Ferramentas de desenvolvimento empregadas no processo de experimentação.

Ferramenta	**Nome**	**Versão**	**Comentário**
Sistema Operacional	Debian Linux	8.3	-
Linguagem	C++	11	O padrão C++11 foi escolhido por possuir mecanismos na própria linguagem que facilitam o desenvolvimento.
Compilador	GCC	3.6.1	A saída de compilação é melhor estruturada e é compatível com os argumentos das ferramentas GNU.
Linguagem	Python	2.7	Essa linguagem e versão foram escolhidas por terem suporte à operações com números grandes e à biblioteca SageMath.
Builder	GNU Make	4.1	-
Editores de Texto	VIM, Sublime Text	7.4, 3114	-

4.5 Geração de Curvas

Não é propósito deste trabalho realizar a implementação da geração de curvas elípticas e o cálculo da ordem dessas curvas. Para isso foi utilizado a biblioteca SageMath, que supre essas necessidades. Esta é uma biblioteca matemática *open-source* escrita em

Tabela 4: Bibliotecas e ferramentas utilizadas no desenvolvimento

Nome	**Versão**	**Comentário**
SageMath	7.2	Biblioteca matemática escrita em Python que provê operações para curvas elípticas.
smtplib	2.6	Biblioteca para lidar com envio de e-mails

Python por uma comunidade de programadores e matemáticos, que buscam uma alternativa para os principais sistemas proprietários de software matemático como Magma, Maple, Mathematica e Matlab. SageMath é um projeto que fornece vários conceitos como teoria dos números, teoria dos grafos, combinatória, álgebra, criptografia, matemática aplicada, estatística, cálculo simbólico, etc. Para curvas elípticas, esta biblioteca implementa um rápido algoritmo de contagem de pontos (algoritmo Schoof-Elkies-Atkin), o que é bem útil na realização deste trabalho.

5 Resultados

Neste capítulo serão apresentados os resultados obtidos pelo algoritmo Pollard-rho e suas variações juntamente com a descrição dos parâmetros utilizados.

5.1 Descrição dos experimentos

1. Implementar os seguintes algoritmos para solucionar o problema do logaritmo discreto para curvas elípticas:
   - Pollard-rho com único processador.
   - Pollard-rho com paralelização.
   - Pollard-rho com múltiplos processadores.
2. Escolher curvas e seus parâmetros cuja criptografia possa ser "quebrada" em um tempo adequado para que as comparações entre os diversos algoritmos possam ser realizadas.
3. Escolher pontos P e Q pertencentes à curva e elaborar o problema do logaritmo discreto.
4. Solucionar o problema utilizando os algoritmos implementados.
5. Verificar o tempo de execução necessário para cada algoritmo resolver o problema.
6. Comparar o tempo de execução dos algoritmos entre si.
7. Comparar o desempenho do Pollard-rho multi processado para diferentes quantidades de processadores.

5.2 Escolha das curvas

As curvas escolhidas para a execução dos testes foram geradas com o auxílio da biblioteca SageMath. Quanto aos parâmetros, foi escolhido um número primo p para representar o corpo finito no qual a curva está definida. Os parâmetros a e b foram selecionados seguindo a regra estabelecida na Eq. (13). Após isso, foi feito o cálculo da ordem da curva, se a ordem é prima, a curva é utilizada, caso contrário, será descartada e outra curva será gerada seguindo os mesmos passos citados, até que seja encontrada uma curva com a ordem prima. Dessa forma, pode-se evitar ataques baseados na fatoração

da ordem da curva, tais como o ataque utilizando o algoritmo de Pohlig-Hellman (Seção 2.1.2).

Gerou-se diferentes curvas variando o valor de p de 32 a 160 bits para, posteriormente, realizar a comparação de tempo entre os algoritmos. Foram também selecionados dois pontos P e Q pertencentes à curva. Por fim, todas essas informações foram armazenadas em um arquivo de texto para ser processado pelos algoritmos.

Os parâmetros das curvas que foram "quebradas" utilizando os algoritmos implementados nesse trabalho se encontram no Anexo C juntamente com os pontos P e Q utilizados no problema do logarítmo discreto.

5.3 Implementação e execução dos algoritmos

5.3.1 Pollard-rho serial

Para realizar a implementação do Pollard-rho com único processador (serial) seguiu-se a descrição do pseudoalgoritmo no Anexo A juntamente com a teoria explanada na Seção 3.2. O ponto inicial foi escolhido de forma arbitrária para cada vez que o algoritmo fosse executado e foi utilizado apenas um processador, pois esta abordagem não possui suporte à paralelização.

A maior vantagem desse algoritmo é que ele necessita de uma quantidade ínfima de memória, pois utiliza o ciclo de Floyd, descrito na Seção 3.2.1. Dessa forma, este algoritmo pode ficar executando por tempo indeterminado mesmo em uma máquina com pouca memória.

5.3.2 Pollard-rho com paralelização

Buscando aproveitar a baixa quantidade de memória utilizada pelo Pollard-rho serial, e diminuir o tempo necessário para resolver o problema do logaritmo discreto, foi implementada uma versão modificada do algoritmo serial para que o código pudesse executar de forma paralela, afim de poder utilizar todos os processadores disponíveis da máquina. Por isso, este algoritmo difere do algoritmo serial apenas do ponto de vista estatístico.

Basicamente são utilizados M núcleos (virtuais) em que cada um executa um processo rodando o algoritmo Pollard-rho de forma independente, com diferentes pontos iniciais X_0 aleatórios, enquanto o algoritmo serial seleciona apenas um ponto inicial. Por mais que esta abordagem não seja a mais eficiente em termos de tempo execução, essa medida obteve melhorias, as quais serão apresentados na Seção 5.4.

5.3.3 Pollard-rho com múltiplos processadores

O algoritmo mais eficiente é o Pollard-rho com múltiplos processadores (multi-processado), pois este é diferente em relação aos outros algoritmos quanto à técnica e o tempo esperado de processamento, de acordo com a Seção 3.3. A utilização de um servidor central para o armazenamento de pontos é o ponto positivo do algoritmo, pois o processo que calcula os pontos ganha em velocidade por não ter a responsabilidade de checar a colisão entre os pontos e também por evitar a necessidade de um recurso compartilhado entre os processos, já que o servidor central é o único responsável por armazenar e checar a colisão entre os pontos.

O ponto negativo dessa abordagem é que o servidor não pode armazenar infinitos pontos, pois existe o limite da memória RAM. Para isso utilizou-se a propriedade distintiva (Seção 3.3.1), que vai limitar a quantidade de pontos que o servidor irá armazenar. Porém, caso essa propriedade seja muito restritiva, a possibilidade de encontrar uma colisão diminui, pois o servidor central terá um espaço amostral de pontos menor para encontrar uma colisão e, consequentemente, o tempo de execução do algoritmo irá aumentar. Por outro lado, caso a propriedade for muito abrangente, a memória pode esgotar rapidamente. Dessa forma, é necessário encontrar um equilíbrio para essa propriedade distintiva, pois quanto mais abrangente, maior a quantidade de memória utilizada para armazenar os pontos, quanto mais restritiva, menor a quantidade de pontos armazenados e maior o tempo para encontrar uma colisão entre estes pontos.

Durante a execução desse algoritmo, foram feitos diversos testes sobre a utilização da propriedade distintiva mais adequada até chegar a uma que fosse considerada boa. A propriedade escolhida foi o *Hamming weight* da coordenada x do ponto ser menor que 24. Além da dificuldade inerente a essa propriedade, ficou evidente também o problema de memória RAM, pois, apesar da utilização de uma propriedade distintiva para armazenar os pontos (o que diminui o número de pontos a serem armazenados), 16GB de memória se tornavam insuficientes após aproximadamente 25 horas de execução do algoritmo, resultando em falta de memória a partir de uma curva de 66 bits.

5.4 Tempo de execução

Com os experimentos realizados durante este trabalho, foi possível realizar uma comparação empírica entre três variantes do algoritmo Pollard-rho para ataques à criptografia de curvas elípticas. Assim, foi possível verificar os pontos fortes e as fragilidades que cada uma das abordagens oferece, bem como verificar na prática a dificuldade computacional para resolver o problema do logaritmo discreto.

Os resultados apresentados na Tab. (5) demonstram que utilizar múltiplos processos para a solução do problema é a abordagem que oferece um desempenho melhor caso

Tabela 5: Resultado dos algoritmos Pollard-rho

bits	**Serial**	**Paralelizado**	**Multiprocessado**
32	0h00m01s	0h00m01s	0h00m01s
36	0h00m08s	0h00m04s	0h00m01s
40	0h00m46s	0h00m19s	0h00m03s
44	0h02m58s	0h01m26s	0h00m14s
48	0h17m11s	0h14m47s	0h01m04s
52	0h20m26s	0h16m49s	0h04m54s
56	1h27m28s	2h46m18s	0h29m12s
60	15h11m05s	8h56m45s	0h41m38s
64	2d 14h37m58s	1d 15h52m12s	6h44m23s
65	3d 20h45m48s	-	10h33m34s
66	6d 04h30m10s	-	-

seja levado em consideração apenas o tempo de execução do algoritmo. Já em contrapartida, essa abordagem traz uma preocupante restrição física para sua implementação, que é a quantidade de memória necessária para a realização dos experimentos.

Para amenizar esse problema, foi utilizado uma propriedade distintiva que seleciona os pontos que devem ser armazenados na memória, descartando todos os outros pontos que não atendiam tal propriedade. A escolha dessa propriedade distintiva oferece um *trade-off* inevitável, pois, caso seja selecionada uma propriedade muito restritiva, a quantidade de pontos armazenados será significativamente menor, aumentando o "tempo de vida" da memória e permitindo que o algoritmo seja executado por mais tempo. Porém, ao restringir muito a quantidade de pontos armazenados, o espaço amostral que o algoritmo dispõe para encontrar uma colisão é menor, e portanto, será necessário mais processamento para encontrar uma colisão e, consequentemente, o tempo necessário para resolver o problema do logaritmo discreto será maior. Diante desse *trade-off*, pode-se verificar a importância da escolha dessa propriedade distintiva, algo que necessita e merece mais pesquisas por parte da comunidade acadêmica.

Já os algoritmos Pollard-rho serial e paralelo não possuem o problema da memória que a versão com múltiplos processadores apresenta. A vantagem desses algoritmos é utilizar sempre a mesma quantidade de memória para os cálculos, ou seja, o consumo de memória será sempre o mesmo durante toda a execução do algoritmo. Porém, é possível perceber pela Tab. (5) que esses algoritmos são mais lentos para resolver o problema do logaritmo discreto.

A melhoria de desempenho do algoritmo paralelo em relação ao serial é justificada por uma questão estatística, pois o algoritmo serial escolhe um ponto inicial de forma aleatória, e dependendo dessa escolha, o algoritmo pode demorar mais tempo para encontrar uma colisão. Já na versão paralelizada, o mesmo algoritmo serial é executado em paralelo por vários processos, onde cada um deles é inicializado com um ponto inicial.

Dessa forma, um desses processos vai partir de um ponto inicial mais propício a encontrar uma colisão do que os outros processos e, consequentemente, este processo irá encontrar uma colisão em um tempo menor do que os outros.

Os resultados da Tab. (5) podem ser colocados em um gráfico *Mono-Log*, em que a coordenada x do gráfico representa o tamanho da curva em *bits* e a coordenada y representa o tempo, em segundos, necessário para resolver o problema do logaritmo discreto. O resultado é o gráfico da Fig. (7).

Pelo gráfico da Fig. (7) é possível perceber que os resultados experimentais se aproximam de uma reta. Isto demonstra que pode-se obter uma função que irá calcular o tempo aproximado para resolver o problema do logaritmo discreto partindo do tamanho da curva em bits, e que esta função é exponencial do tipo $y = ke^{cx}$, em que e é a base dos logaritmos neperianos.

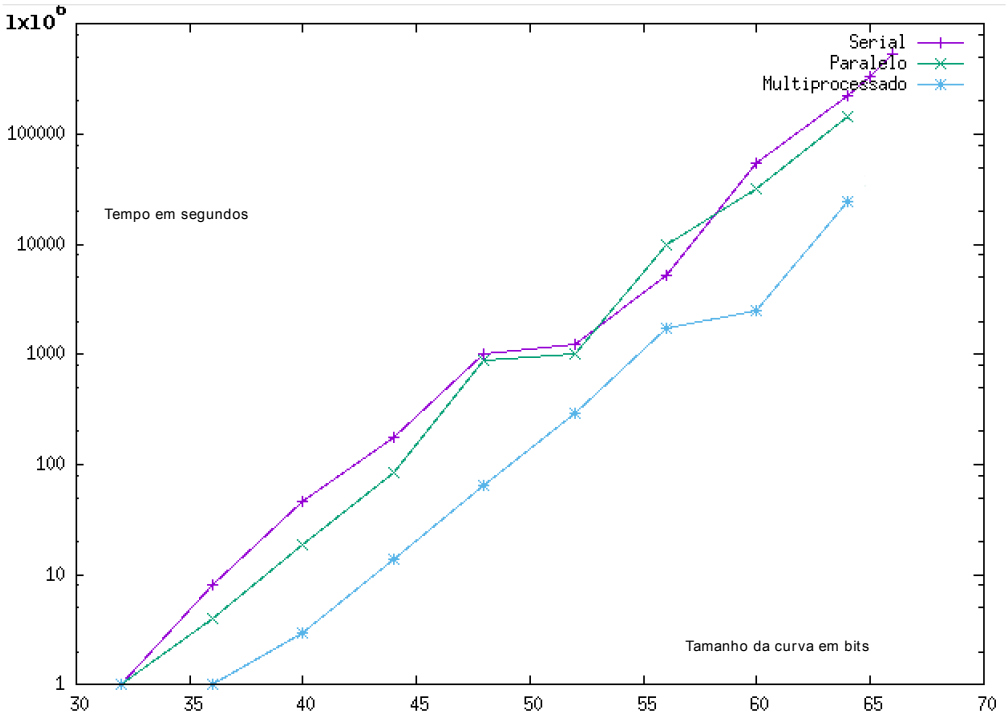

Figura 7: Tamanho em *bits* X Tempo em segundos (escala logarítmica)

Para obter a equação que relacione o tempo necessário em segundos para "quebrar" uma curva de n bits, foi aplicada a função $ln(x)$ em ambos os lados da equação $y = ke^{cx}$, obtendo-se

$$ln(y) = ln(k) + cx \cdot ln(e) = ln(k) + cx \tag{1}$$

Em seguida, foram escolhidos dois pontos A e B de uma das curvas do gráfico 7, e substituídos os valores de x e y na Eq. (1) respectivamente pelas coordenadas de A e B,

obtendo-se o sistema

$$\begin{cases} ln(Y_A) = ln(a) + b(X_A) \\ ln(Y_B) = ln(a) + b(X_B) \end{cases}$$

Com esse sistema, ao fazer $ln(Y_B) - ln(Y_A)$, obtém-se o valor de b, e em seguida o valor de $ln(a)$. Por último, aplica-se a função exponencial de ambos os lados da equação $ln(y) = ln(a) + bx$ resultante, com os valores de $ln(a)$ e de b já substituídos. O resultado é uma equação do tipo

$$y = e^{ln(a)+bx}$$

em que $e \approx 2.718281828$.

Dessa forma, as equações que relacionam o tamanho da curva em *bits* ao tempo necessário em segundos para "quebrá-la" foram as seguintes

- *Pollard-rho serial*:

$$y = e^{-10.33+0.354x}$$

- *Pollard-rho paralelizado*:

$$y = e^{-11.94+0.372x}$$

- *Pollard-rho multiprocessado*:

$$y = e^{-13.9+0.375x}$$

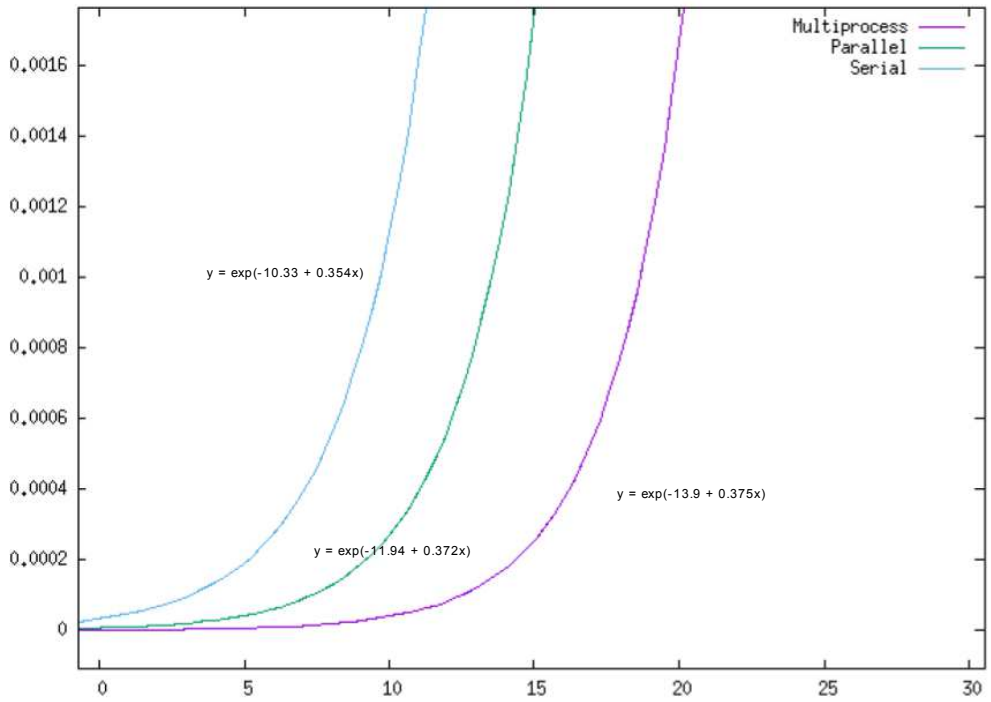

Figura 8: Equações dos algoritmos de Pollard-rho

Com essas equações, é possível obter o tempo aproximado em segundos para "quebrar" uma curva de n *bits* utilizando os algoritmos implementados. Por exemplo, para saber quanto tempo o algoritmo Pollard-rho serial levaria para "quebrar" uma curva de 80 *bits*, basta substituir na fórmula obtida para essa algoritmo

$$y = e^{-10.33+0.354*80}$$

o valor obtido é $y = 65006641.53$ segundos, ou seja, fazendo a conversão para dias, levariam 752.39 dias (cerca de 2 anos) para quebrar uma curva de 80 *bits* com o algoritmo Pollard-rho serial e com as configurações de *hardware* utilizadas.

O gráfico da Fig. (8) permite visualizar melhor o comportamento dos algoritmos. Pode-se perceber que a curva do algoritmo serial é a que cresce mais rapidamente, ou seja, este algoritmo necessitará de mais tempo para "quebrar" a curva do que os demais.

Outro experimento realizado foi medir o desempenho do algoritmo Pollard-rho multiprocessado em função da quantidade de processadores (*cores*). Para isso, foi selecionada uma curva de 40 *bits*. Os testes foram realizados com a quantidade de um até sete processadores.

A mesma curva foi executada 100 vezes para cada caso de teste, variando do número de *cores* e com o ponto inicial aleatório para cada instância do algoritmo. Calculando-se a média aritmética, a variância e o desvio-padrão para os dados coletados obtém-se os valores da Tab. (6).

Tabela 6: Valores do algoritmo Pollard-rho multiprocessado para quantidades diferentes de *cores*

Cores	**Média (s)**	**Desvio-padrão (s)**
1	80.50	36.44
2	42.47	20.71
3	24.92	13.52
4	19.88	9.79
5	18.77	10.53
6	20.38	10.13
7	20.72	9.89

Observando os dados da Tab. (6), pode-se observar de maneira geral que existe uma melhoria de desempenho do tempo de execução em função da quantidade de processadores utilizados. É possível perceber também que, com uma quantidade maior de processadores, o desvio-padrão tende a diminuir, ou seja, os dados dos testes não tiveram variações abruptas em relação à média. Os testes utilizando apenas um processador (*core*) foi o teste que apresentou maior variação em torno da média, por isso, teve o maior desvio padrão. Inclusive este caso é mais lento que o algoritmo serial, pois nem todos os pontos são considerados, apenas os pontos considerados distintos.

Nota-se que a partir de seis processadores o tempo de execução se difere do esperado, como pode ser visto pelo gráfico da Fig. (9) . Isso se deve ao fato de que a comunicação entre processos (servidor e cliente) é feita via *pipe* e este canal pode estar sendo sobrecarregado do lado do servidor por conta do processamento dos pontos distintos provenientes dos vários clientes. E isso acaba se tornando o gargalo da implementação.

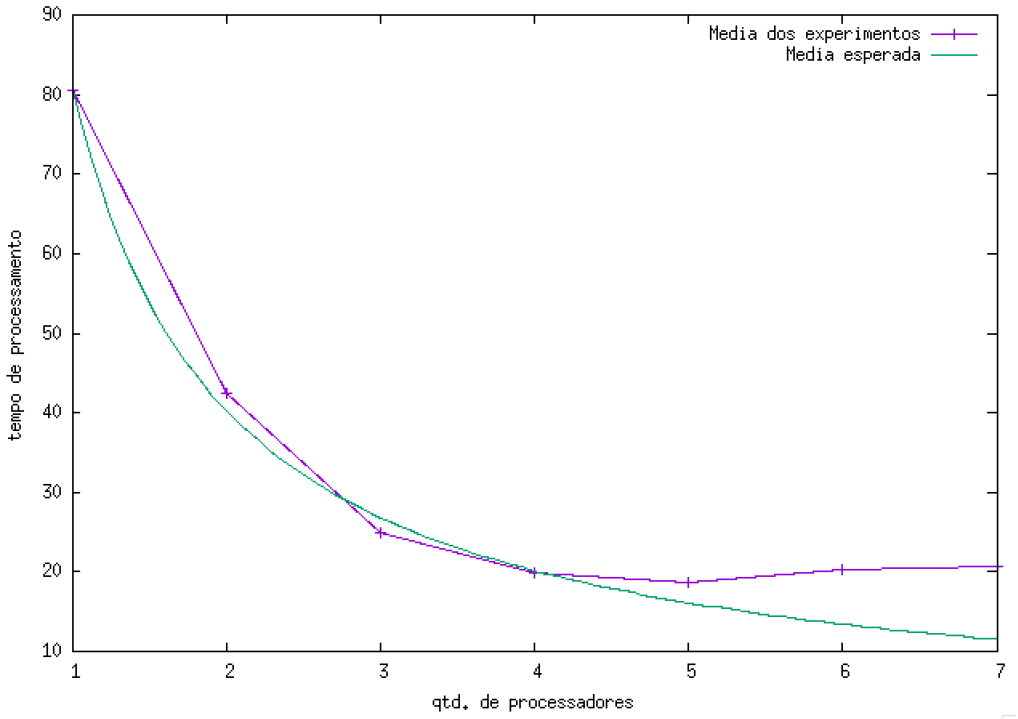

Figura 9: Tempo de execução(s) em função da quantidade de processadores

É importante saber que na geração dos resultados deste trabalho foi possível utilizar apenas uma máquina para a execução dos algoritmos, o que garante que as características de hardware utilizadas foram as mesmas, porém essa medida acaba comprometendo bastante tempo do cronograma. Até porque foram executados diversos testes que dependendo levam horas e até dias. Os resultados da Tab. (5) mostram valores referentes à média dos tempos coletados para os valores de *bits*.

6 Considerações finais

O presente trabalho apresentou os principais conceitos matemáticos da criptografia de curvas elípticas mais pertinentes à solução do problema do logaritmo discreto relacionado à curvas elípticas (ECDLP). Além disto, foi apresentado o algoritmo Pollard-rho, atualmente considerado o mais eficiente no ataque à criptografia de curvas elípticas, e os conceitos nos quais este algoritmo se baseia. Foram realizadas implementações de três versões deste algoritmo (as versões serial, paralelizada e multiprocessada), as quais foram utilizadas em um número de testes incluindo alguns em que foi variado o número de processadores (núcleos) empregados. Finalmente os tempos de execução para a "quebra" da criptografia de curva elíptica na execução destas diferentes versões foram anotados e comparados.

Outro resultado observado foi uma diminuição no tempo de processamento com o aumento do número de núcleos empregados. Este tempo se apresenta, de forma geral, inversamente proporcional ao número de núcleos empregados, o que era esperado segundo a análise teórica. A partir de seis processadores, porém, os dados experimentais diferem dos valores esperados. Isto se deve a um gargalo de comunicação (*pipe*) entre os núcleos que calculam os pontos distintos e o núcleo que verifica se uma coincidência aconteceu entre eles.

Para realização deste trabalho foi preciso ter um espírito investigativo para pesquisar as melhores soluções, tomar as melhores decisões de projeto e escolher as ferramentas mais adequadas, objetivando desenvolver um trabalho com a melhor qualidade possível, enquanto otimizando o esforço dispensado. Neste contexto, os conhecimentos de matemática, probabilidade e estatística, criptografia e, especialmente, de engenharia de software, adquiridos ao longo do Curso de Engenharia de Software na FGA foram de suma importância para que pudéssemos implementar os algoritmos Pollard-rho da forma mais eficiente e com a melhor performance possível. Dentre os *trade-offs* e decisões de projeto ao longo do desenvolvimento deste TCC podemos destacar:

1. A escolha do algoritmo Pollard-rho multiprocessado para a utilização eficiente do poderio computacional hoje disponível a pesquisadores e desenvolvedores;

2. A escolha da linguagem Python para poupar esforço de desenvolvimento de software em áreas que não eram o objetivo maior do projeto;

3. A utilização de múltiplos processos em Python, ao invés de múltiplas threads, visto que em Python múltiplas threads são executadas no mesmo processador (núcleo) devido a uma limitação do sistema de *run-time* da linguagem;

4. A escolha da propriedade distintiva dos pontos distintos do algoritmo Pollard-rho, objetivando o melhor *trade-off* possível entre o uso de memória RAM e o tempo de execução do algoritmo.

Finalmente, gostaríamos de reiterar que este projeto nos suscitou a oportunidade de aplicar vários dos conhecimentos teóricos adquiridos durante o curso à solução de um problema real e importante no universo da segurança da informação, área extremamente em demanda no mundo profissional do engenheiro de software de hoje, além de solidificar o aprendizado estes conhecimentos.

6.1 Trabalhos futuros

Como sugestões de continuidade do projeto, existem algumas etapas ou passos importantes para melhorar ainda mais o desempenho dos algoritmos. São elas:

1. *Realizar experimentos com a linguagem C/C++*: apesar da facilidade, eficiência e ter um foco matemático, Python é uma linguagem de alto nível interpretada, enquanto a linguagem C/C++ é de médio nível, pois combina características de linguagens de alto e baixo níveis, por isso vale a pena realizar testes para essa linguagem e comparar observar seus resultados.

2. *Aprofundar a pesquisa sobre a propriedade distintiva dos pontos*: por ser uma propriedade determinante para a execução do algoritmo Pollard-rho com múltiplos processadores, por si só é uma etapa que merece um estudo mais aprofundado.

3. *Implementar o algoritmo Pollard-rho com automorfismo*: apesar de ter sido apresentada a teoria necessária para aplicar o Pollard-rho com automorfismo, este algoritmo não foi contemplado na parte de resultados do trabalho, pois exige um alto nível de complexidade de implementação para que pudesse ser concluído neste trabalho. Pois é necessário encontrar uma função ψ que seja um automorfismo para cada uma das curvas geradas e isso requer um grande esforço.

4. *Implementar o algoritmo Pollard-rho multiprocessado num supercomputador (múltiplos nós, nós com múltiplos núcleos e com GPUs)*: isso traria uma dimensão maior e mais real ao aspecto da possibilidade de utilização efetiva do algoritmo Pollard-rho na área de criptoanálise.

Referências

ALVARADO, A. An exposition of schoof's algorithm. Arizona State University, 2005. Citado 2 vezes nas páginas 37 e 75.

BAI, S.; BRENT, R. P. On the efficiency of pollard's rho method for discrete logarithm. Department of Computer Science, Australian Computer Society, 2008. Citado na página 46.

BAILEY L. BATINA, D. J. B. D. V. Breaking ecc2k-130. Cryptology ePrint Archive, 2009. Citado na página 48.

CAVALCANTE, A. L. B. Teoria dos números e criptografia. Departamento de Sistemas de Informação da UPIS, 2015. Citado na página 32.

COUTINHO, S. C. *Números Inteiros e Criptografia RSA*. 2. ed. [S.l.]: IMPA, 2014. Citado 2 vezes nas páginas 28 e 30.

DOMINGUES, H. H.; IEZZI, G. *Álgebra Moderna*. 4. ed. [S.l.]: Atual Editora, 2003. Citado 5 vezes nas páginas 25, 26, 27, 28 e 29.

GIL, A. C. Como elaborar projetos de pesquisa - edição 4. 2008. Citado 2 vezes nas páginas 51 e 52.

GILBERT, W. J.; NICHOLSON, W. K. *Modern Algebra with Applications*. 2. ed. [S.l.]: Wiley-Interscience, 2004. Citado 3 vezes nas páginas 27, 28 e 30.

HALIM, S. *Competitive Programming*. 3. ed. [S.l.]: paperback, 2013. Citado na página 25.

HANKERSON, D.; MENEZES, A.; VANSTONE, S. *Guide to Elliptic Curve Cryptography*. [S.l.]: Springer, 2004. Citado 9 vezes nas páginas 31, 33, 35, 37, 43, 44, 46, 48 e 49.

HEFEZ, A.; VILLELA, M. L. T. *Códigos Corretores de Erros*. 2. ed. [S.l.]: IMPA, 2008. Citado na página 27.

HOFFSTEIN, J. *An Introduction to Mathematical Cryptograpfy*. [S.l.]: Springer, 2008. Citado na página 39.

KNUDSEN, L. R. T. Twofish. 1998. Citado na página 39.

KNUTH, D. E. *The Art of Computer Programming 3rd Edition*. [S.l.: s.n.], 1997. Citado na página 46.

LAPORTA, M.; PIZZIRANI, A. A new iterating function in the pollard rho method for discrete logarithms. 2014. Disponível em: <http://www.m-hikari.com/ams/ams-2014/ams-133-136-2014/49695.html>. Citado na página 45.

LEE, T. C. An implementation of elliptic curve cryptosystem. Department of Computer Science and Information Systems Saginaw Valley State University, 2011. Citado na página 21.

LEWINTER, M.; MAYER, J. *Elementary Number Theory with Programming*. [S.l.]: Wiley, 2015. Citado 2 vezes nas páginas 23 e 24.

MCGEE, J. J. René schoof's algorithm for determining the order of the group of points on an elliptic curve over a finite field. Virginia Polytechnic Institute and State University, 2006. Citado 2 vezes nas páginas 75 e 76.

OORSCHOT, P. V.; WIENER, M. On diffie-hellman key agreement with short exponents. 1996. Citado na página 47.

PELZL, J. On the security of elliptic cure cryptosystem against attacks with special-purpose hardware. Departamento de Sistemas de Informação da UPIS, 2006. Citado na página 40.

POLLARD, J. Monte carlo methods of index computation (mod p). 1978. Citado na página 40.

PORTNOI, M. Criptografia com curvas elípticas. Universidade Salvador - UNIFACS, 2005. Citado na página 31.

SANGALLI, L. A. Criptossistemas baseados em curvas elípticas e seus desafios. Campinas, SP, Brasil, 2011. Citado na página 21.

SANTOS, J. P. d. O. *Introdução à Teoria dos Números*. [S.l.]: IMPA, 2014. Citado 2 vezes nas páginas 23 e 24.

SEET, M. Z. Elliptic curve cryptography - improving the pollard-rho algorithm. 2007. Citado 4 vezes nas páginas 33, 39, 43 e 76.

SHOKRANIAN, S. *Algebra 1*. 1. ed. [S.l.]: Ciência Moderna, 2010. Citado na página 31.

SHOUP, V. *A Computational Introduction to Number Theory and Algebra*. [S.l.]: Cambrigde University Press, 2005. Citado 2 vezes nas páginas 28 e 29.

SILVERMAN, J. H. *The Arithmetic of Elliptic Curves*. 2. ed. [S.l.]: Springer, 2009. Citado 2 vezes nas páginas 73 e 75.

STALLINGS, W. *Cryptography and Network Security Principles and Pratice 5th Edition*. [S.l.]: Prentice Hall, 2011. ISBN 10: 0-13-609704-9, 13: 978-0-13-609704-4. Citado 7 vezes nas páginas 21, 24, 33, 34, 35, 36 e 38.

TAFNER, E. P.; SILVA, R. Apostila de metodologia científica. 2007. Citado 2 vezes nas páginas 51 e 52.

TRAVASSOS, G. H. *Introdução à engenharia de software experimental*. UFRJ, 2002. (RT-ES-590/02). Disponível em: <http://www.ufpa.br/cdesouza/teaching/topes/4-ES-Experimental.pdf>. Citado 2 vezes nas páginas 51 e 52.

WANG, P.; ZHANG, F. An efficient collision detection method fo computing discrete logarithms with pollard's rho. School of Information Science and Technology, Sun Yat-sen University, Guangzhou 510006, China, 2011. Citado na página 46.

WOHLIN, C. *Experimentation in Software Engineering: An Introduction*. [S.l.]: Norwell, MA, USA: Kluwer Academic Publishers, 2000. ISBN 0-7923-8682-5. Citado na página 51.

Apêndices

APÊNDICE A – Otimizando a multiplicação de um ponto

A multiplicação de um ponto da curva elíptica é de extrema importância para a criptografia de curvas elípticas, pois, como foi visto, sua segurança depende da dificuldade em resolver o Problema do Logaritmo Discreto para Curvas Elípticas. Como a multiplicação de um ponto é definida em termos de sucessivas somas desse ponto com ele mesmo, então a multiplicação de um ponto se torna onerosa ao multiplicá-lo por um número muito grande. Para exemplificar, considere uma curva E/K e o ponto $P \in E(K)$, para calcular a multiplicação desse ponto por um número n seria necessário fazer uma soma recursiva para calcular os pontos

$$P, \quad [2]P = P + P, \quad [3]P = [2]P + P, \quad \ldots, \quad [n]P = [n-1]P + P$$

dessa forma, seria necessário realizar $n-1$ somas para encontrar o valor de $[n]P$. (SILVERMAN, 2009)

Um algoritmo mais eficiente para multiplicar um ponto $P \in E(K)$ por um número n é descrito abaixo (retirado de (SILVERMAN, 2009))

1. Escreva n como uma expansão binária

   $$n = \epsilon_0 + \epsilon_1 \cdot 2 + \epsilon_2 \cdot 2^2 + \cdots + \epsilon_t \cdot 2^t$$

   com $\epsilon_0, \ldots, \epsilon_t \in \{0, 1\}$ e $\epsilon_t = 1$.

2. Atribua $Q = P$ e $R = \begin{cases} O, & \text{se} \quad \epsilon_0 = 0 \\ P, & \text{se} \quad \epsilon_0 = 1 \end{cases}$

3. Repita para $i = 1, 2, \ldots, t$

   Atribua $Q = [2]Q$

   Se $\epsilon_i = 1$ então $R = R + Q$

4. Retornar R

APÊNDICE B – Algoritmo de Schoof

O algoritmo de Schoof é um algoritmo para calcular a ordem de uma curva elíptica em um tempo polinômial, que é dado por $O((\log\ n)^8)$ (SILVERMAN, 2009).

Considere uma curva elíptica definida sobre um corpo finito E/F_q e expressa pela equação $y^2 = x^3 + Ax + B$. Considere ainda que $q = p^n$ e que $p \neq 2, 3$. Pelo teorema de Hasse é dado que

$$\#E(F_q) = q + 1 - t, \qquad \text{com } |\mathrm{a}| \leq 2\sqrt{q}$$

e considere um conjunto de números primos $S = \{2, 3, 5, 7, \ldots, L\}$ tal que

$$\prod_{l \in S}(l \geq 4\sqrt{q})$$

e a característica p do corpo finito não pertença a S. O algoritmo consiste em encontrar t mod l para cada primo $l \in S$, e enfim descobrir o valor de t utilizando o teorema do resto chinês.(ALVARADO, 2005)

B.0.0.1 Caso com $l = 2$

Para o caso em que $l = 2$, suponha que a equação $x^3 + Ax + B$ tenha uma raiz $\alpha \in \mathbb{F}_q$, então existe um ponto $(\alpha, 0) \in E(\mathbb{F}_q)$. A soma desse ponto com ele mesmo é o ponto no infinito $\mathcal{O}$, logo esse ponto possui ordem dois e dizemos que ele pertence ao grupo de pontos da curva elíptica com ordem dois, que é representado por $(\alpha, 0) \in E[2]$. O teorema de Lagrange diz que a ordem do subgrupo deve dividir a ordem do grupo, logo, se existe um subgrupo de ordem dois então a ordem do grupo será par, ou seja, $\#E(\mathbb{F}_q) = q + 1 - t \equiv 0 \pmod 2$, e por q ser ímpar, conclui-se que $t \equiv 0 \pmod 2$. (ALVARADO, 2005)

Dessa forma, se for provado que a equação $x^3 + Ax + B$ possui uma raíz em $\mathbb{F}_q$, então $t \equiv 0 \pmod 2$, caso contrário, se a raíz não existe, então $t \equiv 1 \pmod 2$.(MCGEE, 2006)

Uma forma verificar se a equação $x^3 + Ax + B$ possui uma raíz em $\mathbb{F}_q$ é fazendo o cálculo do seu máximo divisor comum com o polinômio $x^q - x$. Não será demonstrado nesse trabalho que o polinômio $x^q - x$ possui q raízes distintas em $\mathbb{F}_q$, logo calculando-se $\mathrm{mdc}(x^3 + Ax + B, x^q - x)$ será possível saber se a equação possui uma raiz em $\mathbb{F}_q$. Caso o mdc entre os polinômios seja igual a um, então a equação $x^3 + Ax + B$ não possui raiz em $\mathbb{F}_q$ e consequentemente $t \equiv 1 (\mathrm{mod}\ 2)$.(MCGEE, 2006)

B.0.0.2 Caso com $l > 2$

Para calcular t mod l para $l > 2$, é necessário fazer uso de uma matématica mais rebuscada. Para o propósito desse trabalho, serão dadas algumas definições e o algoritmo para realizar o calculo.

Dada uma curva elíptica E, o seu **j-invariante** é dado por

$$j = j(E) = 1728\frac{4A^3}{4A^3 + 27B^2} \tag{1}$$

onde $A^3 + 27B^2 \neq 0$. (SEET, 2007)

Outro conceito importante para o algoritmo de Schoof é o de polinômios de divisão. Tais polinômios são definidos sobre as coordenadas de um ponto da curva elíptica e seu resultado é zero para pontos de determinada ordem. Explicando melhor, define-se $E[n]$ como o conjunto dos pontos de E que possuem ordem n, a saber

$$E[n] = \{P \in E(\mathbb{F}_q) \quad | \quad nP = \mathcal{O}\}$$

Assim, o polinômio de divisão ψ_n de uma curva E possui a propriedade $\psi_n(x, y) = 0$ se e somente se o ponto com coordenadas $(x, y) \in E(\mathbb{F}_q$ possui ordem n, ou seja, $(x, y) \in E[n]$.(MCGEE, 2006)

Esses polinômios $\psi_m(x, y)$ podem ser obtidos recursivamente por

$$\begin{aligned}
&\psi_0 = 0\\
&\psi_1 = 1\\
&\psi_2 = 2y\\
&\psi_3 = 3x^4 + 6Ax^2 + 12Bx - A^2\\
&\psi_4 = 4y(x^6 + 5Ax^4 + 20Bx^3 - 5A^2x^2 - 4ABx - 8B^2 - A^3)\\
&\psi_{2m+1} = \psi_{m+2}\psi_m^3 - \psi_{m-1}\psi_{m+1}^3\\
&\psi_{2m} = \frac{\psi_m(\psi_{m+2}\psi_{m-1}^2 - \psi_{m-2}\psi_{m+1}^2)}{2y}
\end{aligned}$$

com $m > 2$. (SEET, 2007)

Agora, o algoritmo(SEET, 2007) para calcular t mod l para $l > 2$

1. Calcular $p_l \equiv p \bmod l$ com $|p_l| < l/2$.
2. Calcular a coordenada x' de

$$(x', y') = (x^{p^2}, y^{p^2}) + p_l(x, y) \bmod \psi_l$$

3. Para cada $h = 1, 2, \cdots, \frac{l-1}{2}$, calcular a coordenada x_h de $(x_h, y_h) = j(x, y)$

(a) Se $x' - x_h^p \equiv 0(\text{mod } \psi_l)$, ir para (b). Senão, tentar o próximo valor de h.

(b) Calcular y' e y_h. Caso

$$\frac{y' - y_h}{y} \equiv 0(\text{mod } \psi_l)$$

então $t \equiv h(\text{mod } l)$. Caso contrário, $t \equiv -h(\text{mod } l)$.

4. Caso o item (a) da etapa anterior não encontre valor de h que satisfaça a equivalência, então fazer $\omega^2 \equiv p(\text{mod } l)$. Caso ω não exista, então $t \equiv 0(\text{mod } l)$.

5. Se mdc(numerador$(x^p - x_\omega), \psi_l) = 1$, então $t \equiv 0(\text{mod } l)$. Senão, calcular

$$\text{mdc}(\text{numerador}(\frac{y^p - y_\omega}{y}), \psi_l)$$

Se mdc $\neq 1$, então $t \equiv 2\omega(\text{mod } l)$, caso contrário $a \equiv -2\omega(\text{mod } l)$.

Anexos

ANEXO A – Primeiro Anexo

Este anexo se refere à Seção 3.2.

O algoritmo recebe como parâmetro uma curva elíptica E e dois pontos P e Q pertencentes a E e calcula o valor de x tal que $x = \log_P Q$.

Código A.1: Algoritmo Pollard-rho com único processador.

```
1  pollardRho_singleProcessor(E: Curva, P: Ponto, Q: Ponto): inteiro
2      seja L, k, inteiro
3      seja an, bn, inteiro
4      seja am, bm, inteiro
5      seja Xn, Xm, Ponto
6      seja c, d, array inteiro
7      seja R, array Ponto
8
9      k ← E.order()
10     para j ← 1, enquanto j ≤ L, faca
11         c[j] ← random() % k
12         d[j] ← random() % k
13         R[j] ← c[j]*P + d[j]*Q
14
15     an ← random() % k
16     bn ← random() % k
17     Xn ← an*P + bn*Q
18     am ← an
19     bm ← bn
20     Xm ← Xn
21
22     enquanto Xn != Xm faca
23         j ← H(Xn, L)
24         Xn ← Xn + R[j]
25         an ← an + c[j]
26         bn ← bn + b[j]
27
28     se bn = bm entao
29         retorne "falha"
30
31     seja x, inteiro
32     x ← (an - am)/(bm - bn) % k
33     retorne x
34
35 H(P: Ponto, L: inteiro): inteiro
36     retorne P.x % L + 1
```

ANEXO B – Segundo Anexo

Este anexo se refere à Seção 3.3.

O algoritmo recebe como parâmetro uma curva elíptica E e dois pontos P e Q pertencentes a E e calcula o valor de x tal que $x = \log_P Q$.

Código B.1: Algoritmo Pollard-rho paralelizado.

```
1  pollardRho_parallelized(E: Curva, P: Ponto, Q: Ponto): inteiro
2      seja L, k, inteiro
3      seja an, bn, inteiro
4      seja am, bm, inteiro
5      seja Xn, Xm, Ponto
6      seja c, d, array inteiro
7      seja R, array Ponto
8
9      seja a, b, inteiro
10     seja X, ponto
11     seja Y, Point
12
13     k ← E.order()
14     para j ← 1, enquanto j ≤ L, faca
15         c[j] ← random() % k
16         d[j] ← random() % k
17         R[j] ← c[j]*P + d[j]*Q
18
19     # para cada processador M faca
20     a ← random() % k
21     b ← random() % k
22     X ← an*P + bn*Q
23
24     # repita ate que o servidor receba um ponto Y distinto pela 2a vez
25     enquanto duplicatedDistinguishedPointNotReceived() faca
26         se X = distinguishedPoint(X)
27             sendToServer(a, b, X)
28             j ← H(X)
29             X ← X + R[j]
30             a ← a + c[j] % k
31             b ← b + d[j] % k
32
33     an ← firstTripleCollided_a()
34     bn ← firstTripleCollided_b()
35     am ← secondTripleCollided_a()
36     bm ← secondTripleCollided_b()
37
```

```
38      se bn = bm entao
39          retorne "falha"
40
41      seja x, inteiro
42      x ← (an - am)/(bm - bn) % k
43      retorne x
44
45  H(P: Ponto, L: inteiro): inteiro
46      retorne P.x % L + 1
```

ANEXO C – Terceiro Anexo

A seguir serão descritos os parâmetros das curvas utilizadas e os pontos selecionados na elaboração do problema do logaritmo discreto (ECDLP).

Tabela 7: Parâmetros das curvas

bits	**A**	**B**	**p**	**ordem**
32	48489	97594	2882746957	2882734229
36	50533	32996	40682286653	40682161199
40	42144	98649	950263177883	950264094121
44	40198	11702	16020544557377	16020543189151
48	52920	71257	228362934047953	228362914116653
52	99400	82990	3504467965402487	3504467954199811
56	50404	48342	51490278520613677	51490278431089031
60	82106	75478	706856769607124263	706856770750892953
64	98626	27290	10136088024214031251	10136088023118239857
65	4838	85770	18505261939516721599	18505261939128715183
66	78319	6465	53604234314297933089	53604234316981472713

Tabela 8: Pontos utilizados no ECLDP

bits	**P (x, y)**	**Q (x, y)**
32	(1366664103, 1798920923)	(2702135874, 2667016446)
36	(25075567916, 24630560942)	(10330877254, 15721202004)
40	(573167557027, 832383072150)	(630765322529, 829885145927)
44	(7321690921039, 5900903534434)	(15339793358558, 5065066467746)
48	(90391684716440, 216804106155420)	(189599746109552, 60036328328093)
52	(730734419482594, 2275382146632833)	(601104431610521, 3428181333227735)
56	(27290054680121669, 33873600851117647)	(17717646814745382, 41475147684384198)
60	(279391159031135790, 412346973450156951)	(477223543628388853, 375727204622926174)
64	(2310651219632187277, 8645536242470898633)	(4494019437538409888, 5769703912640477331)
65	(7641240624886147510, 16388105095474639144)	(15127239423762891439, 12518025431655074976)
66	(16503064980171743392, 44363827824736096237)	(38223962489371357849, 9876943738967192018)