UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

JOÃO LAURO GARIBALDI JUNIOR

Aplicabilidade de Criptografia Homomórfica

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Raul Fernando Weber

Porto Alegre
2018

Este trabalho é dedicado à minha mãe,
Carmem de Lourdes Ramos Braga,
e ao meu pai, João Lauro Garibaldi,
por tudo o que representam para mim.

AGRADECIMENTOS

Ao professor Raul Fernando Weber pela orientação e amizade, pontos que ajudaram a iniciar e organizar o trabalho.

Ao meu amigo e colega Gabriel Mattos Langeloh pelas discussões produtivas sobre questões matemáticas, e pela disposição para ajudar na revisão.

Ao professor João Cesar Netto por aceitar participar desse trabalho como avaliador.

Ao professor Rodrigo Machado pela dedicação às suas aulas e pela atenção que sempre deu aos seus alunos.

À professora Aline Bona pela dedicação e amor às aulas de matemática do ensino médio, garantindo um ensino decente de matemática no Colégio Estadual Ruben Berta.

RESUMO

Esse trabalho visa apresentar alguns conceitos da área de criptografia com a finalidade de familiarizar o leitor com conceitos básicos, para então apresentar criptografia homomórfica e avaliar sua utilidade e desempenho. Para tal finalidade, foram descritos algoritmos de encriptação famosos, e criado um protótipo a fim de testar o desempenho de uma aplicação que utilize uma técnica de criptografia homomórfica para realizar operações com dados encriptados, se utilizando da biblioteca *GNU Multi Precision Arithmetic Library (GMP)* para as linguagens *C* e *C++*, a fim de poder replicar algoritmos de encriptação trabalhando com números que possuem uma representação em *bits* maior do que as dos tipos nativos da linguagem *C++*. Fica claro com o que é discutido ao longo do texto que essa é uma forma de garantir melhor proteção em termos de confidencialidade para os dados, porém é notável, através dos resultados, que há um alto custo no processamento de dados encriptados.

Palavras-chave: Encriptação homomórfica. criptografia. chave pública.

Homomorphic Encryption Applicability

ABSTRACT

This work aims to present some of the concepts in cryptography area for the reader to become acquainted with some basic concepts so he can be introduced to homomorphic encryption, and see it's use and performance. For this purpose, popular encryption algorithms were described, and a prototype was created to test performance of an application implementing a homomorphic encryption technique to operate on encrypted data, making use of the library *GNU Multi Precision Arithmetic Library (GMP)* for *C* and *C++* languages, so it can reproduce encryption algorithms working with numbers that possess representation in bits bigger than the *C++* language native types. It is clear, based on what is discussed throughout the text, that it is a way to assure a higher protection in terms of confidentiality to data, but it is notable, by the results, there is a high cost in processing encrypted data.

Keywords: homomorphic encryption. public key. cryptography.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AES	Advanced Encryption System
CDH	Computational Diffie-Hellman
DDH	Decisional Diffie-Hellman
DH	Diffie-Hellman
DRCA	Decisisional Composite Residuosity Assumption
GMP	GNU Multi Precision Arithmetic Library
IACR	International Association for Cryptologic Research
IF	Integer Factorization
IMM	Inverso Multiplicativo Modular
MDC	Máximo Divisor Comum
MMC	Mínimo Múltiplo Comum
VoIP	Voice over Internet Protocol

SUMÁRIO

1 INTRODUÇÃO

O processamento de dados criptografados é uma pauta antiga no meio teórico relacionado a segurança da informação e criptanálise, já existente pelo menos desde a década de 1970 (ADLEMAN; ADLEMAN; DERTOUZOS, 1978).

Muitos algoritmos de encriptação desenvolvidos há décadas atrás possuem homomorfismo parcial, desde a sua criação, porém com o poder computacional da época o processamento da informação criptografada não era viável, uma vez que o tamanho dos dados encriptados eram maiores que o dos dados em claro na maioria das situações, o que além de um custo alto em espaço na memória gerava um custo alto no processamento de grandes dados. Porém, recentemente com o desenvolvimento do primeiro sistema de encriptação completamente homomórfico (GENTRY, 2009), com a realidade da computação em nuvem, e o poder computacional da atualidade, em conjunto com a crescente preocupação com segurança da informação, a área voltou a ganhar destaque e desde então novas aplicações, *APIs* e *frameworks* foram desenvolvidos e estudados.

Na prática, a técnica proposta de utilizar o sistema de *bootstraping* (GENTRY, 2009) tornava o sistema de encriptação usado muito mais caro em termos de operações e tempo de execução, sendo uma prova de conceito não utilizável na prática por computadores modernos comuns e aplicações de baixa latência de resposta, porém outros sistemas foram propostos com o fim de criar um sistema de encriptação ou técnica que possibilitasse computação homomórfica parcial, total ou a chamada *somewhat homomorphic encryption* (GENTRY, 2009).

O objetivo da encriptação dos dados para o processamento se justifica pois no contexto original de criptografia de chave pública e simétrica, os dados são encriptados de uma ponta a outra da comunicação, não havendo garantias da proteção dos dados quando os mesmos chegam ao destinatário, como, por exemplo, a presença de *backdoors*, podendo assim comprometer a informação sensível que apenas o remetente de determinados dados e pessoas selecionadas pelo mesmo deveriam ter acesso. Além disso, em aplicações remotas, como uma aplicação *cliente-servidor* clássica, que trabalham com encriptação dos dados precisam descriptografar os dados do lado do servidor para processamento dos mesmos, permitindo que esses sistemas vazem os dados caso comprometidos, ou que isso aconteça até mesmo de forma intencional e maliciosa.

A confidencialidade é o ponto principal abrangido pela ideia de homomorfismos aplicados a computação de sistemas de encriptação, uma vez que a aplicação de criptografia

homomórfica não influencia positivamente questões de integridade e disponibilidade, os princípios chave na área de segurança da informação (NIST, 2016). De fato, a encriptação homomórfica aplicada sem nenhum protocolo especial compromete a integridade, pois é necessária a maleabilidade dos dados para a sua aplicação, que é a capacidade de alterar os dados encriptados de forma que ao se decriptar o mesmo se obtenha um resultado diferente do original porém coerente com o sistema, o que não costuma ser desejável em sistemas de encriptação tradicionais.

A motivação para esse trabalho é estudar o funcionamento das propriedades homomórficas presentes em algoritmos de chave pública famosos e populares na área de segurança e criptanálise, visando compreender as propriedades matemáticas que garantem os homomorfismos de operações, contribuindo com uma revisão de algoritmos de encriptação conhecidos e demonstrações de provas matemáticas do funcionamento de tais sistemas parcialmente homomórficos. Outro objetivo do trabalho é a implementação de um programa com funcionalidade homomórfica com o fim de validar o estudo teórico, concluindo algumas informações sobre o desempenho de aplicações que trabalham com criptografia homomórfica, a partir da criação de um protótipo de sistema de votação utilizando criptografia homomórfica, e, finalmente, análise de resultados experimentais para poder estimar complexidade e viabilidade da implementação de tais sistemas.

Este trabalho segue organizado da seguinte maneira. No capítulo 2 serão apresentadas definições básicas sobre a criptografia e seus diferentes tipos empregados na computação. O capítulo 3 apresenta a matemática envolvida em diferentes sistemas de encriptação, de forma a encriptar e decriptar os dados. Já no capítulo 4 o objetivo é verificar, dado o que foi visto no capítulo 3, como acontece a homomorfismo em operações de sistemas de encriptação, focado especificamente em chave pública. São referenciadas no capítulo 5 algumas aplicações práticas de sistemas que utilizam a criptografia homomórfica para de preservar a confidencialidade dos dados, e também é apresentado um protótipo na seção 5.3 com a finalidade de demonstrar na prática a teoria que foi apresentada ao longo do trabalho através de um exemplo de uma simples aplicação de votação, bem como possibilitar a avaliação de desempenho da mesma.

2 CRIPTOGRAFIA

No estudo de técnicas para garantir a segurança e/ou privacidade da comunicação, estudo esse denominado criptologia, a criptografia é o estudo do desenvolvimento de algoritmos de encriptação e decriptação para garantir a mesma (STALLINGS, 2011).

Nesse capítulo será visto um resumo sobre as grandes classes de criptografia nas seções 2.2 e 2.4, e na seção 2.5 será discutida uma técnica criptográfica inerente a alguns algoritmos de encriptação.

2.1 Definições Básicas

Nesta seção são apresentados conceitos abordados ao longo da monografia para que o leitor esteja familiarizado com os mesmos ao se deparar com algum deles nas seções e capítulos seguintes.

As primeiras definições que seguem são definições de alto nível sobre a área da criptografia.

Definição 1. Um *atacante* é um usuário que visa violar os princípios de segurança de um sistema criptográfico.

Definição 2. Segurança da informação é o termo que descreve o conjunto de técnicas para controlar o acesso e controle da informação digital, com o objetivo de proteger três categorias de violação (SALTZER; SCHROEDER, 1975), sendo elas:

- Vazamento de informação a pessoas não autorizadas.
- Modificação da informação por indivíduos não autorizadas.
- Negação não autorizada de acesso à informação.

Os princípios visando proteger essas categorias de violações de segurança vieram a ser conhecidos como, respectivamente, confidencialidade, integridade e disponibilidade, conhecidos na área como tríade *CIA* (*Confidentiality, integrity and availability*) (PARKER, 2010).

A seguir são apresentadas definições importantes para o entendimento das operações matemáticas realizadas nos algoritmos de encriptação e técnicas que serão apresentadas posteriormente nesse trabalho.

Definição 3. Um grupo, na álgebra, dados um conjunto G e uma operação binária $*$:

$G \times G \longrightarrow G$, é uma tupla $(G, *)$ onde são respeitadas as propriedades de:

- Associatividade: $\forall a, b, c \in G$, $(a * b) * c = a * (b * c)$;
- Elemento neutro: $\exists e \in G$, $\forall a \in G$, onde e é o elemento neutro de G, tal que $a * e = e * a = a$;
- Inverso: $\forall a \in G$, $\exists b \in G$, onde b também é denotado como a^{-1}, o inverso de a, de forma que $a * b = b * a = e$.

Definição 4. Um grupo $(G, *)$ é dito um grupo abeliano sss o grupo possui comutatividade entre os elementos, ou seja, $\forall a, b \in G$, $a * b = b * a$

Definição 5. Um grupo de inteiros módulo n, tipicamente denotado $(\mathbb{Z}/n\mathbb{Z}, +)$ ou $\mathbb{Z}_n$ é um grupo abeliano tal que:

- $\forall x \in \mathbb{Z}_n$, $0 \leq x < n$ e $x \in \mathbb{Z}$;
- $\forall a, b \in \mathbb{Z}_n$, a soma dos operandos a e b é dada por $a + b \pmod{n}$;
- O elemento neutro $e \in \mathbb{Z}_n$ é 0;
- O inverso de 0 é 0, e para qualquer outro elemento $a \in \mathbb{Z}_n$ o inverso $a^{-1} = n - a$.

Definição 6. Um grupo multiplicativo módulo n é , denotado por $\mathbb{Z}_n^*$ um grupo abeliano tal que:

- A operação binária do grupo é a multiplicação $\times$;
- $\forall a \in \mathbb{Z}_n^*$, $0 < a < n$ e $\exists b \in \mathbb{Z}_n^*$ tal que $a \times b \equiv 1 \pmod{n}$;
- A operação de multiplicação é definida como $a \times b \pmod{n}$ $\forall a, b \in \mathbb{Z}_n^*$
- O elemento neutro $e \in \mathbb{Z}_n^*$ é 1;
- $\forall a, b \in \mathbb{Z}$, $b = a^{-1}$ sss $a \times b = e \pmod{n}$.

Definição 7. A ordem de um grupo representa o número de elementos presentes no mesmo. Algumas propriedades da ordem de um grupo são:

- Seja $|G|$ a ordem de um grupo G, $\forall a \in G$, $a^{|G|} = 1$;
- Seja G_{sub} um subgrupo de G, $|G| \equiv 0 \pmod{|G_{sub}|}$.

Definição 8. A ordem de um elemento a de um grupo G, seja e o elemento identidade do grupo, é definida como o menor número $0 < m \in \mathbb{N}$ tal que $a^m = e$. A ordem de um elemento a também é a ordem do subgrupo gerado por a.

Definição 9. Um corpo $\mathbb{F}$ é qualquer conjunto que, com operações de adição e multiplicação, satisfaça as propriedades conforme especificadas na tabela 1.

Tabela 1: Axiomas de corpos

Propriedade	Adição	Multiplicação
Associatividade	$a + (b + c) = (a + b) + c$	$a \times (b \times c) = (a \times b) \times c$
Comutatividade	$a + b = b + a$	$a \times b = b \times a$
Distributividade	$a \times (b + c) = a \times b + a \times c$	$(a + b) \times c = a \times c + b \times c$
Identidade	$a + 0 = 0 + a = a$	$a \times 1 = 1 \times a = a$
Inversos	$a + a^{-1} = a^{-1} + a = 0$	$a \times a^{-1} = a^{-1} \times a = 1, \forall (a \neq 0) \in \mathbb{F}$

Definição 10. Um corpo finito, ou campo de Galois, $GF(p^n)$, é um corpo com ordem finita prima p. No caso especial em que $n = 1$, se chama o corpo finito de corpo primo. No caso de corpos primos $GF(p)$, os elementos do conjunto são números inteiros $0 \leq x < p$. Quando o corpo é um campo de Galois da forma $GF(p^n), n \geq 2$, os elementos são polinômios com coeficientes $0 \leq c < p$ e grau $0 \leq i < n$, contendo p^n elementos. Assim como em um corpo primo a aritmética é realizada em módulo p, a aritmética polinomial de um campo de Galois $GF(p^{n>1})$ é realizada usando como módulo um polinômio irredutível de grau n qualquer, onde quaisquer polinômios de mesmo grau geram o mesmo corpo a menos de isomorfismo.

Definição 11. A função $\phi(n)$, conhecida como função totiente de Euler, pode ser usada para encontrar a quantidade de números co-primos um número $n \in \mathbb{N}$. Sendo assim, o resultado da função $\phi(n)$ é igual a ordem de um grupo multiplicativo módulo n. A função é definida, levando em consideração os fatores primos p de n, $p|n$, como:

$$n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

Definição 12. A função de Carmichael aplicada a $n \in \mathbb{N}$, $\lambda(n)$, no contexto de grupos multiplicativos módulo n, resulta no menor valor $m \in \mathbb{N}$ tal que $a^m \equiv 1 \pmod{n}$ $\forall a \in \mathbb{Z}_n^*$, ou ainda, a maior ordem de um elemento dentre os elementos do grupo. A função de Carmichael é definida como:

$$\lambda(n) = \begin{cases} \frac{1}{2}\varphi(n), \ \forall n = 2^x, x \geq 3 \wedge x \in \mathbb{N} \\ \varphi(n) \text{ caso contrário} \end{cases}$$

2.2 Criptografia Simétrica

A criptografia moderna originalmente surgiu com a ideia de chave simétrica, ou seja, a mesma chave usada na encriptação é a chave que deve ser usada na decriptação. É a ideia usada desde a criptografia clássica, onde o objetivo era manter uma informação secreta e apenas as pessoas que conheciam o segredo da técnica criptográfica utilizada deveriam ser capazes de ter acesso à informação.

Normalmente as técnicas usadas não utilizam nenhum algoritmo com alta complexidade a ser resolvido para assegurar a informação, mas sim uma geração aleatória, através de algoritmos que assegurem a difusão e confusão da informação, a partir de uma chave secreta, também chamada de segredo, que é utilizada no procedimento de forma a influenciar no processo de randomização e que é necessária para o processo inverso, de decriptação. Sendo assim, um atacante qualquer que deseja decriptar uma informação encriptada através de criptografia simétrica deve ter conhecimento da chave tendo acesso a mesma ou então gerando a chave através da técnica de força bruta, o que garante uma solução altamente segura e resistente a análises dos algoritmos. Um grande problema derivado disso foi o problema de distribuição da chave secreta na comunicação, pois muitas vezes a informação secreta deveria ser acessível por pessoas específicas além daquela que realizou a encriptação. Esse foi um problema muito grande anterior ao desenvolvimento do algoritmo *Diffie-Hellman (DH)*, pois nessa época qualquer informação passada por uma rede era visível, então por mais que a informação estivesse encriptada, a chave precisaria ser visível para o destinatário. As alternativas para isso ou eram muito caras economicamente como, por exemplo, estabelecer um canal físico de rede privado entre as duas partes, ou então em termos de tempo, como envio através de um entregador físico extremamente confiável. Conforme será discutido na seção 2.4, há razões para algoritmos de encriptação simétrica ainda serem utilizados hoje em dia, e no mesmo, serão vistos algoritmos mais relevantes na área.

2.3 Comunicação em Canal Inseguro

Já inspirado em propostas e estudos anteriores sobre como lidar com comunicação privada, um estudo foi proposto no ano de 1974 (MERKLE, 1978), uma forma teórica de se obter comunicação privada em canais inseguros.

As propostas de comunicação segura da época consideravam que uma criptografia

simétrica seria usada, e para tal, haveria a necessidade de um canal de transmissão seguro para a chave que decifraria uma mensagem encriptada transmitida em canal não seguro. As principais premissas da época para um canal seguro eram as seguintes:

1. Todas as tentativas de alterar a mensagem no canal ou de injetar mensagens falsificadas por parte de um atacante deveriam ser detectadas;
2. Um atacante deveria ser incapaz de visualizar qualquer mensagem que passasse pelo canal.

Foi proposta uma modificação na segunda premissa de forma que fosse permitido a qualquer um ver o que fosse passado no canal, e dessa forma seria necessário um método para que os envolvidos na comunicação pudessem chegar a um acordo sobre a chave através do canal de maneira simples, mas que para um atacante que estivesse interceptando as mensagens, o processo de descobrir a chave que foi acertada entre os envolvidos fosse cara para um atacante (MERKLE, 1978).

O método apresentado por ele para exemplificar isso ficou conhecido como *Quebra-Cabeças de Merkle*, o qual consistia no processo descrito da seguinte forma:

- Uma das partes inicia a comunicação enviando diversos "quebra-cabeças" no canal público. Tendo um número n de "quebra-cabeças", cada um com uma complexidade m para ser resolvido, onde cada um devolve um par $(identificador, chave)$. O remetente dos "quebra-cabeças" sabe qual identificador corresponde a qual chave;
- Do lado do destinatário, o mesmo seleciona qualquer "quebra-cabeça" aleatoriamente e o resolve com complexidade m. Após a resolução, o destinatário obtém a chave e o respectivo identificador, e envia o identificador de volta para o remetente, sabendo que a chave usada será a que o destinatário obteve;
- O remetente recebe do destinatário o identificador da chave a ser usada, então sabe qual delas será usada para a troca de mensagens. Após isso as duas partes já possuem a chave enquanto o atacante, no pior caso, terá que solucionar todos os "quebra-cabeças" enviados pelo canal.

É possível ver que, enquanto para aqueles legitimamente envolvidos na comunicação o tempo de processamento se dá pelo envio de n "quebra-cabeças" e a complexidade m de solucionar um deles, ou seja, $O(m+n)$. Para o atacante a complexidade no pior caso é aquela na qual o atacante precisa solucionar todos os "quebra-cabeças" para encontrar a chave correta, ou seja, uma complexidade $O(m \times n)$, ou ainda, em um cenário onde $m = n$, $O(n^2)$.

Como já fora mencionado anteriormente, essa técnica serviu apenas para ilustrar a necessidade de um algoritmo facilmente aplicável pelas partes interessadas mas que fosse muito mais complexo para um outro indivíduo não autorizado. Foi constatado que uma complexidade quadrática não era o suficiente para garantir a proteção de dados contra um ataque, mas sugeriu que a existência de um algoritmo onde a complexidade para o atacante fosse exponencial ainda pudesse ser encontrado (MERKLE, 1978), e então uma comunicação segura poderia ser estabelecida em um canal inseguro. Com essa premissa, o algoritmo de troca de chaves *Diffie-Hellman* foi desenvolvido alguns anos depois, e logo após, o *RSA*, um sistema de criptografia de chave pública.

2.4 Criptografia Assimétrica

No passado, quando surgiu uma preocupação maior com a privacidade na rede, foram propostos técnicas a fim de garantir a encriptação dos dados. Contudo, um grande problema era a questão da decriptação. Para garantir o ocultamento da informação era necessário se usar algum algoritmo criptográfico que tornasse infactível a computação dos dados por um terceiro indesejado, e para tal, deveria haver uma chave para o segredo da encriptação.

Um grande problema para essa solução de comunicação privada era o compartilhamento da chave. Estabelecer um meio de comunicação seguro para o compartilhamento da chave era ineficiente e tornava a própria criptografia de dados praticamente desnecessária. Parte da solução descrita por (MERKLE, 1978) é de fato utilizada hoje em dia, considerando que o canal para o compartilhamento da chave não fosse secreto, e sim um canal público onde qualquer pessoa indesejada pudesse obter as informações que passassem através do mesmo. A premissa era que o remetente da mensagem, que também era a parte que realizaria o processo de encriptação, forneceria a chave de modo que a decriptação por parte do destinatário seria feita de maneira muito mais eficiente do que um atacante poderia realizá-la. Merkle não forneceu no seu trabalho uma troca realmente eficiente e segura, mas exemplificou a ideia principal de uma distribuição de chaves através de um canal não seguro onde a decriptação teria um alto custo para o atacante enquanto seria factível para o destinatário. Essa ideia foi o que motivou trabalho seguintes a encontrar um meio eficaz e seguro de realizar a distribuição de chaves.

Em 1976, Whitfield Diffie e Martin Hellman publicaram o artigo que motivou a pesquisa sobre criptografia de chave pública com um algoritmo eficaz para a época, que

atingia a complexidade exponencial imaginada pelo trabalho de Merkle, que inclusive foi usado como uma das fontes de pesquisa por Diffie e Hellman (DIFFIE; HELLMAN, 1976). O artigo propôs uma definição de sistema criptográfico de chave pública baseada em algumas propriedades que serão apresentadas neste capítulo, mas que foi adotada na confecção de sistemas futuros, como *ElGamal*. O próprio protocolo *DH* proposto foi adotado para troca de chaves, apesar de hoje em dia também se adotar outros algoritmos tão populares quanto, incluindo versões modificadas de *DH* para lidar com o poder computacional atual e futuro, como versões que utilizam curvas elípticas.

Desde então, criptografia assimétrica ou de chave pública vem sendo usada no dia-a-dia sempre que se utiliza aplicações visando comunicação segura, e diversos outros algoritmos foram criados para utilizar criptografia assimétrica para garantir um meio seguro de comunicação, dentre eles um dos mais famosos da última década, o *RSA*, o qual será discutido no capítulo 3.

2.4.1 Esquema Híbrido

Na grande maioria dos casos, esquemas baseados em chave pública são utilizados para realizar uma troca de chaves, enviando poucas mensagens encriptadas utilizando os algoritmos de criptografia assimétrica. A maior parte da comunicação utiliza a chave compartilhada para um algoritmo criptográfico simétrico que é responsável por encriptar e decriptar o restante das mensagens em uma comunicação longa, devido ao tamanho mais compacto das chaves e, normalmente, das cifras geradas pela encriptação simétrica.

2.5 Criptografia Homomórfica

O uso de homomorfismos na criptografia surgiu como um conceito apresentado em uma publicação do ano de 1978 do trio do *Massachussets Institute of Technology*, composto por Ronald Rivest, Len Adleman e Michael Dertouzos. A ideia é que, apesar de operar em um hardware comum, a operação de encriptação permitirá o sistema do computador operar nos dados sem descriptografá-los até que o resultado final chegue no destino(ADLEMAN; ADLEMAN; DERTOUZOS, 1978).

Os autores do artigo citado fizeram a suposição da necessidade desse tipo de criptografia ao trabalhar sobre dados de um banco de dados em computadores remotos,

pois questionaram a segurança de expor os dados secretos para um servidor remoto com o fim de realizar computação sobre os mesmos.

O nome da criptografia homomórfica, ou encriptação homomórfica, é oriundo do conceito de homomorfismo, que é um mapeamento h que preserva a estrutura entre duas estruturas algébricas A e B, denotado:

$$h : A \longrightarrow B$$

No caso aplicado à criptografia o homomorfismo pode ser uma função $\mathcal{E}$ de encriptação, e as estruturas algébricas podem ser dois grupos $(G, *)$ e $(H, *')$, sendo G o conjunto que representa o espaço das mensagens em texto claro contendo elementos $g \in G$, e H o conjunto que representa o espaço das mensagens encriptadas contendo elementos $\mathcal{E}(g) = h \in H$, tal que:

$$\mathcal{E} : (G, *) \longrightarrow (H, *') \tag{1}$$

De acordo com a equação 1, sendo que $\mathcal{E}$ preserva a estrutura dos elementos, se tem que, para $g_1, g_2, g_1 * g_2 \in G$ e $\mathcal{E}(g_1), \mathcal{E}(g_2), \mathcal{E}(g_1) *' \mathcal{E}(g_2) \in H$:

$$\mathcal{E}(g_1 * g_2) = \mathcal{E}(g_1) *' \mathcal{E}(g_2)$$

Um sistema completamente homomórfico, o primeiro tendo sido criado e demonstrado por (GENTRY, 2009), é um sistema que possui suporte a qualquer número de multiplicações e adições, pois tais operações podem ser usadas para representar operações binárias de *AND* e *XOR*, respectivamente. A partir de portas *AND* e *XOR* é possível representar a porta *NAND*, onde portas *NAND* podem ser usadas para representar qualquer outro circuito. Tal sistema proposto era ineficiente em termos de tempo porém serviu como prova de conceito da possibilidade de criar um sistema criptográfico completamente homomórfico. Mesmo com o desenvolvimento de novas técnicas, a criptografia de sistemas completamente homomórficos continua sendo pouco eficiente comparada a algoritmos de criptografia que são utilizados na atualidade.

A maioria dos sistemas com propriedades homomórficas não são completamente homomórficos mas sim parcialmente homomórficos, normalmente com suporte apenas a alguma operação específica que permite um homomorfismo no espaço do dado criptografado. A grande maioria dos algoritmos com propriedades homomórficas são algoritmos de criptografia de chave pública.

No capítulo 4, serão vistas em detalhes as propriedades homomórficas de alguns algoritmos de chave pública.

3 ALGORITMOS DE ENCRIPTAÇÃO

Nesse capítulo serão vistos algoritmos de encriptação baseados em criptografia de chave simétrica, bem como algoritmos baseados em criptografia de chave assimétrica ou pública. Ainda serão vistas propriedades homomórficas de algoritmos para trabalharmos com criptografia homomórfica parcial.

3.1 Advanced Encryption System

É um algoritmo de cifragem derivado da cifra de Rjindael. É o algoritmo mais popular na área de criptografia simétrica, e utilizado pelo governo americano em documentos classificados como secretos(NIST, 2001). O *AES* é uma cifra de bloco, atuando sobre blocos de 128 *bits*, e chaves de tamanho 128, 192 e 256 *bits*.

O *AES* é composto de quatro etapas principais, sendo elas a expansão de chave, a rodada inicial, uma etapa com um número $n-1$ de rodadas processadas diferentemente da rodada inicial e final, onde n é definido no início do algoritmo de acordo com o tamanho da chave, e, por fim, a última etapa é a rodada final.

Antes da rodada inicial, a rotina *KeyExpansion*, responsável pela expansão da chave é executada. A chave secreta inicialmente é representada por uma matriz com um número de colunas fixo $Nb = 4$ independente do tamanho da chave, e um número de linhas $Nk = 4$ e rodadas $Nr = 10$ para um chave secreta de 128 $bits$, $Nk = 6$ e $Nr = 12$ para uma chave de 192 *bits*, e $Nk = 8$ e $Nr = 14$ para uma chave secreta de 256 *bits*. A rotina expande a chave a fim de gerar uma chave expandida composta de chaves de 128 *bits*, uma para cada rodada do algoritmo, enumeradas de 0 (rodada inicial) à Nr (rodada final), totalizando um número de palavras de $Nw = (Nr + 1) \times Nb$. Os primeiros $Nb \times Nk \times 8$ *bits* da chave expandida são a chave secreta original e os *bits* seguintes são gerados conforme o seguinte algoritmo:

```
KeyExpansion
Input : keySize // tamanho da chave em bytes
        key // chave secreta de tamanho keySize bytes
Output: expKey // vetor de bytes contendo a chave expandida
BEGIN
```

```
for iKey :=0 to keySize-1 do: //chave secreta no inicio
  expKey[iKey] := key[iKey]

nb := 4 // tamanho da palavra AES
i  := 1 // round inicial de RCON

if      keySize = 16:   nr := 10
else if keySize = 24:   nr := 12
else if keySize = 32:   nr := 14

while(iKey < keySize*(nr+1)):// gera Nr chaves de Nk*Nb bytes
  for iTemp := 3 downto 0:
    tmp[iTemp] := expKey[iKey-iTemp-1]

  tmp := ROTL(tmp) //rotate para a esquerda de 1 byte

  for iTemp := 0 to 3: //SBOX aplicada a cada byte da palavra
    tmp[iTemp] := SBOX[tmp[iTemp]]

  tmp := tmp[0] XOR RCON(i)

  i := i+1 // incrementa o i utilizado em RCON

  for iTemp := 0 to 3 do:
    expKey[iKey] := tmp[iTemp] XOR expKey[iKey-keySize]
    iKey         := iKey+1

  //Cada nova palavra AES resulta do XOR da palavra anterior
  //gerada com a palavra que inicia keySize bytes antes
  for iTemp :=0 to 11:
    expKey[iKey] := expKey[iKey-nb] XOR expKey[iKey-keySize]
    iKey         := iKey+1

  if keySize=32: //caso especial de iKey mod 8 = 0
    for iTemp :=1 to nb:
```

```
        expKey[iKey] := SBOX[expKey[iKey-nb]]
        iKey         := iKey+1

    //completa os Nk*Nb-iKey bytes restantes da chave da rodada
    while ((iKey mod keySize) <> 0):
    for iTemp :=0 to 3: //0, 2 ou 3 vezes conforme keySize
      expKey[iKey] := expKey[iKey-nb] XOR expKey[iKey-keySize]
END
```

Os valores da função *RCON* normalmente são pré-computados e armazenados em uma *lookup-table* para otimizar o algoritmo, já que o limite é o valor mínimo de entrada da função é 0 e o máximo é 255, sendo que o *AES* utiliza de entrada para a função *RCON* um valor $1 \leq x \leq 10$. A rotina de *RCON*, na verdade, suporta valores até 255, porém no *AES* o máximo usado é 10.

Após a rotina *KeyExpansion*, vem as rodadas inicial, e as demais Nr rodadas, onde dentro de cada rodada, existem algumas rotinas definidas que são necessárias para o algoritmo, sendo elas nomeadas: *SubBytes*, *ShiftRows*, *MixColumns* e *AddRoundKey*. Apenas as rodadas intermediárias utilizam as 4 rotinas, na mesma ordem em que foram apresentadas, enquanto a rodada inicial utiliza apenas a rotina de *AddRoundKey* e a rodada final não possui a rotina *MixColumns*. Cada uma dessas rotinas é executada sobre uma matriz de estado que inicialmente é o texto claro, inicialmente alterado com a utilização da chave secreta. A ordem completa de operações é melhor ilustrada na figura 4.

A rotina das rodadas é a rotina *SubBytes* que nada mais é do que uma substituição do *byte* de acordo com uma chamada *S-Box*, que seria uma matriz quadrada que serve como *look-up table* de *bytes*, ou seja, simplesmente o mapeamento de um *byte* para outro. A *S-Box* foi projetada para ser resistente à criptanálise diferencial e linear (DAEMEN; RIJMEN, 2002) utilizando um corpo finito de Galois GF, dado que em $GF(2^8)$ se tem:

$$\frac{GF(2^8)}{(x^8 + x^4 + x^3 + x + 1)}$$

Seja b_i um *bit* de um *byte* onde $0 \leq i \leq 7$, sendo b_0 é o *bit* menos significativo e b_7

o mais significativo, um *byte* qualquer é representado em $GF(2^8)$ da seguinte forma:

$$\sum_{i=0}^{7} b_i \times x^i = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0$$

O cálculo inicial da *S-Box* é simplesmente o inverso modular da representação polinomial do *byte* de entrada em módulo do polinômio irredutível usado no algoritmo, $x^8 + x^4 + x^3 + x + 1$, porém, para acabar com a propriedade matemática de se obter o valor de entrada pela simples computação do inverso multiplicativo no campo finito em que os cálculos são realizados, utiliza-se a seguinte operação para realizar uma transformação afim sobre um vetor que representa os *bits* b_i, $0 \leq i \leq 7$ do *byte* resultante da operação de inverso multiplicativo, e então obter o resultado da *S-Box* de *Rjindael*:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

ShiftRows é a rotina que realiza uma operação de rotação à esquerda em cada uma das n linhas da matriz de estado. Para cada linha $0 < i \leq n$ é realizada a operação de rotação à esquerda $i - 1$ vezes, onde todos os *bytes* são deslocados uma posição à esquerda, e o primeiro antigo primeiro *byte* ocupa a posição do último *byte*. O processo é descrito de forma visual na figura 1.

Figura 1: Rotina ShiftRows

$$\begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \xrightarrow{\text{Sem rotação}} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \xrightarrow{\text{RotateLeft 1}} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \xrightarrow{\text{RotateLeft 2}} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \xrightarrow{\text{RotateLeft 3}} \end{bmatrix} \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,1} & b_{1,2} & b_{1,3} & b_{1,0} \\ b_{2,2} & b_{2,3} & b_{2,0} & b_{2,1} \\ b_{3,3} & b_{3,0} & b_{3,1} & b_{3,2} \end{bmatrix}$$

A rotina *MixColumns* é uma operação onde a matriz estado é usada como operando em uma operação de produto escalar entre polinômios, onde a segunda matriz é uma

matriz fixa definida pela multiplicação de dois polinômios de quatro termos em módulo do polinômio $x^4 + 1$, entre o polinômio fixo $a(x) = 3x^3 + x^2 + x + 2$ e o polinômio formado por quatro *bytes* b onde:

$$b(x) = \sum_{i=0}^{3} b_i \times x^i$$

A matriz que representa a multiplicação escalar entre $a(x)$ e $b(x)$ pode ser visualizada na equação 2, enquanto é possível criar *look-up tables* que contém o resultado da multiplicação para todos os valores de *bytes* possíveis para cada coeficiente de $a(x)$, como visto nas figuras 2 e 3.

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \times \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{2}$$

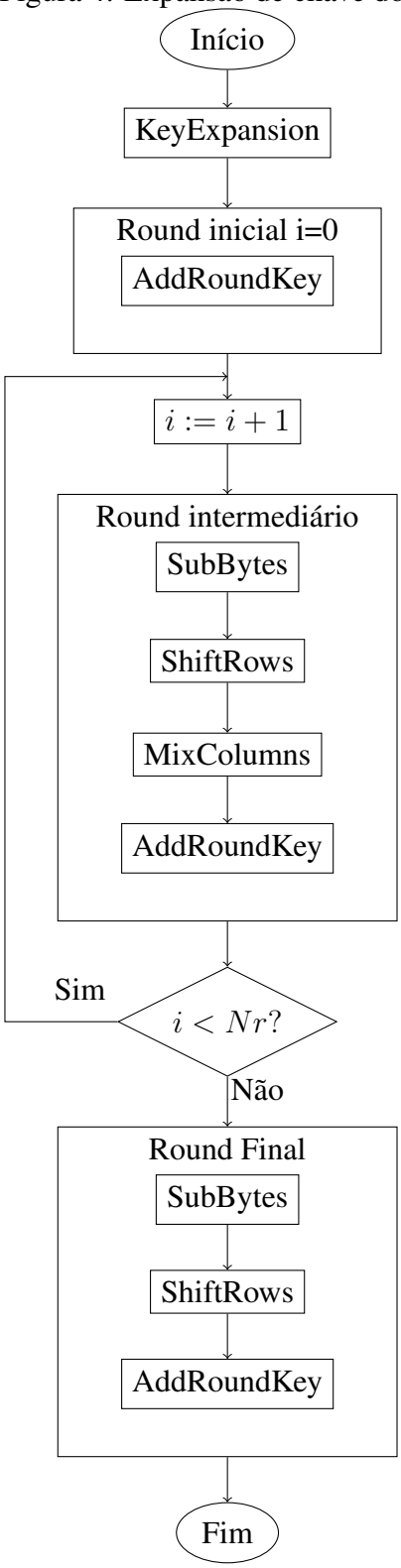
Figura 2: *Look-up table* de resultados para todas as entradas possíveis de byte multiplicado por 2 em $GF(2^8)$

$$\begin{bmatrix}
00 & 02 & 04 & 06 & 08 & 0A & 0C & 0E & 10 & 12 & 14 & 16 & 18 & 1A & 1C & 1E \\
20 & 22 & 24 & 26 & 28 & 2A & 2C & 2E & 30 & 32 & 34 & 36 & 38 & 3A & 3C & 3E \\
40 & 42 & 44 & 46 & 48 & 4A & 4C & 4E & 50 & 52 & 54 & 56 & 58 & 5A & 5C & 5E \\
60 & 62 & 64 & 66 & 68 & 6A & 6C & 6E & 70 & 72 & 74 & 76 & 78 & 7A & 7C & 7E \\
80 & 82 & 84 & 86 & 88 & 8A & 8C & 8E & 90 & 92 & 94 & 96 & 98 & 9A & 9C & 9E \\
A0 & A2 & A4 & A6 & A8 & AA & AC & AE & B0 & B2 & B4 & B6 & B8 & BA & BC & BE \\
B0 & C2 & C4 & C6 & C8 & CA & CC & CE & D0 & D2 & D4 & D6 & D8 & DA & DC & DE \\
E0 & E2 & E4 & E6 & E8 & EA & EC & EE & F0 & F2 & F4 & F6 & F8 & FA & FC & FE \\
1B & 19 & 1F & 1D & 13 & 11 & 17 & 15 & 0B & 09 & 0F & 0D & 03 & 01 & 07 & 05 \\
3B & 39 & 3F & 3D & 33 & 31 & 37 & 35 & 2B & 29 & 2F & 2D & 23 & 21 & 27 & 25 \\
5B & 59 & 5F & 5D & 53 & 51 & 57 & 55 & 4B & 49 & 4F & 4D & 43 & 41 & 47 & 45 \\
7B & 79 & 7F & 7D & 73 & 71 & 77 & 75 & 6B & 69 & 6F & 6D & 63 & 61 & 67 & 65 \\
9B & 99 & 9F & 9D & 93 & 91 & 97 & 95 & 8B & 89 & 8F & 8D & 83 & 81 & 87 & 85 \\
BB & B9 & BF & BD & B3 & B1 & B7 & B5 & AB & A9 & AF & AD & A3 & A1 & A7 & A5 \\
DB & D9 & DF & DD & D3 & D1 & D7 & D5 & CB & C9 & CF & CD & C3 & C1 & C7 & C5 \\
FB & F9 & FF & FD & F3 & F1 & F7 & F5 & EB & E9 & EF & ED & E3 & E1 & E7 & E5
\end{bmatrix}$$

Figura 3: *Look-up table* de resultados para todas as entradas possíveis de byte multiplicado por 3 em $GF(2^8)$

$$\begin{bmatrix}
00 & 03 & 06 & 05 & 0C & 0F & 0A & 09 & 18 & 1B & 1E & 1D & 14 & 17 & 12 & 11 \\
30 & 33 & 36 & 35 & 3C & 3F & 3A & 39 & 28 & 2B & 2E & 2D & 24 & 27 & 22 & 21 \\
60 & 63 & 66 & 65 & 6C & 6F & 6A & 69 & 78 & 7B & 7E & 7D & 74 & 77 & 72 & 71 \\
50 & 53 & 56 & 55 & 5C & 5F & 5A & 59 & 48 & 4B & 4E & 4D & 44 & 47 & 42 & 41 \\
C0 & C3 & C6 & C5 & CC & CF & CA & c9 & D8 & DB & DE & DD & D4 & D7 & D2 & D1 \\
F0 & F3 & F6 & F5 & FC & FF & FA & F9 & E8 & EB & EE & ED & E4 & E7 & E2 & E1 \\
A0 & A3 & A6 & A5 & AC & AF & AA & A9 & B8 & BB & BR & BD & B4 & B7 & B2 & B1 \\
90 & 93 & 96 & 95 & 9C & 9F & 9A & 99 & 88 & 8B & 8E & 8D & 84 & 87 & 82 & 81 \\
9B & 98 & 9D & 9E & 97 & 94 & 91 & 92 & 83 & 80 & 85 & 86 & 8F & 8C & 89 & 8A \\
AB & A8 & AD & AE & A7 & A4 & A1 & A2 & B3 & B0 & B5 & B6 & BF & BC & B9 & BA \\
FB & F8 & FD & FE & F7 & F4 & F1 & F2 & E3 & E0 & E5 & E6 & EF & EC & E9 & EA \\
CB & C8 & CD & CE & C7 & C4 & C1 & C2 & D3 & D0 & D5 & D6 & DF & DC & D9 & DA \\
5B & 58 & 5D & 5E & 57 & 54 & 51 & 52 & 43 & 40 & 45 & 46 & 4F & 4C & 49 & 4A \\
6B & 68 & 6D & 6E & 67 & 64 & 61 & 62 & 73 & 70 & 75 & 76 & 7F & 7C & 79 & 7A \\
3B & 38 & 3D & 3E & 37 & 34 & 31 & 32 & 23 & 20 & 25 & 26 & 2F & 2C & 29 & 2A \\
0B & 08 & 0D & 0E & 07 & 04 & 01 & 02 & 13 & 10 & 15 & 16 & 1F & 1C & 19 & 1A
\end{bmatrix}$$

Figura 4: Expansão de chave do AES.



Finalmente, a rotina *AddRoundKey* utiliza os 128 *bits* da chave expandida correspondentes a rodada i atual, onde a rodada inicial é a rodada $i = 0$ e a rodada final é a rodada $i = Nr$, sendo esses os *bits* contando a partir da posição $i \times 128$ até a posição

$i \times 128 + 127$ da chave expandida, e realiza uma operação de *XOR bit* a *bit* com o estado atual da chave, sendo o estado da rodada 0 igual ao bloco de 128 *bits* em texto claro que se deseja encriptar.

A decriptação é um processo com os mesmos passos onde a entrada de texto claro é substituída pela entrada de texto cifrado, e as operações são feitas a partir de operações inversas canônicas do *AES*, como pode ser visto em (NIST, 2001), sempre utilizando a mesma chave usada na encriptação, de forma simétrica.

3.2 Troca de Chaves Diffie-Hellman

A metodologia para a troca de chaves proposta por Diffie e Hellman se baseou fundamentalmente na ideia de Merkle de usar um canal não seguro e uma função que fosse computável para as partes relacionadas a comunicação mas difícil para atacantes.

Diffie-Hellman se baseia no problema do logaritmo discreto, que é dado por encontrar o valor de x tal que $b^x = a$. O algoritmo usa propriedades exponenciais para gerar chaves públicas de duas partes e a partir das mesmas gerar chaves compartilhadas, a partir da chave pública de cada um e de sua respectiva chave privada (DIFFIE; HELLMAN, 1976). Seja $\{k\}$ o espaço de chaves usadas na encriptação e decriptação, o algoritmo *DH* de troca de chaves é descrito como um par de famílias $\{E_k\}_{k\in\{k\}}$ e $\{D_k\}_{k\in\{k\}}$ de algoritmos representando transformações reversíveis:

$$E_k : \{m\} \longrightarrow \{m\}$$

$$D_k : \{m\} \longrightarrow \{m\}$$

1. $\forall k \in \{k\}$, E_k é a função inversa de D_k;
2. $\forall k \in \{k\}$ e $\forall m \in \{m\}$, os algoritmos E_k e D_k são facilmente computáveis;
3. Para quase todo $k \in \{k\}$, a derivação de qualquer algoritmo facilmente computável equivalente a D_k, a partir de E_k, é computacionalmente infactível;
4. $\forall k \in \{k\}$, é computacionalmente factível a computação de E_k e D_k a partir de k.

Devido a terceira propriedade, a transformação E_k pode ser compartilhada em qualquer canal não seguro pois, com um k devidamente escolhido, D_k não será conhecida por um atacante nem derivável a partir de E_k.

Diffie e Hellman propuseram uma técnica para encriptação e decriptação nesse

sentido a partir da criação de uma chave pública e uma chave privada, que são geradas, nessa proposta, através da escolha de 3 parâmetros, p, g, x. Essa escolha se dá escolhendo um p aleatório que seja primo e forme um campo finito $GF(p)$, também descrito como $\mathbb{Z}_p$, e então são escolhidos um x aleatório, tal que $1 < x < p - 1$,e um g aleatório tal que $g \in GF(p)$ e então gerar um valor privado s_i do usuário i tal que s_i é simplesmente o valor do x escolhido, e um valor y_i também do usuário i que é calculado usando os parâmetros definidos anteriormente da seguinte forma:

$$y_i = g^x \pmod p = g^{s_i} \pmod p$$

Dessa forma, um usuário j querendo enviar mensagens para o usuário i, deve ter seus próprios valores y_j e s_j, e para isso, o usuário i deixa o conjunto p, g, y_i disponível no início da comunicação ou em algum arquivo público qualquer, pois é sua chave pública, assim o usuário j pode calcular seus valores y_j e s_j,com as informações de p e g.

Para obter uma chave k_{ij} compartilhada entre os dois usuários que se comunicam, também chamada de chave de sessão, o usuário j deve iniciar a comunicação com o usuário i obtendo a chave pública de i para poder calcular a chave de sessão k_{ij}, sendo a mesma calculada da seguinte forma:

$$k_{ij} = y_i^{s_j} \pmod p = g^{s_i s_j} \pmod p$$

O usuário j então envia sua contribuição para que o usuário i possa gerar a chave da sessão que será compartilhada pelos dois, enviando $g^{s_j} = y_j$ para o usuário i que pode então calcular a chave k_{ij} compartilhada entre os dois:

$$k_{ji} = y_j^{s_i} \pmod p = g^{s_j s_i} \pmod p = g^{s_i s_j} \pmod p = y_i^{s_j} \pmod p = k_{ij}$$

Os dois usuários finalmente obtém uma chave que os dois possam compartilhar e usar como entrada para criptografar mensagens entre os dois de forma que qualquer atacante que esteja espionando a comunicação no canal não seguro não possa obter o conteúdo das mesmas por ser computacionalmente infactível adivinhar a mensagem ou a chave utilizada. Essa premissa é baseada na suposição de que quebrar o algoritmo de troca de chaves *DH* seria tão difícil quanto resolver o problema do logaritmo discreto para grandes números. Essa conjectura ainda não foi quebrada desde sua proposição, e acredita-se fortemente que ela seja válida e então realmente o esquema de chaves públicas

de *DH* só possa ser quebrado caso algum surja algum algoritmo que torne o cálculo do logaritmo discreto computacionalmente factível, pois já foi provado que $DLP \leq_p CDH$, onde o problema de resolver *CDH* é dado por encontrar um algoritmo que encontre em tempo polinomial, dados g, g^{s_i}, g^{s_j}, o valor de $g^{s_i s_j}$. Como pode-se observar, resolvendo o problema do logaritmo discreto para g^{s_i} ou g^{s_j}, seria possível obter uma das chaves secretas e então seria possível calcular a chave compartilhada combinando-a com a chave pública da qual a chave secreta não foi derivada. Apesar disso, não há nada que indique que a resolução de *CDH* ajude a resolver o logaritmo discreto para g^{s_i} e g^{s_j}.

3.3 ElGamal

Após o desenvolvimento do *RSA*, Taher ElGamal desenvolveu um algoritmo de chave pública que com a complexidade de decriptação baseada na solução do logaritmo discreto sobre campos finitos (ELGAMAL, 1985), fortemente inspirado pelo algoritmo da troca de chaves *Diffie-Hellman*.

ElGamal é baseado no problema de logaritmo discreto, o qual se acredita ser um problema difícil, e se difere de *DH* por ser uma algoritmo de encriptação e não de troca de chaves, sendo que ElGamal mostrou, através da utilização do sistema de *DH* para criar um sistema de criptografia de chave pública e assinaturas digitais, que para tal não havia a necessidade da utilização uma função simples de ser computada porém difícil de se encontrar a inversa sem uma informação especial, também chamada de função arapuca (ou *trapdoor function*, em inglês).

A diferença entre o *ElGamal* e *DH* é puramente conceitual, uma vez que *DH* é um protocolo de troca de chaves interativo, enquanto no *ElGamal* é um sistema de encriptação onde a troca das informações pode se dar em momentos bem separados no tempo, além de fornecer uma técnica de encriptação específica para ser usada com a chave pública de um usuário. Enquanto no protocolo para a troca de chaves as duas partes da comunicação devem interagir em tempo real para então um dos usuários obter a informação pública do outro, gerando uma chave de sessão, no sistema criptográfico de *ElGamal*, um usuário x pode gerar sua chave pública p, g, g^{s_x} baseada nos parâmetros de *DH*, para que se inicie uma comunicação através de mensagens criptografadas pelo algoritmo estabelecido pelo sistema com o uso da chave compartilhada. Um usuário y que deseje se comunicar com o usuário x, toma conhecimento, no momento da comunicação, da chave pública do usuário x, e então com base no gerador g da chave pública de x pode gerar um valor aleatório

$1 < s_y < g$ e com isso derivar a chave compartilhada k_{xy} que vai cifrar as mensagens, onde:

$$k_{xy} = g^{s_x s_y} \pmod{p}$$

Com seu valor s_y, o gerador g e a chave k_{xy}, o usuário y pode criar uma mensagem cifrada $c = \{g^{s_y}, k_{xy} \times m\}$, sendo m a mensagem em texto claro que y deseja enviar para x. É notável que a chave secreta é computada da mesma forma que a chave de sessão em *DH*, mas diferentemente de *DH* a chave é usada para encriptar o texto com uma operação de multiplicação entre ela e o texto claro, e não como entrada para um algoritmo qualquer de encriptação como na troca de chaves. Ao receber $c = \{g^{s_y}, k_{xy} \times m\}$ do usuário y, o usuário x também computa a chave de encriptação $k_x y$ utilizando sua chave privada s_x e com o parâmetro g^{s_y} recebido do usuário y, utilizando então k_{xy} para obter M onde:

$$m = \frac{k_{yx} \times m}{k_{xy}} = \frac{g^{s_y s_x} \times m}{g^{s_x s_y}} \pmod{p}$$

Uma comunicação dessa forma é de uma via, ou seja, todo o processo realizado é para que um usuário y envie mensagens encriptadas para x. Nesse caso, para y enviar mensagens encriptadas com *ElGamal* para y, o usuário x deve obter a chave pública de y, e todo o processo anterior é repetido onde os papéis de x e y como destinatário e remetente dos dados são trocados entre si.

A dificuldade para um atacante quebrar o sistema de encriptação está na dificuldade de solucionar o problema do *CDH*, que acredita-se ser equivalente a dificuldade do logaritmo discreto, como em *DH*, visto que o cálculo para a chave compartilhada é o mesmo que o cálculo para a chave de sessão na troca de chaves.

3.3.1 ElGamal Exponencial

O sistema *ElGamal* tem algumas variações além daquela que deu origem ao artigo, sendo uma delas o ElGamal exponencial, que trabalha com mais exponenciações, onde existindo usuários x e y, uma mensagem m em texto claro, um gerador g para a chave pública de x, para que y se comunique com x, y obtém a chave pública $\{p, g, g^{s_x}\}$ de x e então computa a chave compartilhada $k_{xy} = g^{s_x s_y}$ e a mensagem cifrada c, onde:

$$c = \{g^{s_y}, k_{xy} \times g^m\}$$

A decriptação ocorre então quando x recebe c, extraindo os elementos do conjunto, calculando a chave compartilhada $k_{xy} = g^{s_y s_x}$ e então gerando a mensagem original m conforme a seguinte equação:

$$m = log_g(\frac{k_{yx} \times g^m}{k_{xy}}) \pmod{p}$$

$$m = log_g(g^m) \pmod{p}$$

3.4 Rivest–Shamir–Adleman

O sistema *RSA* foi um dos primeiros sistemas práticos propostos de criptografia assimétrica, proposto em um artigo publicado em 1977 pelos cientistas da computação Ron Rivest e Adi Shamir, e pelo matemático Leonard Adleman. Apesar de o matemático inglês Clifford Cocks ter descrito um algoritmo similar no ano de 1973, a sua descoberta só foi revelada em 1997, sendo mantida em segredo anteriormente a esta data, e após a publicação do algoritmo ele passou a ser conhecido informalmente como *Kid-RSA(KRSA)*, por se tratar de uma versão mais simples e sem implementação devido ao alto custo para o poder computacional necessário para executar o algoritmo na época que foi descrito.

O *RSA* foi patenteado em 1983 com uma patente válida por 17 anos. A empresa *RSA Security*, fundada pelos criadores do *RSA*, liberou o algoritmo em domínio público duas semanas antes do vencimento de sua patente, no ano de 2000.

Quatro passos básicos descrevem o *RSA*, sendo eles a geração de chave, distribuição de chave, encriptação e decriptação. O algoritmo tem sua segurança baseada no problema de fatoração de números inteiros, ou *Integer Factorization (IF)* e usa números inteiros de grandes magnitudes de forma a inviabilizar algoritmos eficientes que possam violar a segurança dos dados criptografados (RIVEST; SHAMIR; ADLEMAN, 1978), onde *IF* é o problema de encontrar os fatores inteiros que compõem o número, e no caso do *RSA*, fatores inteiros primos.

A geração das chaves de um usuário u_1 se dá nos seguintes passos:

1. Escolha aleatória de dois números primos grandes p e q;
2. Calcular $n = pq$;
3. Calcular a função totiente de Carmichael aplicada a n, $\lambda(n)$;
4. Escolher aleatoriamente um número inteiro e tal que $1 < e < \lambda(n)$, e $\lambda(n)$ e e são

co-primos entre si. Nesse caso, $\lambda(n) = MMC(p-1, q-1)$;

5. Determinar d, sendo d o *IMM* de e em módulo $(p-1) \times (q-1)$.

Após isso vem a etapa de distribuição de chave, onde um outro usuário u_2 obtém a chave pública de u_1, de modo a permitir a comunicação privada de u_2 para u_1, sendo a chave pública de u_1 o conjunto $\{n, e\}$. O usuário u_1 mantém a chave privada d em segredo. O usuário u_2, detendo a chave pública $\{n, e\}$ de u_1, pode então encriptar a mensagem m que ele deseja enviar para u_1 em uma mensagem cifrada c, que é calculada da seguinte maneira:

$$c = m^e \pmod{n}$$

O usuário u_1, ao receber c, passa para a fase de decifração da mensagem. Nessa fase, o usuário obtém m a partir de c utilizando sua chave secreta d e aplicando na mensagem cifrada a técnica de decifração a seguir (deve ser lembrado que e e d foram escolhidos de forma a serem inversos multiplicativos um do outro em módulo n), seja D_r a função de decriptação de *RSA*:

$$D_r(c) = c^d = m^{ed} \pmod{n} = m$$

O algoritmo se baseia na dificuldade de solucionar o problema de fatoração de inteiros, em particular, inteiros primos, o qual é conhecido por ser um problema difícil. O objetivo da fatoração de primos é descobrir dado um número inteiro x qualquer, dois números primos $a, b \in \mathbb{Z}$ tais que:

$$a \times b = x$$

Verificando a aplicação do problema no *RSA*, nota-se a relevância da dificuldade, visto que para $x = n$, solucionar o problema de fatoração de números primos forneceria a um atacante a informação exatamente de p e q, que são os valores necessários para se computar $\lambda(n)$ e descobrir o inverso multiplicativo modular de e em módulo $\lambda(n)$, a chave de decriptação d.

O algoritmo dessa forma que é descrita, a forma original, é criticado por algumas falhas, sendo a principal delas a encriptação de mensagens pequenas onde $m^e < n$, pois nesse caso, a mensagem pode ser descoberta se aplicando a raiz e fora do módulo. Além disso, as mensagens encriptadas com uma chave pública são determinísticas, onde sendo $u_x(m_i, p)$ o envio da mensagem i a partir do usuário i encriptada com a chave pública p, e sendo $\mathcal{E}_r(m, p)$ a encriptação de uma mensagem m com uma chave pública p no sistema *RSA*, para qualquer usuário y enviando uma mensagem $j = i$ com a mesma chave pública

p, $\mathcal{E}_r(m_i, p) = \mathcal{E}_r(m_j, p)$. A solução para esse problema é a aplicação de técnicas de *padding* na mensagem antes da encriptação.

Atualmente, a recomendação do *NIST* para o tamanho de chave para o *RSA* é que o n obtido a partir de p e q seja um inteiro de pelo menos 2048 *bits*.

3.5 Paillier

O sistema de *Paillier*, assim como o *RSA*, é um sistema de encriptação de criptografia de chave pública, baseado no problema de decisão de residuosidade composta, o *DCRA*. Esse é um problema que pode ser reduzido ao problema de fatoração de números inteiros, onde a resolução do problema de fatoração de inteiros também resolve o problema da residuosidade composta, ou seja, $DCRA \leq_p IF$ (PAILLIER, 1999), sendo *DCRA* definido como determinar a existência de y tal que:

$$z \equiv y^n \pmod{n^2}$$

A geração de chaves do sistema é similar a geração de chaves no *RSA* em alguns pontos, por exemplo, na seleção de dois números primos grandes diferentes um do outro, e na multiplicação entre eles, o que resulta na similaridade entre os problemas, porém os passos são diferentes, como especificado abaixo:

1. Primeiro devem ser escolhidos aleatoriamente dois números primos p e q, tal que $MDC(p \times q, (p-1), (q-1)) = 1$;
2. Após isso, se calcula $n = p \times q$ e $\lambda = MMC((p-1), (q-1))$;
3. No terceiro passo, é selecionado aleatoriamente um número $g \in \mathbb{Z}_{n^2}$, onde $|\mathbb{Z}_{n^2}|$ divide a ordem de g. Caso g não atenda o requisito, é escolhido aleatoriamente um novo $g \in \mathbb{Z}_{n^2}$, repetindo-se o terceiro passo. Caso contrário, o algoritmo prossegue para o quarto passo;
4. É calculado o inverso multiplicativo de λ, o número $\mu = (L(g^\lambda \ mod \ n^2)) - 1 \pmod{n}$, onde $L(x) = \dfrac{x-1}{n}$.

A chave pública do sistema é o par (n, g) e a chave privada o par (λ, μ). Em relação a geração da chave, a restrição de $MDC(pq, (p-1), (q-1)) = 1$ pode ser atendida ao se escolher p e q de mesmo tamanho. Além disso, ao se utilizar p e q de mesmo tamanho, é possível utilizar uma variante mais simples da geração de chave que atende aos critérios de

encriptação e decriptação, sendo os passos dessa geração alternativa de passos definidos como abaixo:

1. O primeiro passo é o mesmo descrito no algoritmo anterior de geração de chaves, garantindo que p e q sejam primos de mesmo tamanho;
2. Calcula-se $n = p \times q$ e se define $\lambda = \phi(n)$, sendo $\phi(n) = (p-1) \times (q-1)$ a função totiente de Euler aplicada em $\mathbb{Z}_n$;
3. É definido $g = n + 1$;
4. O inverso multiplicativo de λ, ϕ, é definido como sendo $\lambda = \phi(n)^{-1} \pmod{n}$.

Um usuário que deseja enviar uma mensagem m para o dono da chave pública gerada, obtém a chave pública (n, g) do usuário, e então escolhe aleatoriamente um valor $r \in \mathbb{Z}_n$, tal que $0 < r < n$, cifrando a mensagem m na mensagem cifrada c onde, seja $\mathcal{E}_p(m, g, n)$ a função de encriptação de Paillier da mensagem m com os parâmetros públicos g e n:

$$c = \mathcal{E}_p(m, g, n) = g^m \times r^n \pmod{n^2}$$

O usuário destinatário da mensagem c pode decriptar a mensagem utilizando a chave secreta (λ, μ) e obter m, seja $D_p(c, \lambda, \mu)$ a função de decriptação em *Paillier* utilizando os parâmetros privados λ e μ para decriptar a mensagem encriptada c, da seguinte forma:

$$\begin{aligned}
D_p(c, \lambda, \mu) &= D_p(\mathcal{E}_p(m, g, n), \lambda, \mu) \\
&= L(C^\lambda \; mod \; n^2) \times \mu \pmod{n} \\
&= L((g^m \times r^n)^\lambda \; mod \; n^2) \times (L(g^\lambda \; mod \; n^2))^{-1} \pmod{n} \\
&= \frac{L((g^m \times r^n)^\lambda \; mod \; n^2)}{L(g^\lambda \; mod \; n^2)} \pmod{n} \\
&= \frac{(g^m \times r^n)^\lambda \; mod \; n^2 - 1}{n} \times \frac{n}{g^\lambda \; mod \; n^2 - 1} \pmod{n} \\
&= \frac{(g^m \times r^n)^\lambda \; mod \; n^2 - 1}{g^\lambda \; mod \; n^2 - 1} \pmod{n} \\
&= \frac{(g^{m\lambda} \times r^{p \times q \times \lambda} \; mod \; n^2) - 1}{(g^\lambda \; mod \; n^2) - 1} \pmod{n} \qquad (3)
\end{aligned}$$

Pelo teorema de Carmichael, para um $n = p \times q$ onde p e q são números primos, seja um número a coprimo a n, e a função $\lambda(n)$ a função totiente de Euler, se tem que:

$$a^{\lambda(n)} \equiv 1 \pmod{n}$$

Ou seja, se tem que:

$$a^{n\lambda(n)} \equiv 1 \pmod{n^2} \tag{4}$$

A partir dessa resolução, a equação 3 pode ser simplificada:

$$D_p(\mathcal{E}_p(m,g,n),\lambda,\mu) = \left(\frac{g^{m\lambda}-1}{g^{\lambda}-1} \pmod{n^2}\right) \pmod{n} \tag{5}$$

Para poder simplificar a equação 5 é necessário observar que é possível simplificar $g^{m\lambda}$ e g^{λ}. Para isso, é necessário primeiramente provar que g é um número da forma $(1+x)^n$. A prova é apresentada abaixo:

Lema 13. $(1+n)^x \equiv 1 + n \times x \pmod{n^2} \forall x \geq 0$

Prova. A prova desse lema pode ser feita por indução. A premissa inicial é que $(1+n)^x = 1 + nx \pmod{n^2}$. O primeiro passo é provar que a igualdade é válida para o caso base, após isso apresentar o passo indutivo.
Caso base: $x = 0$

$$(1+n)^0 \equiv 1 + n \times 0 \pmod{n^2}$$

$$1 \equiv 1 \pmod{n^2}$$

Hipótese: $\forall x \geq 0, (1+n)^x \equiv (1 + n \times x) \pmod{n^2}$
Passo indutivo: $x = x + 1$, $(1+n)^{x+1} \equiv (1 + n \times (x+1)) \pmod{n^2}$

$$\begin{aligned}
(1+n)^{x+1} &= (1+n)^x \times (1+n) \\
(1+n)^x \times (1+n) &\equiv (1+nx)(1+n) \pmod{n^2} \\
(1+n)^x \times (1+n) &\equiv 1 + nx + n + n^2 \pmod{n^2} \\
(1+n)^x (1+n) &\equiv 1 + nx + n \pmod{n^2} \\
(1+n)^x (1+n) &\equiv 1 + nx + n \pmod{n^2} \\
(1+n)^{x+1} &\equiv 1 + (x+1) \times n \pmod{n^2}
\end{aligned}$$

□

Lema 14. *A ordem de $g \in \mathbb{Z}^*_{n^2}$ é múltipla de n*

Prova. Pelo teorema de Carmichael, é possível ver na equação 4 a prova do lema, visto

que $\lambda(n)$ representa o ***MMC*** da ordem de um elemento de $\mathbb{Z}_n$, o qual por sua vez é múltiplo da ordem do grupo. □

A partir dos lemas 13 e 14, a equação 5 pode ser simplificada:

$$\begin{aligned} D_p(\mathcal{E}_p(m,g,n),\lambda,\mu) &= \left(\frac{(1+n\times k\times m\times\lambda \pmod{n^2})-1}{(1+n\times k\times\lambda \pmod{n^2})-1}\right) \pmod{n} \\ &= \left(\frac{n\times k\times m\times\lambda}{n\times k\times\lambda} \pmod{n^2}\right) \pmod{n} \\ &= \frac{m}{1} \pmod{n} \end{aligned}$$

Como um requisito para o algoritmo é de que $0 \leq m < n$:

$$m \pmod{n} = m$$

A decriptação também é verificável caso a encriptação seja realizada utilizando-se das chaves geradas no segundo método de geração de chaves descrito:

$$\begin{aligned} D_p(\mathcal{E}_p(m,g,n),\lambda,\mu) &= L(C^\lambda \, mod \, n^2)\times\mu \pmod{n} \\ &= L(C^{\phi(n)} \, mod \, n^2)-1)\times\phi^{-1}(n) \pmod{n} \\ &= \frac{((g^m\times r^n)^{\phi(n)} \, mod \, n^2)-1}{n}\times\phi^{-1}(n) \pmod{n} \\ &= \frac{(g^{m\phi(n)}\times r^{n\phi} \, mod \, n^2)-1}{n}\times\phi^{-1}(n) \pmod{n} \end{aligned} \tag{6}$$

O grupo ao qual pertence mensagem cifrada C foi visto como sendo $\mathbb{Z}_{n^2} = \mathbb{Z}_{(p\times q)^2}$, sendo assim, se tem que $|\mathbb{Z}_{(p\times q)^2}| = |\mathbb{Z}_{n^2}|$ e, pela função totiente de Euler $\phi(x)$, sabe-se que:

$$|\mathbb{Z}_{(p\times q)^2}| = \phi((p\times q)^2) = (p^1\times(p-1))\times(q^1\times(q-1)) = p\times(p-1)\times q\times(q-1)$$

Além disso, seja e o elemento neutro de um grupo G e x um outro elemento tal que $x \in G$, a definição de ordem de um elemento pertencente ao grupo diz que a ordem é o menor inteiro $n > 0$ tal que:

$$x^n = e$$

Ainda, pelo teorema de Lagrange sabe-se que a ordem de um subgrupo divide a ordem do grupo de forma que existe um valor $0 < u$ tal que, para um elemento x de ordem

n pertencente a um grupo G de ordem $|G|$:

$$n \times u = |G|$$

Com a aplicação do teorema de Lagrange, é possível observar que qualquer elemento do grupo G elevado a ordem de G também resulta no elemento neutro e:

$$\begin{aligned} x^n &= e \\ x^{n(u)} &= e \\ x^{|G|} &= e \end{aligned}$$

Sabendo que $|\mathbb{Z}_{(p\times q)^2}| = \phi((p \times q)^2) = p \times (p-1) \times q \times (q-1)$ e que qualquer elemento de um grupo elevado a ordem do subgrupo gerado pelo mesmo resulta no elemento neutro, é possível simplificar a equação 6:

$$\begin{aligned} & D_p(\mathcal{E}_p(m, g, n), \lambda, \mu) \\ &= \frac{(g^{\phi(n)m} \; mod \; n^2) - 1}{n} \times \phi^{-1}(n) \pmod{n} \\ &= \frac{((n+1)^{\phi(n)m} \; mod \; (p \times q)^2) - 1}{n} \times \frac{1}{(p-1) \times (q-1)} \pmod{n} \\ &= \frac{(1 + n \times (p-1) \times (q-1) \times m \; mod \; (p \times q)^2) - 1}{n \times (p-1) \times (q-1)} \pmod{n} \\ &= \frac{n \times (p-1) \times (q-1) \times m \; mod \; (p \times q)^2}{n \times (p-1) \times (q-1)} \pmod{n} \\ &= \frac{m \; mod \; (p \times q)^2}{1} \pmod{n} \\ &= \left(m \; mod \; (p \times q)^2\right) \pmod{n} \\ &= m \pmod{n} \\ &= m \end{aligned}$$

Esse é o último algoritmo de encriptação a ser visto neste trabalho. Serão exploradas no capítulo seguinte algumas outras propriedades interessantes desse algoritmo em relação a criptografia homomórfica.

4 PROPRIEDADES HOMOMÓRFICAS

Sistemas completamente homomórficos ainda são uma área em desenvolvimento e estudo. As vantagens de um sistema de encriptação completamente homomórfico já foram mencionadas na seção 5.3.4, porém ainda há a questão da ineficiência, em relação ao tempo, nos sistemas desenvolvidos. Apesar da grande utilidade e aparente necessidade de um sistema completamente homomórfico, há muitos sistemas que poderiam se beneficiar de homomorfismos parciais, em determinadas operações e domínios, e tais homomorfismos são comumente encontrados em algoritmos de chave pública, como *Paillier* e *ElGamal*.

4.1 ElGamal

O sistema *ElGamal* apresenta homomorfismo multiplicativo, uma vez que uma operação de multiplicação com dois textos cifrados em *ElGamal* possui um homomorfismo direto com a encriptação da multiplicação dos valores em texto claro, onde sejam m_1 e m_2 duas mensagens em texto claro, $\mathcal{E}_e$ a função de encriptação de *ElGamal*, D_e a função de decriptação, sendo (p, g^x, g) os parâmetros de chave pública utilizada na comunicação, x a chave privada do destinatário e y o valor secreto gerado pelo remetente da mensagem encriptada, e g^y o valor público também gerado pelo remetente para essa comunicação, onde dada a função $\mathcal{E}_e(m, g^x, y)$ de encriptação de uma mensagem m com os parâmetros públicos g^x e privado y, se tem:

$$\mathcal{E}_e(m_1, g^x, y) \times \mathcal{E}_e(m_2, g^x, z) = \mathcal{E}_e(m_1 \times m_2, g^x, y + z)$$

E a decriptação do produto dos dados encriptados resulta na decriptação do produto dos dados em claro como abaixo:

$$D_e(\mathcal{E}_e(m_1, g^x, y) \times \mathcal{E}_e(m_2, g^x, z), g^{y+z}, x) = D_e(\mathcal{E}_e(m_1 \times m_2, g^x, y+z), g^y, x) = m_1 \times m_2 \tag{7}$$

O homomorfismo da encriptação de *ElGamal* na equação 7 acima é verificável realizando a substituição do algoritmo $\mathcal{E}_e$ pela fórmula de encriptação do *ElGamal*, dada uma chave pública $\{p, g, g^{s_x}\}$, seja $h = g^{s_x}$ onde $1 < s_x < p - 1$, e valores secretos $1 < s_y, s_z < \dfrac{(p-1)}{2}$, $\mathcal{E}_e(m_i, \beta) = \mathcal{E}_e(m_i, g^\alpha, \beta)$, $D_e(c, g^\beta) = D_e(c, g^\beta, \alpha)$, α a chave secreta do usuário detentor da chave pública e β o parâmetro privado do remetente da

mensagem, para um usuário encriptando duas mensagem, ou para dois usuários diferentes encriptando uma mensagem cada, é demonstrado abaixo:

$$\begin{aligned}
\mathcal{E}_e(m_1, s_y) \times \mathcal{E}_e(m_2, s_z) &= (g^{s_y}, g^{s_x s_z} \times m_1) \times (g^{s_y}, g^{s_x s_z} \times m_2) \pmod{p} \\
&= (g^{s_y}, h^{s_z} \times m_1) \times (g^{s_y}, h^{s_z} \times m_2) \pmod{p} \\
&= g^{s_y+s_z}, h^{s_y+s_z} \times (m_1 \times m_2) \pmod{p} \\
&= \mathcal{E}_e(m_1 \times m_2, s_y + s_z)
\end{aligned}$$

Pode-se provar que a decriptação de forma similar, substituindo D_e pelo processo de decriptação visto no capítulo anterior, onde dada a mensagem cifrada c, a decriptação ocorre de tal forma que, para mensagens duas mensagens encriptadas como descrito acima que forem recebidas se tem:

$$\begin{aligned}
D_e(c_1 \times c_2) &= D_e(\mathcal{E}_e(m_1, s_y) \times \mathcal{E}_e(m_2, s_z), g^{s_y+s_z}) \\
&= D_e(((g^{s_y}, g^{s_x s_y} \times m_1) \times (g^{s_z}, g^{s_x s_z} \times m_2)), g^{s_y+s_z}) \\
&= D_e((g^{s_y+s_z}, h^{s_y+s_z} \times (m_1 \times m_2)), g^{s_y+s_z}) \\
&= D_e(\mathcal{E}_e(m_1 \times m_2, g^{s_x}, s_y + s_z), g^{s_y+s_z}) \\
&= \frac{h^{s_y+s_z} \times m_1 \times m_2}{g^{(s_y+s_z)s_x}} \pmod{p} \\
&= \frac{g^{s_x(s_y+s_z)} \times m_1 \times m_2}{g^{(s_y+s_z)s_x}} \pmod{p} \\
&= \frac{g^{s_x(s_y+s_z)} \times m_1 \times m_2}{g^{(s_y+s_z)s_x}} \pmod{p} \\
&= m_1 \times m_2
\end{aligned}$$

4.1.1 ElGamal Exponencial

O sistema de ElGamal pode ser modificado para apresentar homomorfismo aditivo ao invés de multiplicativo, quando se utiliza a versão Exponencial do ElGamal, conforme descrita na seção 3.3.1. Sendo $\mathcal{E}_{exp}(m, g^x, s)$ a função de encriptação de *ElGamal* exponencial de uma mensagem m utilizando um parâmetro secreto s e um parâmetro público

g^x, abaixo é demonstrada a propriedade homomórfica do sistema em relação à adição:

$$\begin{aligned}
&\mathcal{E}_{exp}(m_1, g^{s_x}, s_y) \times \mathcal{E}_{exp}(m_2, g^{s_x}, s_z) \\
&= (g^{s_y}, g^{s_x s_z} \times g^{m_1}) \times (g^{s_y}, g^{s_x s_z} \times g^{m_2}) \pmod{p} \\
&= (g^{s_y}, h^{s_z} \times g^{m_1}) \times (g^{s_y}, h^{s_z} \times g^{m_2}) \pmod{p} \\
&= (g^{s_y+s_z}, h^{s_y+s_z} \times (g^{m_1+m_2})) \pmod{p} \\
&= \mathcal{E}_{exp}(m_1 + m_2, g^{s_x}, s_y + s_z)
\end{aligned}$$

Seja o texto cifrado $c = \mathcal{E}_{exp}(m, g^x, s)$ e $D_{exp}(c, h^x)$ a função de decriptação utilizando o parâmetro de sessão $h = g^y$, é possível observar abaixo que a soma decriptação aplicada a dois textos cifrados individualmente, ou seja, simplesmente a soma de dois textos em claro é igual ao resultado da decriptação da multiplicação dos textos cifrados correspondentes:

$$\begin{aligned}
&D_{exp}((\mathcal{E}_{exp}(m_1, g^{s_x}, s_y) \times \mathcal{E}_{exp}(m_2, g^{s_x}, s_z)), g^{s_y+s_z}) \\
&= D_{exp}(((g^{s_y}, g^{s_x s_y} \times g^{m_1})(g^{s_z}, g^{s_x s_z} \times g^{m_2})), g^{s_y+s_z}) \\
&= D_{exp}((g^{s_y+s_z}, h^{s_y+s_z} \times (g^{m_1+m_2})), g^{s_y+s_z}) \\
&= D_{exp}((g^{s_y+s_z}, h^{s_y+s_z} \times (g^{m_1+m_2})), g^{s_y+s_z}) \\
&= D_{exp}(\mathcal{E}_{exp}(m_1 + m_2, g^{s_x}, s_y + s_z), g^{s_y+s_z}) \\
&= log_g(\frac{h^{s_y+s_z} \times g^{m_1+m_2}}{g^{(s_y+s_z)s_x}} \pmod{p}) \\
&= log_g(\frac{g^{s_x(s_y+s_z)} \times g^{m_1+m_2}}{g^{(s_y+s_z)s_x}} \pmod{p}) \\
&= log_g(g^{m_1+m_2} \pmod{p}) \\
&= m_1 + m_2
\end{aligned}$$

4.2 RSA

O *RSA* como foi proposto originalmente também possui propriedades homomórficas multiplicativas, visto que, sejam m_1 e m_2 duas mensagens, considerando o sistema

criptográfico *RSA* como descrito na seção 3.4:

$$\begin{aligned}\mathcal{E}_r(m_1, n, e) \times \mathcal{E}_r(m_2, n, e) &= m_1^e \times m_2^e \pmod{n} \\ &= (m_1 \times m_2)^e \pmod{n} \\ &= \mathcal{E}_r(m_1 \times m_2, n, e)\end{aligned}$$

A decriptação $D_r(\mathcal{E}_r(m_1, n, e), d) \times D_r(\mathcal{E}_r(m_2, n, e), d)$ para m_1 e m_2 também possui homomorfismo para uma mensagem $m_1 \times m_2$ conforme é demonstrado abaixo:

$$\begin{aligned}D_r(\mathcal{E}_r(m_1, n, e), d) \times D_r(\mathcal{E}_r(m_2, n, e)) &= D_r(m_1^e) \times D_r(m_2^e) \pmod{n} \\ &= (m_1^e)^d \times (m_2^e)^d \pmod{n} \\ &= (m_1^e \times m_2^e)^d \pmod{n} \\ &= ((m_1 \times m_2)^e)^d \pmod{n} \\ &= D_r(\mathcal{E}_r(m_1 \times m_2, n, e), d) \\ &= m_1 \times m_2\end{aligned}$$

Porém, como mencionado na seção 3.4, o *RSA* não é seguro sem a aplicação de algum tipo de *padding*, com isso, e com a aplicação de *padding*, não é possível garantir que a demonstração acima é válida para quaisquer mensagens m_1, m_2.

4.3 Paillier

O sistema de *Paillier* também é um sistema parcialmente homomórfico aditivamente. Intuitivamente, é possível supor que, devido a mensagem ser um expoente de um gerador, seja um homomorfismo da forma $\mathcal{E}_p : x \times y \in \{m\} \longrightarrow x + y \in \{c\}$. Isso se mostra verdade ao aplicar-se o algoritmo de encriptação em duas mensagens diferentes m_1 e m_2, sendo r e s dois números aleatoriamente escolhidos onde $r, s \in \mathbb{Z}_n$, e existindo uma chave pública (g, n), $g \in \mathbb{Z}_{n^2}$:

$$\begin{aligned}\mathcal{E}_p(m_1, n, g, r) \times \mathcal{E}_p(m_2, n, g, s) &= (g^{m_1} \times r^n \; mod \, n^2) \times (g^{m_2} \times s^n \; mod \, n^2) \\ &= g^{m_1} \times r^n \times g^{m_2} \times s^n \pmod{n^2} \\ &= g^{m_1+m_2} \times (r^s)^n \pmod{n^2} \\ &= \mathcal{E}_p(m_1 + m_2, n, g, r \times s)\end{aligned}$$

Também é necessário mostrar que o resultado da decriptação de $\mathcal{E}_p(m_1{+}m_2, n, g, rs)$ é igual ao resultado da decriptação de $\mathcal{E}_p(m_1, n, g, r)$ somado ao resultado da decriptação de $\mathcal{E}_p(m_2, n, g, s)$ para provar o homomorfismo aditivo do sistema. Seja $D_p(c, \lambda, \mu)$ o processo de decriptação do texto cifrado c de *Paillier* utilizando os parâmetros de chave privada λ e μ se tem a prova do processo:

$$
\begin{aligned}
& D_p(c_1, \lambda, \mu) + D_p(c_2, \lambda, \mu) \\
& = D_p(\mathcal{E}_p(m_1, n, g, r), \lambda, \mu) + D_p(\mathcal{E}_p(m_2, n, g, s), \lambda, \mu) \\
& = L(C_1 \ mod \ n^2) + L(C_2 \ mod \ n^2) \pmod{n} \\
& = L(g^{m_1 \times r \times n \ mod \ n^2) + L(g^{m_2 \times s \times n} \ mod \ n^2)} \pmod{n} \\
& = \frac{((g^{m_1} \ mod \ n^2) - 1) + ((g^{m_2} \ mod \ n^2) - 1)}{n} \times \frac{n}{(g \ mod \ n^2) - 1} \pmod{n} \\
& = \left(\frac{(g^{\lambda m_1} - 1) + (g^{\lambda m_2} - 1)}{g^{\lambda} - 1} \pmod{n^2} \right) \pmod{n} \\
& = \left(\frac{g^{\lambda m_1} - 1}{g^{\lambda} - 1} + \frac{g^{\lambda m_2} - 1}{g^{\lambda} - 1} \pmod{n^2} \right) \pmod{n} \\
& = \left(\frac{1 + \lambda \times k \times n \times m_1 - 1}{1 + \lambda \times k \times n - 1} + \frac{1 + \lambda \times k \times n \times m_2 - 1}{1 + \lambda \times k \times n - 1} \pmod{n^2} \right) \pmod{n} \\
& = \left(\frac{\lambda \times k \times n \times m_1}{\lambda \times k \times n} + \frac{\lambda \times k \times n \times m_2}{\lambda \times k \times n} \pmod{n^2} \right) \pmod{n} \\
& = m_1 + m_2 \\
& = D_p(\mathcal{E}_p(m_1 + m_2, rs), \lambda, \mu)
\end{aligned}
$$

4.4 Sistemas Parcialmente Homomórficos

Estes são apenas alguns dos sistemas de encriptação com propriedades homomórficas da literatura. Ainda existem outros, alguns dos quais possuem a propriedade de serem completamente homomórficos, ou seja, com homomorfismos aditivo e multiplicativo no mesmo sistema, o que não é o foco deste trabalho, e sim explorar as propriedades homomórficas de sistemas populares da literatura sobre criptografia e que são utilizados há décadas. Também é clara a limitação que os sistemas apresentados possuem ao serem utilizados sem nenhum tipo de mapeamento ou com grupos diferentes dos apresentados, como, por exemplo, utilizar grupos multiplicativos inteiros e com isso tornar o sistema limitado a trabalhar com números inteiros positivos.

Além disso, ao se utilizar um sistema com homomorfismo aditivo e com as restri-

ções apresentadas, não se pode, por exemplo, implementar sistemas que trabalhem com porcentagens e cálculos mais complexos, como calculadoras científicas e sistemas bancários. Da mesma forma, um sistema que apresente apenas homomorfismo multiplicativo não é utilizável em contagens arbitrárias. De acordo com a última afirmação, é importante notar que no sistema de *Paillier*, por exemplo, quando uma constante k no domínio das mensagens em texto claro é multiplicada por um texto cifrado $\mathcal{E}_p(m)$, a decriptação resulta no produto $k \times m$, porém, isso foge da definição de homomorfismo e não é considerada criptografia homomórfica, uma vez que o dado correspondente a constante k não está encriptado.

5 APLICAÇÃO

No que segue, esse capítulo aborda temas relacionados a aplicações no mundo real que podem se beneficiar da implementação de técnicas criptográficas homomórficas, bem como a implementação de um protótipo de um desses sistemas, a qual é descrita na seção 5.3

5.1 Confidencialidade

O ponto chave da criptografia homomórfica é a confidencialidade. Diversos dados de usuários trafegam na rede, mas mesmo com a criptografia de chave pública assegurando a privacidade entre um ponto e outro, às vezes não é possível saber por quantos pontos intermediários a informação pode ser interceptada, principalmente no contexto atual da computação onde a computação em nuvem se faz muito presente. Além disso, muita informação pode ser salva e processada em um servidor remoto, como por exemplo, em sistemas que trabalham com arquitetura de cliente-servidor.

Dados encriptados que permitam operações homomórficas visam evitar a exposição do dado até mesmo para que sejam realizadas operações sobre eles, o que evitaria o problema de, por exemplo, falta de confiança em um servidor onde um indivíduo mantém suas aplicações e dados. É necessário notar, que mesmo em um servidor onde se tem a confiança de que os dados presentes no mesmo não serão acessados pelo gerenciamento do mesmo, sempre há o risco de um ataque ou exposição indevida através de, por exemplo, contaminação por vírus, e os dados encriptados são a proteção contra esse tipo de ameaça.

5.2 Exemplos de Aplicações

Na academia, a ideia para aplicações reais surgiu para que se pudesse realizar consultas a bancos de dados encriptados utilizado operações no espaço da informação encriptada desde (ADLEMAN; ADLEMAN; DERTOUZOS, 1978). De fato, o uso de criptografia homomórfica em aplicações de bancos de dados é visto na literatura como em (PRAVEEN, 2016), e também em aplicações reais como o sistema CryptDB(POPA et al., 2011). Dessa forma, consultas criptografadas podem recuperar dados criptografados em bases de dados na nuvem de maneira que apenas o cliente da requisição saiba qual a

consulta realizada e seu conteúdo.

A CipherCloud[1] é uma solução em nuvem onde os dados são encriptados antes da transferência para o servidor, onde os dados não são decriptados em momento algum enquanto no servidor, apenas quando recuperados pelo cliente, depois de deixar o servidor. A motivação para o sistema é a falta de garantia sobre a encriptação dos dados entre os servidores de uma companhia, mesmo que haja garantia de encriptação no armazenamento. Outro fator é o acesso que um administrador do sistema pode ter sobre as chaves usadas, permitindo acesso interno malicioso de alguém da empresa com permissão de administrador sobre os dados, ou até mesmo caso o acesso de algum vazamento de credenciais de alguém com tal poder sobre o servidor. A *CipherCloud* não possui qualquer indício que utilize criptografia homomórfica, no sentido de que não há informação sobre propriedades homomórficas de sistemas de encriptação utilizadas, ou seja, não é possível analisar ou assegurar matematicamente propriedades vistas nesse trabalho. Porém, é notável que um sistema que visa garantir as propriedades citadas, pode se beneficiar da implementação de sistemas de encriptação que possuam homomorfismos em todas ou algumas de suas operações.

Outra aplicação prática de homomorfismos na criptografia se dá na implementação de sistemas de votação online, como o sistema Helios[2], que é utilizado para votações da International Association for Cryptologic Research[3] (IACR), entidade sem fins lucrativos que promove estudos científicos sobre o tema de criptologia.

Em suma, com o aumento de dados trafegando pela rede com processamento distribuído e o aumento da computação e dados na nuvem, há evidentemente uma tendência da necessidade de sistemas que implementem criptografia homomórfica.

5.3 Protótipo

Essa seção visa descrever o experimento com um protótipo de programa que simula uma votação utilizando criptografia homomórfica. O sistema de encriptação escolhido para tal foi o sistema proposto por Paillier (PAILLIER, 1999), visto que o homomorfismo parcial aditivo que o sistema fornece é suficiente para implementar um esquema de votação, onde é necessária a soma de votos de candidatos para decidir um vencedor.

[1]Fonte: CipherCloud. Disponível em <http://ciphercloud.com/wp-content/uploads/2016/08/20160531-CSP-Encryption-Comparison-Guide.pdf>. Acesso em 2 de dezembro de 2017

[2]Fonte: Helios. Disponível em <https://heliosvoting.org>

[3]Fonte: IACR. Disponível em <https://www.iacr.org>

5.3.1 Tecnologias Utilizadas

A linguagem escolhida para a implementação do protótipo foi a linguagem *C++*, compilada utilizando o *GNU Compiler Collection (GCC)*. Os motivos dessa escolha:

- Linguagem compilada: diferentemente de linguagens interpretadas ou linguagens híbridas (compiladas em tempo de execução ou com código intermediário compilado), linguagens compiladas já estão codificadas em código de máquina, tendo suas instruções executadas rapidamente
- Popularidade: como uma das linguagens mais utilizadas no mundo moderno[4] é possível encontrar diversos *frameworks*, *APIs* e bibliotecas para a linguagem que podem auxiliar no desenvolvimento de uma aplicação.
- Otimização: o *GCC* oferece diversos níveis de otimização para as linguagens suportadas, no caso, *C++*, além da vantagem de ser compilada, tem as instruções refatoradas pelo compilador para otimizar o código em termos de tempo de execução sem modificar a semântica do programa (dependendo do nível de otimização).

Também foi utilizada a biblioteca *GNU Multi Precision Arithmetic Library (GMP)*, uma biblioteca para a linguagem *C*, que fornece uma interface de objetos e alguns métodos para *C++*. A biblioteca tem a finalidade de fornecer funções e estruturas para aritmética de precisão arbitrária, e foi escolhida pois não é possível utilizar, para se demonstrar um caso real de aplicação de criptografia de chave pública com os tamanhos atuais, os tipos nativos de *C++* sem uma complexa implementação de uma interface que permita trabalhar com grandes números, o que a *GMP* já fornece.

5.3.2 Voto

Segundo o Tribunal Superior Eleitoral, o número de eleitores brasileiros em 2017 foi de 146.158.440[5], o que pode ser representado em uma estrutura $log_2(146.158.440) \approx 28$ *bits*, logo, o número 146.158.440 é o máximo de votos que um candidato poderia ter em teoria. Como não é possível saber exatamente o resultado de uma eleição, todos os

[4]Fonte: TIOBE Index. Disponível em <https://www.tiobe.com/tiobe-index/>. Acesso em 2 de dezembro de 2017.

[5]Fonte: Tribunal Superior Eleitoral. Disponível em <http://www.tse.jus.br/imprensa/noticias-tse/2017/Novembro/identificacao-biometrica-atinge-mais-de-346-mil-eleitores-no-rn>. Acesso em: 01 de dezembro de 2017

candidatos devem suportar a possibilidade máxima de votos, sendo assim, conforme o exemplo acima, todo candidato deve ter um espaço $|E|$ de 28 *bits*. Para um número n_c de candidatos, o espaço necessário para a estrutura que armazena o voto é de $n_c \times |E|$ *bits*. O voto de um candidato é representado pelo *bit* menos significativo do correspondente ao total de votos de um candidato com o valor 1 e os demais *bits* tem valor 0. Em um cenário de 100 candidatos com os dados apresentados, uma estrutura de 2800 bits seria o mínimo necessário, e o valor do módulo deveria ser um número de, no mínimo de 2801 *bits*, para assegurar pelo menos um voto, crescendo conforme o número de votos totais.

O método usado para preencher os requisitos descritos foi usar uma estrutura de *bitset*, que é como um vetor de *bits*, com um tamanho de $|E| \times n$, onde, como o protótipo foi feito para lidar com n_c candidatos, sendo então um *bitset* de $n_c \times |E|$ posições, cada uma representando um *bit*. Como na linguagem *C++* as estruturas indexadas de tamanho t possuem índice $0 \leq i < t$, essa estrutura que representa um voto individual para qualquer candidato é dividida em n_c partes de $|E|$ *bits* onde o candidato $i + 1$ tem seus votos representados pelos *bits* $b_j..b_{j+|E|-1}$, $j = (|E| - i) \times n$, cujo *bit* mais significativo se encontra à esquerda e o menos significativo à direita. Abaixo exemplos de votos em uma eleição com 3 candidatos, com um máximo de 9 eleitores:

Voto de um eleitor no candidato 1: 000 000 001
Voto de um eleitor no candidato 2: 000 001 000
Voto de um eleitor no candidato 3: 001 000 000

5.3.3 Geração de chaves

Na inicialização do sistema, é definido o número de *bits* responsáveis pelo nível de segurança no sistema de *Paillier*, ou seja, o número de *bits* do parâmetro público n_p da chave pública (n_p, g), onde g é o gerador, n_p o módulo em que os cálculos são realizados e L é o tamanho em bits desejado para n_p.

A geração de chaves se dá a partir da função *jGeneratePaillierKeys*. Os parâmetros p, q e n_p, o último gerado a partir de p e q, são gerados por uma função chamada *jPaillierInitParam*, que é chamada pela *jGeneratePaillierKeys*. A partir do valor de n_p, *jGeneratePaillierKeys* são gerados os valores privados λ e μ, e o parâmetro público restante, o gerador g.

A função *jPaillierInitParam*, conforme o algoritmo abaixo, utiliza uma função

auxiliar em um laço chamada *get_random_bits*, que acessa o arquivo */dev/urandom* do sistema operacional Linux que fornece dados pseudo-randômicos utilizados em diversas implementações de bibliotecas que trabalham com criptografia. Seja L_b o número de *bits* escolhido para a chave de *Paillier*, a função é usada para selecionar $\frac{L_b}{2}$ *bits* aleatórios do arquivo para gerar um valor inicial para o parâmetro p, processo esse repetido em laço até que o *bit* mais significativo de p seja 1, e após isso, um novo laço seleciona o próximo primo provável de valor maior ou igual a p, laço esse que é interrompido quando a função da *GMP*, *mpz_probab_prime_p*, indica que o número é provavelmente um número primo. A função *mpz_probab_prime_p* é baseada no algoritmo *Miller–Rabin* (RABIN, 1980), um algoritmo que testa a primalidade de um número com uma quantidade parametrizável de iterações, onde a confiança na resposta probabilística sobre a primalidade do número cresce conforme o número de iterações. O parâmetro q é escolhido da mesma forma que o parâmetro p, e então, o parâmetro n_p é resultante do produto $p \times q$. Caso o valor n_p não tenha um número L_b de *bits*, o processo é repetido até que, em *bits*, $|n_p| = L_b$, para então retornar os valores de n_p, p e q.

```
void jPaillierInitParam(mpz_class *p, mpz_class *q,
   mpz_class *n){
  mpz_t r;
  *n = 1;
  int bits = NBITS.get_si();
  while(mpz_sizeinbase(n->get_mpz_t(),2) != bits){
    *p=4; //garante que entra no primeiro while
    *q=4; //garante que entra no primeiro while
    while(mpz_sizeinbase(p->get_mpz_t(), 2) != bits/2){
      get_random_bits(&r, bits/2);
      mpz_set(p->get_mpz_t(), r);
      mpz_clear(r);
    }
    while(mpz_probab_prime_p(p->get_mpz_t(), 100) == 0){
      mpz_nextprime(p->get_mpz_t(), p->get_mpz_t());
    }
    while(mpz_sizeinbase(q->get_mpz_t(), 2) != bits/2){
```

```
        get_random_bits(&r, bits/2);
        mpz_set(q->get_mpz_t(), r);
        mpz_clear(r);
      }
      while(mpz_probab_prime_p(q->get_mpz_t(), 100) == 0){
        mpz_nextprime(q->get_mpz_t(), q->get_mpz_t());
      }
      *n = (*p)*(*q);
    }
  }
```

A função *jGeneratePaillierKeys* retorna os valores dos parâmetros públicos n_p e g, sendo o parâmetro n_p computado conforme o algoritmo acima, e os parâmetros privados do sistema, λ e μ.

A partir de n_p, p e q retornados, é calculado o valor de $\lambda = MMC(p-1, q-1)$, e então são selecionados L *bits* de */dev/urandom* a fim de criar um valor aleatório para o gerador g, até que seja escolhido um $g \in Z_{n_p}$, o que é garantido com o $MDC(g, n_p) = 1$, utilizando a função para calcular o MDC entre dois números da GMP. Com a garantia de que $g \in Z_{n_p}$, é calculado um valor intermediário γ onde:

$$\gamma = g^{\lambda} \pmod{{n_p}^2}$$

Para computar a exponenciação em módulo, é utilizada a função *mpz_powm* da *GMP*. Após isso o valor de μ é calculado da seguinte forma:

$$\mu = \left(\frac{\gamma - 1}{n_p}\right)^{-1} \pmod{n_p} = \left(\frac{g^{\lambda} - 1 \pmod{{n_p}^2}}{n_p}\right)^{-1} \pmod{n_p}$$

É utilizada a função *mpz_invert* da *GMP* para calcular o inverso multiplicativo de um número em determinado módulo, no caso, o módulo n_p, para calcular o inverso acima. Com isso, ao fim da função *jGenereatePaillierKeys* os valores n_p, g, λ e μ estão inicializados.

```
void jGeneratePaillierKeys(mpz_class *g, mpz_class *n,
    mpz_class *lambda, mpz_class *mu){
  mpz_class p, q;
  mpz_class n2;
  mpz_t mpzAux;
  jPaillierInitParam(&p, &q, n);
  *lambda = lcm(p-1, q-1);
  int count = -1;
  n2 = ((*n)*(*n));

  do{
    get_random_bits(&mpzAux, NBITS.get_si());
    mpz_set(g->get_mpz_t(), mpzAux);
    mpz_clear(mpzAux);
  }while(gcd(*g, n2) != 1);

  mpz_powm(mu->get_mpz_t(), g->get_mpz_t(), lambda->
      get_mpz_t(), n2.get_mpz_t());

  *mu = *mu-1;
  *mu = *mu/(*n);
  mpz_invert(mu->get_mpz_t(), mu->get_mpz_t(), n->
      get_mpz_t());
}
```

No protótipo, o tamanho L_b de chave escolhido foi 4096, e o parâmetro para a função *mpz_probab_prime_p* testar a primalidade dos parâmetros p e q foi o valor 100, significando 100 iterações de *Miller–Rabin* para a decisão de se o número é um provável primo. Também é importante notar que a geração de chaves só precisa ocorrer uma vez durante a execução do sistema, deixando a complexidade do sistema em função da utilização do mesmo após a geração.

5.3.4 Encriptação

A implementação do algoritmo que encripta mensagem é feita a partir de uma função desenvolvida chamada *jPaillierEncrypt*. Essa função recebe como parâmetro os parâmetros da chave pública gerados na geração das chaves, n_p e g, e uma mensagem em texto claro m a qual se deseja encriptar.

A função recebe um número L_b de *bits* igual ao tamanho em *bits* de n_p para criar um parâmetro aleatório $k \in \mathbb{Z}_{n_p}$, ou seja, o k não deve ser um divisor de n_p, o que é garantido analisando o resto da visão $\frac{n_p}{k}$, que deve ser diferente de 0. Após escolhido o parâmetro aleatório k a mensagem m é encriptada em uma mensagem encriptada C da seguinte forma:

$$C = g^m \times k^{n_p} \pmod{{n_p}^2}$$

A função *jPaillierEncrypt* utiliza a chamada de mpz_powm da biblioteca *GMP* para realizar as exponenciações em módulo ${n_p}^2$. O retorno após as operações é um vetor de caracteres representando uma *string* que representa o valor de C.

```
char* jPaillierEncrypt(const char* generator, const char
    * message, const char* mod){
  mpz_class m(message);
  mpz_class g(generator);
  mpz_class n(mod);
  mpz_class n2=n*n;
  mpz_class r;
  mpz_class q;
  mpz_class k;
  mpz_t mpzAux;
  char* result;
  r = 0;

  do{
    get_random_bits(&mpzAux, mpz_sizeinbase(n.get_mpz_t
        (),2));
    mpz_set(k.get_mpz_t(),mpzAux);
```

```
    mpz_cdiv_qr(q.get_mpz_t(), r.get_mpz_t(), n.
        get_mpz_t(), k.get_mpz_t());//quociente q de k/n
        e o resto em r
  }while(r == 0);
  mpz_powm(g.get_mpz_t(), g.get_mpz_t(), m.get_mpz_t(),
     n2.get_mpz_t());
  mpz_powm(k.get_mpz_t(), k.get_mpz_t(), n.get_mpz_t(),
     n2.get_mpz_t());
  g = g*k;
  mpz_mod(g.get_mpz_t(), g.get_mpz_t(), n2.get_mpz_t());

  result = (char*) malloc(sizeof(char)*(g.get_str().size
     ()+1));
  strcpy(result, g.get_str().c_str());
  mpz_clear(mpzAux);
  return result;
}
```

5.3.5 Decriptação

A função responsável pela decriptação de uma mensagem C encriptada, recebe como parâmetros a mensagem C, os parâmetros λ e μ, e o parâmetro da chave pública n_p.

A decriptação que transforma a mensagem encriptada C na mensagem decriptada m é realiza da seguinte maneira:

$$m = \frac{(C^{\lambda} - 1 \pmod{n^2})}{n} \times \mu \pmod{n}$$

A função utiliza as chamadas da biblioteca *mpz_powm* para a exponenciação modular de números inteiros, e a função *mpz_mod* para garantir que o resultado das multiplicações é um número em módulo n_p. As multiplicações e divisões são realizadas com o suporte a interface para a linguagem *C++* que implementa automaticamente chamadas de funções que trabalham com as operações básicas utilizando estruturas de dados da

biblioteca.

O retorno da função, assim como na de encriptação, é um vetor de caracteres correspondente a *string* que representa a mensagem m decriptada.

```
char* jPaillierDecrypt(char* encryptedMessage, mpz_class
    privateL, mpz_class privateM, mpz_class n){
  mpz_class c(encryptedMessage);
  mpz_class n2=n*n;
  char* result;
  int i = 0;
  string stringAux;
  mpz_powm(c.get_mpz_t(), c.get_mpz_t(), privateL.
     get_mpz_t(), n2.get_mpz_t());

  c = c-1;
  c = c/n;
  c = c*privateM;
  mpz_mod(c.get_mpz_t(), c.get_mpz_t(), n.get_mpz_t());

  result = (char*) malloc(sizeof(char)*(c.get_str().size
     ()+1));
  strcpy(result, c.get_str().c_str());

  return result;
}
```

5.3.6 Determinando o vencedor

Cada voto é representado por um número binário de 2800 *bits*. Como o sistema de *Paillier* estabelece que a mensagem m encriptada deve ser estritamente menor que o parâmetro público n_p de *Paillier*, que representa o módulo usado e também é responsável por ditar a complexidade do problema. O tamanho de n_p foi variado para os testes e cada

teste usou uma chave pública diferente gerada pelo algoritmo de geração de chave pública de *Paillier*.

Cada voto realizado é encriptado com a chave pública gerada (n_p, g), e com o parâmetro aleatório r gerado por cada usuário, e então armazenado de forma a ser multiplicado com os votos seguintes. Como visto na seção 3.5, seja x o total de votos:

$$\prod_{i=1}^{x} c_i = \mathcal{E}_p\left(\sum_{i=1}^{x} m_i\right)$$

Visto que cada seção de tamanho $|E|$ representa um candidato e o máximo de votos possíveis que o mesmo pode obter, não há *overflow* na soma de modo que uma seção afete a próxima, ou seja, na soma da grande estrutura de tamanho $n_c \times |E|$, onde cada bloco de tamanho $|E|$ possui exatamente o resultado da soma de todos os votos do candidato da posição correspondente ao bloco, sendo assim, o produto total das mensagens cifradas é decriptado e então os votos de cada candidato são recolhidos percorrendo a estrutura que contém os votos, dividindo essa grande estrutura que contém todos os votos em n blocos. Após isso é necessário apenas um algoritmo simples para decidir o maior valor dentre os blocos correspondentes a cada candidato para decidir o candidato vencedor da eleição.

5.3.7 Resultados

Foram realizados testes para avaliar o desempenho do algoritmo em relação ao tamanho de mensagens encriptada e ao tempo de execução de operações com as mesmas, que podem ser vistos na tabela 2. Os testes foram realizados em um *notebook* com memória *RAM DDR3* de 8*GB* e processador *Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz*, operando a *2.6GHz*.

Os resultados das tabelas 2 e 3 resultaram de experimentos separados, onde para cada experimento foi utilizado um *script* que executou automaticamente um número N de votos no candidato de maior número, no caso 10, conforme especificado na coluna N da tabela 2, e o resultado foi descrito conforme a média de 10 execuções para cada linha de ambas as tabelas. Para esses experimentos, foi adotado um suporte para 14 *bits* de votos totais por candidato para poder suportar os 10000 votos máximos dos experimentos, ou seja, as estruturas de voto utilizadas possuem 140 *bits*. Também é importante ressaltar que o ponto (.) é usado nas tabelas como separador decimal.

Na tabela 2 abaixo, c_{min}, c_{max} e $c_{médio}$ representam, respectivamente, o menor,

o maior e a média do tamanho, em *bits*, gerado para um dado encriptado, e o produto dos dados encriptados, que representa o resultado encriptado da votação, é descrito como $\prod_{i=0}^{N} c_i$, sendo a coluna $\prod_{i=0}^{N} c_i$*(bits)* o tamanho em *bits* desse resultado.

Tabela 2: Avaliação do desempenho do tempo da multiplicação de todos os votos encriptados utilizando o tamanho do módulo *mod* em *bits*, e N sendo o número de iterações, ou ainda, o total de votos realizados

Tempo (ms)	mod	N	c_{min} (*bits*)	c_{max} (*bits*)	$c_{médio}$ (*bits*)	$\prod_{i=0}^{N} c_i$ (*bits*)
0.27800	1024	100	2042	2047	2046	2045
2.35501	1024	1000	2037	2048	2047	2045
24.0347	1024	10000	2034	2047	2046	2046
0.66800	2048	100	4085	4094	4094	4093
6.93298	2048	1000	4084	4095	4094	4093
68.3832	2048	10000	4081	4095	4094	4089
2.05500	4096	100	8180	8191	8190	8191
20.3879	4096	1000	8182	8192	8191	8191
203.967	4096	10000	8179	8192	8191	8191

Os parâmetros referentes ao número de votos e eleitores no teste acima foram ajustados para que fossem os mesmos com os diferentes tamanhos de módulo, pois a mensagem em claro m deve estar dentro do limite como referido na seção 3.5, motivo pelo qual o número de votos possíveis foi fechado em $2^{14} - 1$ e não $2^{28} - 1$ como citado no início do capítulo, pois isso limitaria o número de iterações. Os tamanhos de módulo escolhidos foram com base na utilização atual da criptografia, uma vez que os algoritmos de chave pública utilizavam chaves de 1024 *bits* até pouco tempo atrás, e atualmente se usam chaves públicas de 2048 *bits*. O próximo aumento de tamanho de padrão de chave usado, ao se notar a necessidade de um aumento na segurança, o mesmo será, provavelmente, para 4096 *bits*.

Tabela 3: Avaliação do desempenho da encriptação e decriptação de 10000 votos executados consecutivamente via *script*

Multiplicações (ms)	mod (*bits*)	n_c	|E|	$\sum \mathcal{E}_p$ (ms)	D_p (ms)	$\sum_{i=0}^{N} c_i$ (*bits*)
24.3060	1024	20	14	23381.40	1.5990	2045
24.2488	1024	20	28	24038.30	1.6310	2047
66.4707	2048	20	14	129487.0	11.438	4093
66.1967	2048	20	28	140202.0	11.947	4093
201.048	4096	20	14	709896.0	68.688	8189
200.796	4096	20	28	745985.0	68.450	8187

É possível observar que, na média, o número de *bits* da mensagem encriptada fica próximo ao dobro do número de *bits* do n utilizado. Isso se deve ao fato de que o número de *bits* de um número y em base decimal é $\lfloor \log_2(y) \rfloor + 1$, onde $\lfloor x \rfloor$ representa o valor de x truncado, logo, no caso do algoritmo utilizado, como a mensagem encriptada está em módulo n^2, o número de *bits* é dado $\forall n > 1$ por:

$$\lfloor \log_2(n^2) \rfloor + 1 = \lfloor 2 \times \log_2(n) \rfloor + 1 \tag{8}$$

Pela equação 8 percebe-se que quando $(\log_2(n) - \lfloor \log_2(n^2) \rfloor - 0.5) < 0$, seja $|n|$ o tamanho de n em *bits*, o tamanho em *bits* de n^2 é igual a $2 \times |n| - 1$, e $2 \times |n|$ caso contrário.

Mais genericamente, a equação 8 pode ser representada, para $i > 0$, como:

$$\lfloor \log_2(n^x) \rfloor + 1 = \lfloor x \log_2(n) \rfloor + 1$$

Além disso, o fator probabilístico faz com que o número de *bits* seja próximo do máximo de *bits* possíveis em questão de tamanho, uma vez que a chance dos $b > 0$ primeiros *bits* serem apenas 0 é de 0.5^b, o que significa que o dado encriptado tem aproximadamente o dobro de *bits* da chave. Com isso, é possível notar um grande *overhead* em relação ao tamanho do dado uma vez que, por exemplo, utilizando o *AES* conforme sua definição para encriptar 1 *bit*, o número máximo de *bits* extra é de 127, uma vez que ele trabalha encriptando blocos de 128 *bits*. Em contrapartida, para um dado de 1 *bit* o *overhead* usando uma chave pública de 2048 *bits* pode chegar a ser por volta de 32 vezes maior que o dos dados encriptados com o *AES*, gerando um fluxo de rede bem maior, impactando principalmente no consumo de dados na rede das aplicações.

Em relação a complexidade é possível notar que o mesmo cresce linearmente em relação ao número de iterações, visto que a complexidade do somatório é dada por $n_{it} \times mul_{compl}$, onde n_{it} e mul_{compl} são respectivamente o número de iterações do somatório, ou seja, a quantidade de votos, e a complexidade da multiplicação de dois números de 14 *bits* em módulo especificado conforme a coluna *mod* ao quadrado da tabela 2. Já quanto ao tempo de encriptação, o mesmo tende a convergir para uma complexidade entre $O(n^2)$ e $O(n^3)$, como pode ser visto pelas figuras 5, 6 e 7, o que pode ser explicado devido aos algoritmos de multiplicação conhecidos terem uma complexidade máxima $O(n^2)$, o que no experimento foi o maior responsável pelo tempo total, já que o processo de encriptação foi realizado uma vez por iteração, enquanto a decriptação ocorre apenas uma vez. A multiplicação também ocorre várias vezes durante o processo, mas é possível notar pela tabela 3, em comparação com a tabela 2, que o tempo gasto com a multiplicação de todos os dados encriptados é muito menor do que o tempo gasto com a encriptação dos mesmos, uma vez que o algoritmo de encriptação utiliza 2 multiplicações, 2 divisões e 2 exponenciações modulares, o qual o tempo gasto com o somatório das encriptações $\sum \mathcal{E}_p$ pode ser visualizado na tabela 3. As curvas para estimar a complexidade foram escolhidas levando em conta a complexidade dos piores casos de algoritmos de multiplicação e exponenciação modular, uma vez que não se tinha acesso a complexidade exata dos algoritmos utilizados pela *GMP* para efetuar as operações no experimento, e a estimativa foi realizada levando em consideração que:

$$\lim_{x \to a} \frac{f(x)}{g(x)} = \frac{\lim\limits_{x \to a} f(x)}{\lim\limits_{x \to a} g(x)} \tag{9}$$

Se o resultado da equação 9 é uma constante k, se tem:

$$\frac{\lim\limits_{x \to a} f(x)}{\lim\limits_{x \to a} g(x)} = k \implies \lim_{x \to a} f(x) = k \lim_{x \to a} g(x)$$

Com $x \to \infty$:

$$\lim_{x \to \infty} f(x) \leq k \lim_{x \to \infty} g(x) \implies f(x) \leq k g(x) \tag{10}$$

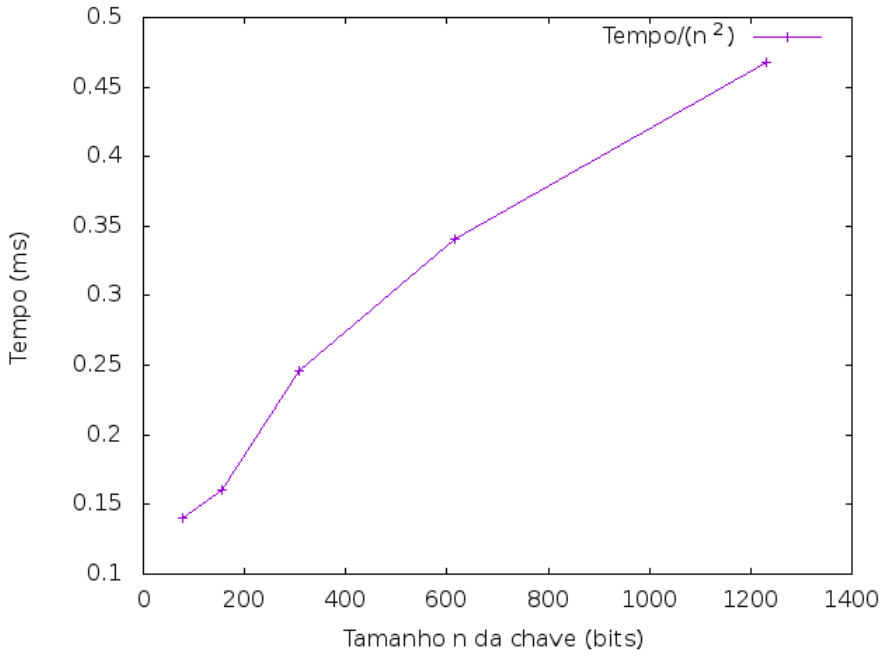
Isto é, $f = O(g)$ quando $\lim\limits_{x \to a} \dfrac{f(x)}{g(x)} = k$, sendo assim, nas figuras 5, 6 e 7, se usou como função f o tempo em função do número de bits n da chave pública, e como g as

funções n^2, n^3 e n^4, respectivamente.

O número de dígitos no módulo de n *bits* foi estimado como sendo, aproximadamente, $\log_{10} n$, uma vez que a complexidade dos algoritmos para as operações referidas leva em conta o número de dígitos dos operandos.

Os gráficos abaixo foram gerados com a utilização do programa *gnuplot*[6], utilizando como entrada os dados crus do tempo de encriptação em função do tamanho do módulo da chave pública.

Figura 5: Gráfico do tempo usado na encriptação dividido por $g(n) = n^2$ em função dos n dígitos da chave, estimado através do número de *bits* da mesma



[6]Fonte: gnuplot. Disponível em <http://www.gnuplot.info>

Figura 6: Gráfico do tempo usado na encriptação dividido por $g(n) = n^3$ em função dos n dígitos da chave, estimado através do número de *bits* da mesma

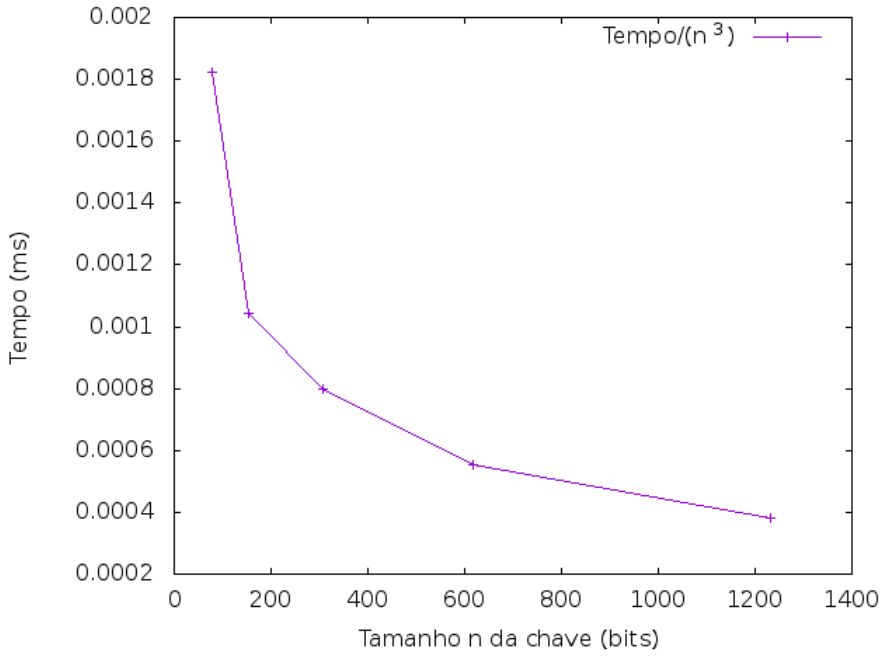
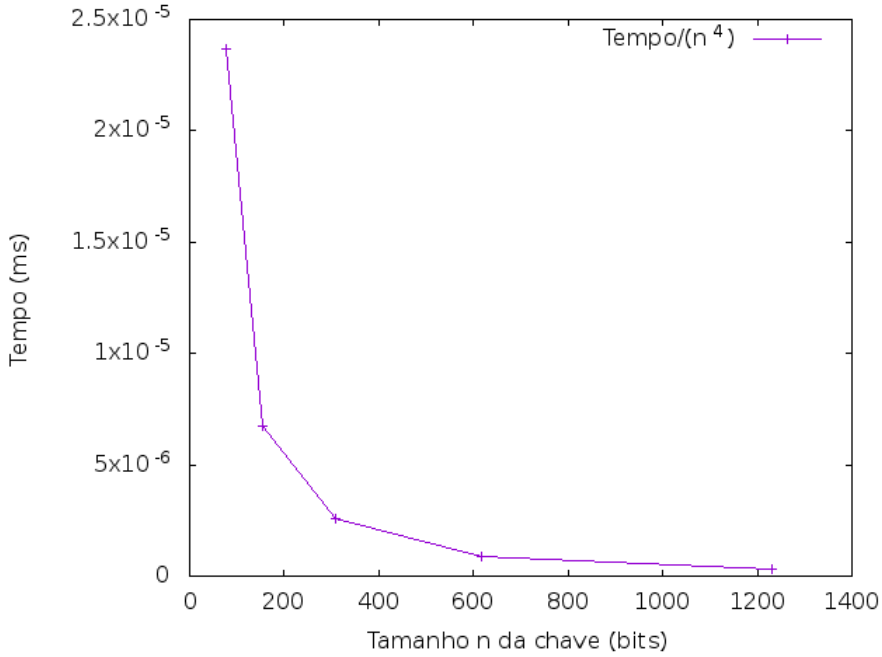


Figura 7: Gráfico do tempo usado na encriptação dividido por $g(n) = n^4$ em função dos n dígitos da chave, estimado através do número de *bits* da mesma



Também é importante ressaltar que nesse caso específico de aplicação, apesar do tempo alto para se obter os resultados, cada operação de votação já poderia realizar a multiplicação com os demais votos armazenados, não necessitando armazenar todos os

votos e realizar o produto de uma vez só, se beneficiando assim do tempo gasto com uma interação com o usuário. Como nota-se pela tabela 2, em função da quantidade de operações com os dados encriptados, o tempo cresce linearmente, tornando-se um problema para aplicações que realizam operações constantemente durante a sua execução, como por exemplo, processamento de vídeos em tempo real com os dados encriptados, ou comunicação utilizando *Voice over Internet Protocol (VoIP)*.

Algumas otimizações podem ser realizadas para transformar o protótipo em uma aplicação em tempo real, como por exemplo, criar um sistema distribuído, onde cada central de votação computaria seu somatório de votos encriptados utilizando uma única chave pública pertencente a um sistema central que é o único a conhecer a chave privada correspondente, assim os sistemas intermediários poderiam realizar a computação sem nunca ter conhecimento sobre o valor da sua parcela do resultado da computação, o que é um ótimo exemplo da segurança proporcionada pela criptografia homomórfica em sistemas distribuídos, uma vez que nenhum dos pontos intermediários tem acesso à informação da votação, a preocupação com o a possível espionagem da informação que passa pelos servidores hospedando os sistemas intermediários de votação diminui em termos de requisitos de confidencialidade.

6 CONCLUSÃO

Foram vistos nesse trabalho diferentes algoritmos de encriptação, e no contexto da criptografia de chave pública, foi possível observar e provar a existência de homomorfismos entre o espaço de uma mensagem de texto claro no domínio de cada sistema de encriptação apresentado para o espaço de texto cifrado do mesmo, bem como foi apresentada uma prova de conceito através de uma implementação de um protótipo de aplicação baseado em um sistema de votação eletrônica utilizando o algoritmo de chave pública de *Paillier*, demonstrando a possibilidade de implementação de criptografia homomórfica baseando-se no desempenho do protótipo apresentado.

Concluiu-se que, apesar da complexidade apresentada pela encriptação e decriptação dos dados, determinadas aplicações, como a aplicação apresentada, não são impactadas de significativamente. No caso específico de votação, a ação de voto e a soma do voto com o acumulador de votos totais podem ser computados em tempo de interação de um usuário com o sistema, fornecendo uma resposta em menos de 1 segundo, o que apesar de não ser um tempo curto em termos de computação, é rápido para a percepção humana, além do fato de a decriptação ocorrer apenas uma única vez nesse tipo de sistema, já que todos os dados são encriptados e assim processados, sendo necessária a decriptação apenas para a divulgação do resultado final, o que também garante que toda a execução se dá sem a necessidade da exposição dos dados, garantindo confidencialidade.

Como demonstrado através de execuções de um *script*, conforme descrito na seção 5.3.7, o problema da complexidade é visível em sistemas onde a computação acontece dado um grande número de operações simultâneas, com o tempo crescendo em uma complexidade entre $O(n^2)$ e $O(n^3)$ em relação ao tamanho da chave escolhido, o que afetaria, por exemplo, operações sobre vídeos transmitidos em tempo real. Além disso, por não utilizar a técnica padrão descrita na seção 2.4.1, há um custo extra muito grande no fluxo de dados na rede usada para a comunicação, devido a diferença de tamanho padrão de chave entre os utilizados por algoritmos de chave pública e os utilizados por algoritmos de chave simétrica, pois qualquer dado encriptado no sistema de encriptação aplicado, de *Paillier*, tende a um tamanho próximo do quadrado do módulo escolhido como parâmetro de chave pública que é muito maior do que o tamanho típico de uma chave de um algoritmo de criptografia simétrica.

Ainda, é possível estabelecer algumas otimizações no protótipo e sistemas baseados nele, como por exemplo, aproveitar a confidencialidade dos dados e dividir um sistema em

vários sistemas para trabalhar com computação distribuída, assim obtendo multiprocessamento a fim de diminuir o custo em termos de tempo. No caso de um sistema de votação, ainda é possível, dada uma eleição qualquer, utilizar dados estatísticos sobre intenções de votos para economizar *bits* na estrutura proposta na seção 5.3.6.

Trabalhos futuros podem se basear no protótipo apresentado e criar um sistema de votação robusto, com preocupações de segurança que vão além da confidencialidade, tal como utilizar assinaturas digitais com a finalidade de aumentar a segurança no quesito de autenticidade, ou ainda, explorar outros tipos de aplicação usando outros sistemas de encriptação, inclusive sistemas completamente homomórficos, visto que o foco deste trabalho foi o estudo e implementação de sistemas de encriptação parcialmente homomórficos. Também há espaço para uma análise de complexidade mais detalhada do processo de encriptação e decriptação de determinados algoritmos, bem como uma comparação de desempenho detalhada entre diferentes sistemas de encriptação.

REFERÊNCIAS

ADLEMAN, R.; ADLEMAN, L.; DERTOUZOS, M. On Data Banks and Privacy Homomorphisms. In: **Foundations of Secure Computation**. [S.l.]: New York: Academic Press, 1978. p. 169–180.

DAEMEN, J.; RIJMEN, V. **The Design of Rijndael**. [S.l.]: Springer, 2002.

DIFFIE, W.; HELLMAN, M. New Directions in Cryptography. In: . [S.l.]: IEEE, 1976. v. 22, p. 644–654.

ELGAMAL, T. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In: **Proceedings of CRYPTO 84 on Advances in cryptology**. [S.l.]: ACM, 1985. p. 10–18.

GENTRY, C. **A Fully Homomorphic Encryption Scheme**. Thesis (PhD) — Stanford University, 2009. <crypto.stanford.edu/craig>.

MERKLE, R. Secure Communications Over Insecure Channels. **Communications of the ACM**, ACM, v. 21, p. 294–299, 1978.

NIST. **Announcing the Advanced Encryption Standard (AES)**. [S.l.], 2001. Available from Internet: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>.

NIST. **Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems**. [S.l.], 2016. Available from Internet: <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160.pdf>.

PAILLIER, P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: **EUROCRYPT'99 Proceedings of the 17th international conference on Theory and application of cryptographic techniques**. [S.l.]: Springer, 1999. p. 223–238.

PARKER, D. B. Our Excessively Simplistic Information Security Model and How to Fix It. **ISSA Journal**, v. 8, p. 12–21, 2010.

POPA, R. A. et al. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In: ACM. **Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)**. [S.l.], 2011.

PRAVEEN, S. Homomorphic Elgamal Encryption in NoSQL for Secure Cloud. **International Journal of Innovative Research in Computer and Communication Engineering**, v. 4, 2016.

RABIN, M. O. Probabilistic Algorithm for Testing Primality. **Journal of Number Theory**, Elsevier, v. 12, p. 128–138, 1980.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. **Communications of the ACM**, ACM, v. 21, p. 120–126, 1978.

SALTZER, J.; SCHROEDER, M. The Protection of Information in Computer Systems. In: **Proceedings of the IEEE**. [S.l.]: IEEE, 1975. v. 63, p. 1278–1308.

STALLINGS, W. **Cryptography and Network Security: Principles and Practice**. 5th edition. ed. [S.l.]: Pearson, 2011.