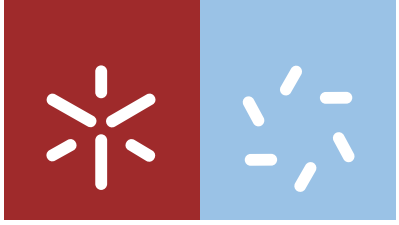
Universidade do Minho
Escola de Ciências

Nunes Tchimúa Mucuata Rafael

Uma abordagem criptográfica a sistemas eleitorais

julho de 2019

Nunes Tchimúa Mucuata Rafael **Uma abordagem criptográfica a sistemas eleitorais**
UMinho|2019



Universidade do Minho
Escola de Ciências

Nunes Tchimúa Mucuata Rafael

Uma abordagem criptográfica a sistemas eleitorais

Dissertação de Mestrado
Mestrado em Matemática e Computação

Trabalho realizado sob orientação do
Doutor José Pedro Miranda Mourão Patrício
e do
Doutor José Carlos Bacelar Almeida

julho de 2019

Direitos de autor e condições de utilização do trabalho por terceiros

DEDICATÓRIA

Especialmente a Deus e
a minha família. Dedico
também a todos que têm torcido
a meu favor!

AGRADECIMENTOS

Agradeço ao nosso criador Deus Todo-Poderoso, pela vida e força que tem me dado todos os dias para seguir em frente, pois sem Ele este trabalho jamais se concretizaria. Agradeço em particular, ao meu Pastor Luís Kamuele pelo encorajamento e interseções, ao meu diretor de trabalho João Domingos Cadete por me fazer acreditar que seria possível. Agradeço também aos meus pais e a toda minha família, aos irmãos em Cristo em Angola e cá em Portugal, pelas interseções e ajuda cedida. Como não podia deixar de ser, agradeço ao coletivo de professores do curso de Matemática e Computação da Universidade do Minho pela qualidade de ensino proporcionada, pois este ganho é fruto das experiências adquiridas e valores que de certeza marcarão toda minha vida.

Nunes Tchimúa Mucuata Rafael

Universidade do Minho, 15 de Julho de 2019.

Declaração de integridade

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

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

RESUMO

Uma abordagem criptográfica a sistemas eleitorais

O presente trabalho, intitulado "Uma Abordagem Criptográfica a Sistemas Eleitorais", tem como objetivo elaborar um esquema criptográfico que visa garantir a confidencialidade, integridade, autenticidade, verificabilidade do voto e o anonimato do eleitor numa eleição eletrónica. A criptografia desempenha um papel fundamental na garantia da segurança do voto eletrónico e do eleitor. De modo a ter uma perceção mais apurada acerca do assunto, inicialmente foi feita uma revisão de bibliografia existente, tanto de autores nacionais e internacionais, onde foi possível colher teorias que impulsionaram a fundamentação da aplicação abordada neste trabalho. Começou-se por apresentar as noções da criptografia, sua aplicação desde os tempos remotos e seu desenvolvimento até aos dias atuais. O RSA foi abordado visando sua aplicação na assinatura digital cega do voto. Para a cifração e decifração dos votos, foi proposto o Elgamal devido às suas propriedades homomórficas. Em seguida, o trabalho narra a eleição eletrónica com intuito de dar uma noção acerca do seu funcionamento no que diz respeito aos diferentes estilos de votos, bem como a necessidades de se assegurar o processo. De modo a garantir simultaneamente o anonimato do eleitor e a autenticidade do voto, propôs-se a técnica de assinatura digital cega, onde o eleitor solicita uma terceira entidade para realizar a assinatura, sem, no entanto, ter acesso ao conteúdo do voto, pois, antes de ser enviado para o assinante, a mensagem (voto) é ofuscada (cegado). A chave de decifração dos votos deve ser mantida em segredo devido à sua importância e sensibilidade para o êxito do processo. O grande problema reside no armazenamento da chave privada de modo a que agentes não autorizados não tenham acesso à mesma. Aplicou-se o Esquema de Partilha de chave secreta de Shamir como uma forma de armazenar a chave secreta de forma segura. Tratou-se da prova de conhecimento zero como uma forma de se provar para alguém que queira verificar se os votos foram devidamente formados, sem que ele (verificador) tenha acesso ao conteúdo de cada voto. Para tal, abordou-se o esquema de Feige-Fiat-Shamir. E por fim, construiu-se um protocolo criptográfico para eleição eletrónica que narra de forma simplificada as etapas que visam a garantia da segurança de um voto eletrónico num processo eleitoral. Chegamos a algumas conclusões e recomendações para futuras abordagens relacionadas ao assunto.

Palavras-chave: Criptografia, eleição eletrónica, técnicas criptográficas,voto eletrónico, segurança de dados.

ABSTRACT

A Cryptographic Approach to Electoral Systems

The present work, entitled "A Cryptographic Approach to Electoral Systems", aims to elaborate a cryptographic scheme that aims to guarantee the confidentiality, integrity, authenticity, verifiability of the vote and the anonymity of the voter in an electronic election. Cryptography plays a key role in ensuring the safety of electronic voting and voters. In order to get a better understanding of the subject, a review of existing bibliographies was made, both by national and international authors, where it was possible to gather theories that gave support to the application of this work. We begin by presenting the notions of cryptography, its application from the earliest times, its development to the present day. The RSA was approached for its application in the blind digital signature of the vote. For the encryption and decryption of the votes, the Elgamal was proposed because of its homomorphic properties. This dissertation then focuses the electronic election in order to give a sense of how it works in terms of different voting styles, as well as the need to secure the process. In order to guarantee both the anonymity of the voter and the authenticity of the vote, a blind digital signature technique was proposed, where the voter requests a third entity to execute the signature, without, however, having access to the content of the vote, since , before being sent to the subscriber, the message (vote) is blinded. The decryption key must be kept secret because of its importance and sensitivity to the success of the process. The big problem lies in storing the secret key so that unauthorized agents have not access to it. The Shamir Secret Sharing Scheme was applied as a way to store the secret key securely. We used Zero- Knowledge Proofs in order to prove to someone who wants to verify that the votes were properly formed, without the verifier having access to the content of each vote. For this, the Feige-Fiat-Shamir scheme was approached. Finally, a cryptographic protocol for electronic voting vote was elaborated that describes in a simplified way the steps that aim at guaranteeing the security of an electronic vote in an electoral process. We have come to some conclusions and recommendations for future approaches related to the subject.

Keywords: Cryptography,cryptographic techniques, data security, electronic election, electronic vote.

Conteúdo

Capítulo 1

Introdução

É notório atualmente o crescimento exponencial dos serviços oferecidos pela internet, o que tem provocado, também, o aumento de número de crimes cibernéticos. A necessidade de se aumentar a segurança da informação que circula na internet tem permitido a criptografia firmar-se no ramo da Computação, e cuja aplicação é indispensável para a garantia da segurança das transações que circulam através da internet.

Numa eleição tradicional, por vezes para garantir a segurança do voto usam-se técnicas que exigem o envolvimento de forças armadas para impedir que agentes maliciosos interfiram no processo, pois o perigo está sempre a espreita. A pergunta que não se quer calar é, como garantir essa segurança numa eleição eletrónica, onde o voto circula de forma digital e não física? Esta foi uma das principais razões que me impulsionaram a levar a cabo esta pesquisa.

Pretendemos, neste trabalho, abordar o funcionamento das técnicas criptográficas aplicadas num sistema de eleição eletrónica.

1.1 Objetivos

a) Objetivo geral:

A presente dissertação tem como objetivo geral:

Elaborar um esquema criptográfico que visa garantir a confidencialidade, integridade, autenticidade, verificabilidade do voto e o anonimato do eleitor numa eleição eletrónica.

b) Objetivos específicos:

Para o alcance deste objetivo (geral), foram traçados os seguintes objetivos específicos:

- Revisar a bibliografia com intuito de identificar as abordagens relacionadas ao

tema em causa e analisar as conclusões de outros autores.

- Introduzir a noção de criptografia para reunir conceitos básicos necessários para a redação, análise e interpretação do conteúdo inerente aos capítulos subsequentes.
- Descrever o criptossistema RSA, pelo fato de ser útil no processo da construção do algoritmo da assinatura cega.
- Identificar os intervenientes do processo de eleição eletrónica, explicitando suas funções.
- Descrever os requisitos de segurança a ter em conta numa eleição eletrónica.
- Descrever o criptossistema Elgamal, pela utilidade das suas propriedades homomórficas.
- Descrever o esquema de assinatura cega, com intuito de garantir autenticidade e anonimato simultaneamente.
- Descrever o esquema de partilha da chave secreta de Shamir, com intuito de efetuar a partilha da chave utilizada para decifração dos votos.
- Descrever o esquema de Feldman para a verificação das partes secretas distribuídas.
- Descrever o esquema de identificação de Feige-Fiat-Shamir, com intuito de se provar à autoridade responsável pelo escrutínio que os votos foram corretamente formados.

Portanto, o trabalho restringe-se apenas em abordar a aplicação da criptografia ao sistema eleitoral eletrónico, pois as técnicas propostas são aplicadas em transações cujo canal de comunicação é inseguro.

Em complemento, optámos por efetuar uma curta introdução histórica da criptografia, com intuito de analisar a sua evolução histórica desde a antiguidade até a história mais recente.

1.2 Estrutura do trabalho

Esta dissertação está subdividida da seguinte forma:

No primeiro capítulo: Introdução à criptografia, trata de aspetos básicos acerca da criptografia, que vão desde a etimologia do termo, sua definição, história e alguns conceitos decorrentes.

No segundo capítulo: Resultados iniciais, faz-se uma curta abordagem à teoria de grupos e teoria de números, nomeadamente, função de Euler, os criptossistemas RSA e Elgamal.

No terceiro capítulo: Eleição eletrónica, numa perspetiva de perceber o processo eleitoral eletrónico, trata-se da eleição eletrónica como uma necessidade para melhoria e celeridade do processo, tendo apresentado os intervenientes do processo, e os requisitos de segurança a ter em conta, pois a não observância dos mesmos pode ser catastrófico para a obtenção de resultados fidedignos.

No quarto capítulo: Abordagem criptográfica a eleição eletrónica. No inicio o capítulo aborda da assinatura cega, uma assinatura digital que permite garantir a autenticidade e preservar o anonimato do eleitor. Isto é, a assinatura cega permite que numa eleição eletrónica o voto seja autêntico, e ao mesmo tempo a identidade do eleitor não seja revelada. Em seguida, tratou-se de um esquema verificável de partilha da chave secreta, esquema proposto como uma das formas de armazenar a chave secreta (usada na decifração dos votos), onde um conjunto de elementos recebem partes da chave, e para a sua recuperação é necessário a junção de um número de partes partilhadas. No entanto, vimos que cada participante verifique se a parte que recebeu é consistente, pois erros ou intenções maliciosas podem ocorrer durante a partilha. Para tal, apresentamos o esquema de partilha de Shamir cuja verificação é feita mediante o esquema de verificação de Feldman. Ainda no mesmo capítulo, falou-se da prova de conhecimento zero (nulo), onde duas entidades, provador e verificador para uma (provadora) provar a outra (verificador) que os votos estão formados corretamente. Verificação esta que é baseada no esquema de identificação de Feige-Fiat-Shamir. E por fim, construiu-se um esquema criptográfico para eleição eletrónica, que é a proposta apresentada no trabalho.

No quinto capítulo foi possível tirar a conclusões que, de certa forma, augura-se que sejam também valiosas e proveitosas para o leitor. Deixam-se recomendações para quem deseja aplicar este tema em estudos futuros.

Temos os anexos, onde se encontram implementações das técnicas criptográficas feitas no Sagemath cuja linguagem de programação é o Python 2.0. Os códigos construídos relativamente ao RSA, Elgamal, Assinatura cega, Prova de conhecimento zero, Esquema de partilha da chave secreta e o esquema criptográfico para eleição eletrónica estão disponibilizados na plataforma Github, com acesso em: *https://github.com/Agente2019/Implementation-E-voting-Cripto.*

1.3 Introdução à Criptografia

Enviar mensagens de forma segura é um desafio muito antigo. O homem sentiu, desde muito cedo, a necessidade de guardar informações em segredo. A criptografia nasceu com a diplomacia e com as necessidades militares. Generais, reis e rainhas, procuravam formas eficientes de comunicação para comandar seus exércitos e governar seus países. A importância de não revelar segredos e estratégias às forças inimigas, motivou o desenvolvimento de técnicas para mascarar uma mensagem, possibilitando apenas ao destinatário ler o conteúdo. As nações passaram a criar departamentos para elaborar sistemas criptográficos. `Ao longo da história, os códigos decidiram o resultado de batalhas` (Martins, 2005).

Portanto, o processo de cifração de mensagens tem um papel cada vez maior na sociedade, e tendo em vista a necessidade de se criar ferramentas capazes de proteger a informação e de prover segurança aos documentos armazenados e transmitidos pelas organizações, tem-se, desta forma, a motivação para o estudo da criptografia.

Etimologicamente, a palavra criptografia deriva do grego: cripto (kriptos)- que significa esconder e grafia (graphein)- que significa escrever. A criptografia tem a finalidade de ocultar uma informação de pessoas não autorizadas, garantindo privacidade, isto é, seu estudo consiste em garantir fundamentalmente a integridade, confidencialidade, autenticidade e anonimato e não repúdio. Portanto, *a criptografia tem o objetivo de escrever uma mensagem de modo que ninguém, exceto o emissor e recetor consiga ler* (Martins, 2005).

A par da criptografia, existe a criptoanálise, ciência que atua contrariamente à anterior. Visto que seu estudo está intrinsecamente direcionado na quebra e comprometimento dos requisitos de segurança da criptografia, a criptoanálise tem a finalidade de quebrar cifras.

De seguida abordamos de forma sinténtica alguns aspectos da criptografia desde os primórdios até a atualidade.

1.3.1 Primórdios da Criptografia

Kahn (1973) citado por Martins (2005, p. 15), relata que os primeiros sinais da criptografia teve início em 2000 a.C, no Egipto e na Mesopotâmia. Os egípcios usaram expedientes criptográficos ao utilizar escrita denominada hieroglífos usada pela nobreza (classe social).

A civilização Indiana também desenvolveu diferentes formas de comunicação secreta, utilizada pelo serviço de espionagem existente na altura.

Na civilização da Mesopotâmia foram encontradas algumas referências cripto-

gráficas em tabelas cuneiformes. A mais antiga data cerca de 1500 a. C.

Na China, e em paralelo com a criptografia, desenvolveu-se a Esteganografia (comunicação secreta que se obtém escondendo a existência das mensagens).

1.3.2 Idade média

O período que vai de 500 a 1400 d. C, os árabes-islâmicos contribuíram no estudo da escrita cifrada, iniciando a criptologia (escrita secreta) com a invenção da criptoanálise para substituição monoalfabética. Em 1379, o papa Clemente VII decide unificar o sistema de cifras da Itália. Entre os sec. XIV e XV, a criptologia generalizou-se na Europa e tornou-se uma atividade prestigiante. Alberti inventou e publicou a primeira cifra polialfabética e criou um disco para cifrar (Martins, 2005).

1.3.3 Criptografia moderna

A criptografia moderna originou-se entre os árabes, os primeiros a documentar sistematicamente os métodos criptoanalíticos. No século XIX, a criptografia foi alcançando um notório avanço mediante as técnicas criptoanalíticas usadas nas missões de guerrilhas. Edgar Allan (1809- 1849) propôs métodos sistemáticos para resolver cifras na década de 1840. Durante a primeira e segunda guerras mundiais, a criptoanálise era aplicada na quebra de códigos durante os combates. Em 1917, Gilbert Vernam propôs uma cifra de teleimpressão, onde uma chave é combinada, caractere por caractere, com o texto puro para produzir o texto cifrado, fato que levou ao desenvolvimento de dispositivos eletromecânicos como máquinas cifradoras. Depois de 1940, a cooperação de criptógrafos britânicos e holandeses permitiu a invasão de vários sistemas de criptografia da Marinha Japonesa (Jackob, 2001).

Com o aparecimento do telégrafo, de máquinas e dispositivos mais elaborados, entra-se na era da criptografia mecanizada. Surgem também sistemas de comunicação à distância, e estes por tornarem as mensagens mais desprotegidas, impulsionaram o desenvolvimento da criptografia. Thomas Jefferson (1743-1826) inventou um sistema de cifras denominadas por wheel cipher. Era um cilindro de madeira que permitia realizar com rapidez e segurança uma substituição polialfabética. Esta descoberta conferiu a Jefferson o título de pai da criptografia americana. Por volta do sec XX, Guglielmo Marconi inicia a era da comunicação sem fios com a invenção da rádio, o que ampliou ainda mais a necessidade de cifrar as mensagens de forma eficaz. Em 1918, o inventor alemão Arthur Scherbius(1878-1929) construiu uma máquina criptográfica elétrica, designada por Enigma. Esta invenção forneceu ao exército alemão o sistema criptográfico mais seguro do mundo e permitiu a prote-

ção das comunicações do exército alemão durante a 2ª guerra mundial. Por volta de 1943, os criptoanalistas britânicos inventaram um computador programável para quebrar uma cifra mais potente que o Enigma, a cifra Lorenz. Em 1976, com a invenção do DES (Data Encrypt Standard), ficou resolvido o problema da estandarização e aumentou a utilização da criptografia como medida de segurança por parte de empresas. Nesta altura, um novo problema tornava-se necessário resolver, conhecido por distribuição de chave. A única maneira segura de transmitir a chave de uma cifra era enviá-la por mão própria, e a menos segura era a transmissão por correio.

Martin Hellman (1945-) pertencente ao Departamento de Engenharia Eletrotécnica da Universidade Stanford, e Whitfield Diffie (1944-) estudante de pós-graduação na altura, associaram-se no estudo com intuito de encontrar uma solução para a distribuição de chave. Algum tempo depois, juntou-se a eles Ralph Merkle (1952-). Diffie, Hellman e Merkle inventaram um esquema de troca de chave, que introduziu a ideia da criptografia assimétrica. Este sistema era pouco prático. Diffie continuou o estudo, e em 1975 publicou o esboço de um novo tipo de cifra que incorpora a criptografia de chave pública. Até então, todas as técnicas eram simétricas. Em Abril de 1977 é criado o RSA por Rivest (1948-), Shamir (1952-) e Adlman (1945-) (Kahn, 1973) citado por Martins (2005).

1.3.4 Criptografia simétrica

A criptografia simétrica é uma área da criptografia que utiliza uma chave privada no processo de cifração/decifração da mensagem. Sendo privada, a chave deve ser partilhada entre o emissor e recetor de forma antecipada, e este processo (de partilha) exige a utilização de canais de comunicação seguros. Se um intruso (atacante) intercetar a chave partilhada, a política de privacidade da comunicação entre ambos fica comprometida. Daí que é necessário a geração e utilização de uma chave criptograficamente segura. Isto é, a segurança criptográfica não depende intrinsecamente da cifra a ser utilizada. Mas sim, a segurança está mais interligada com a segurança da chave (Guimarães, 2001).

O algoritmo de uma cifra simétrica tem o seguinte modelo criptográfico (Krishna, 2004):

(a) Texto claro: essa é a mensagem ou dados originais, que servem como entrada do algoritmo de cifração.

(b) Algoritmo de cifração: realiza diversas substituições e transformações no texto claro (a cifrar).

(c) Chave secreta: é um parâmetro de entrada para o algoritmo de cifração. A chave

é um valor independente do texto claro e do algoritmo, usado para cifrar o texto claro. O algoritmo transformará um parâmetro de saída diferente do texto claro.
(d) Texto cifrado: essa é a mensagem transformada (impercetível), produzida como saída do algoritmo de cifração. Ela depende do texto claro e da chave secreta. O texto cifrado é um conjunto de dados aparentemente aleatório (pseudoaleatório) e, para determinada mensagem, duas chaves diferentes produzirão dois textos cifrados distintos.
(e) Algoritmo de decifração: esse é basicamente o algoritmo de cifração executado de modo inverso. Este algoritmo permite recuperar o texto claro (legível).

1.3.5 Criptografia assimétrica

A criptografia assimétrica (de chave pública) foi inventada por Whitfield Diffie e Martin Hellman em 1976. São utilizadas duas chaves, uma privada e outra pública. Somente a chave pública é acessível a todos os intervenientes na comunicação. Não é feita a partilha de chaves, pois cada agente detém de um par de chaves, uma para cifrar e outra para decifrar.

A criptografia assimétrica vem resolver o problema da distribuição da chave (simétrica) secreta. Cada agente gera um par de chaves de forma independente, e a técnica de armazenamento da chave privada é concebida e conhecida somente pelo titular. É comum ver a combinação das cifras, simétrica e assimétrica, isto é, podemos usar a criptografia de chave pública para cifrar a chave de uma cifra simétrica, mantendo confidencial a referida chave (Nascimento, 2011).

Os sistemas criptográficos de chave assimétrica baseiam-se na dificuldade que existe em se calcular a operação inversa de determinadas operações matemáticas. Existem problemas matemáticos a partir dos quais são construídos os algoritmos de chave pública:

a) Factorização de inteiros

Dado um número n resultado da multiplicação de dois números primos p e q, a dificuldade consiste em encontrar p e q tendo-se somente n. Para números de tamanhos pequenos, um ataque de força bruta pode facilmente encontrar a solução, mas para valores de n de grande tamanho (da ordem de 3000 bits ou mais), a factorização de n seria pouco eficiente.A segurança do RSA (Rivest, Shamir e Adleman) baseia-se neste problema (Masthanamma e Pleya, 2015).

b) Logaritmo discreto (DLP-Discret Logarithm Problem)

Seja a equação

$$y \equiv g^x \mod p, \tag{1.1}$$

onde g é um número inteiro positivo e p um número primo, ambos conhecidos. A dificuldade consiste em, dado o valor de y calcular o valor de x. Não se conhece um algoritmo eficiente que resolva este problema. Isto é, existe uma intratabilidade computacional do logaritmo discreto, e a segurança dos esquemas criptográficos Elgamal e Diffie Hellman baseiam-se neste problema.

1.3.6 Assinatura digital

A assinatura digital é um processo de assinatura eletrónica baseado em um sistema criptográfico assimétrico composto de um algoritmo, mediante o qual é gerado um par de chaves, uma das quais privada e outra pública. Essa técnica permite ao titular usar a chave privada para declarar a autoria da mensagem. O destinatário usa a chave pública do assinante para verificar se a assinatura foi criada mediante o uso da correspondente chave privada e se a mensagem foi alterada depois da assinatura (Júnior, 2011).

Entende-se que uma assinatura digital pode permitir um aumento de segurança em transações consideradas inseguras. Isto é, pode evitar que um elemento não autorizado possa fazer-se passar por outro (autorizado) numa troca de informação. Com a chave pública do signatário, o destinatário pode verificar se houve modificação de alguns ou de todos os bits da mensagem. A assinatura digital visa garantir:

(i) Autenticação da identidade da entidade que assinou a mensagem; **(ii)** Integridade (não alteração acidental ou maliciosa) da mensagem durante a sua transmissão;

(iii) Não repúdio, isto é, o emissor não pode reclamar que não foi ele que assinou a mensagem.

Função Hash: dada uma mensagem m, uma função hash representada por $h(m) = y$ é a menor representação de m, isto é, a assinatura é feita no hash, e não na mensagem. visto que assinar uma mensagem completa, e se essa mensagem for de grande tamanho, para além de se gastar muito tempo no processamento do algoritmo, a assinatura digital resultante consumiria muito espaço em memória (Pass e Shelat, 2010).

Uma função hash deve ser criada de modo que seja resistente à colisões, isto é, dadas duas mensagens m e $m^{'}$, é impossível calcular $h(m)$ e $h(m^{'})$ funções de hash de m e $m^{'}$, respetivamente, tais que $h(m) = h(m^{'})$. Todavia, em vez de assinarmos a mensagem m, assinamos o hash de m.

Porém, se um adversário intersetasse a mensagem, ele não deveria ser capaz de calcular uma outra mensagem $m^{'}$ que tenha o mesmo hash de m. Logo o adversário não será capaz de calcular qualquer mensagem diferente de m que tenha a mesma assinatura de m. Se com a chave pública do signatário não for possível por parte

do destinatário verificar a autenticidade da mensagem recebida, isto significaria que a chave secreta usada para assinar não foi o par dessa chave pública, também pode significar que a mensagem foi alvo de um ataque. Logo, o recetor pode concluir que a origem da mensagem não é fidedigna.

Durante o mesmo processo de verificação, calcula-se o hash da mensagem recebida, e operando a chave pública na assinatura, este valor é comparado com o hash recebido. Se os dois valores de hash forem diferentes, a mensagem recebida será rejeitada pelo destinatário. E se forem iguais, isto significa que a integridade da mensagem foi preservada.

Este processo de autenticação está sujeito de um ataque designado por man-in-the-middle, onde o recetor é levado a fazer a verificação da autenticidade com uma chave pública errada.

Man-in-the-middle é um ataque em que um invasor que escuta e acompanha toda comunicação que circula num canal mantida entre dois agentes, interceta a mensagem e troca a chave pública do emissor. Em seguida envia a sua chave pública para o recetor, cria e transmite uma nova mensagem assinada com a sua chave privada, fazendo acreditar às vítimas que estão se comunicando diretamente uma da outra de forma privada, enquanto que o intruso controla todo sistema. Desta forma, o recetor verifica a autenticidade da mensagem com uma chave pública errada (Erickson, s.d).

Portanto, uma vez que no processo de assinatura digital se inclui a chave pública, um intruso pode enganar o verificador da assinatura. Logo é necessário incluir certificado digital para garantir a associação entre chave pública e identidade do emissor, isto é, para garantir que a chave pública provém de um agente esperado.

1.3.7 Segurança criptográfica

Durante o processo de geração de uma chave, deve-se gerar uma sequência pseudoaleatória de bits de elevado tamanho. A sequência de bits deve ser imprevisível, isto é, dado um segmento inicial não deve ser possível prever a sua continuação. A repetição de chaves, a previsibilidade e as chaves de tamanhos pequenos são sensíveis de serem quebradas por um atacante ativo.

O estudo e aplicação da criptografia tem por objetivo assegurar uma ou mais das seguintes propriedades de segurança (Anjos, 2016):

a) Confidencialidade: garante que uma informação seja manipulada somente por usuários devidamente autorizados. A confidencialidade garante que mais ninguém teve acesso ao conteúdo da mensagem. Portanto, a confidencialidade, refere-se à impossibilidade de um adversário descobrir uma quantidade não desprezível de in-

formação acerca da mensagem transmitida.

b) Integridade: garante que a informação processada ou transmitida chegue ao seu destino exatamente da mesma forma em que partiu da origem. Para se garantir a integridade, propriedade relacionada à precisão das informações, atestando a sua validade de acordo com os padrões e expectativas estabelecidas previamente. O destinatário da mensagem deverá possuir ferramentas para avaliar se a mensagem foi alterada ou não durante seu processo de transmissão, ou seja, atestar se o que chegou ao destino é idêntico ao que foi enviado na origem (não foi modificado).

c) Autenticidade e não repúdio: a autenticidade garante ao destinatário que a mensagem recebida foi realmente enviada pelo emissor (previsto). Na autenticação deve-se provar a real identidade do emissor, isto é, deve-se ter a certeza absoluta de que uma informação provém das fontes anunciadas, Ou seja, que o emissor da mensagem não é falso e que a mesma não foi modificada ao longo do processo de envio e receção. Para tal, deve-se autenticar o remetente. Assim, o seu destinatário consegue de maneira segura identificar e verificar que foi o mesmo (emissor esperado) quem enviou a mensagem, e ninguém mais. Portanto, o recetor da mensagem pode testar se uma mensagem foi intercetada e modificada, e neste caso, conclui se vai rejeitá-la, ou não.

O não-repúdio impossibilita que o remetente de uma mensagem negue a autoria da assinatura. Em alguns ataques é possível que um intruso realize um ataque de personificação, fazendo-se passar, por exemplo, pelo emissor ou por outro agente. Pode-se impedir este fato, garantindo autenticidade da mensagem.

Portanto, é possível garantir-se o não repúdio por meio de ferramentas como certificados e assinatura digital, onde o emissor legalmente e tecnicamente não pode negar a autoria de uma mensagem assinada com uso de seu certificado, pois se garante que ele e somente ele fez a transmissão.

d) Anonimato: é a garantia de que nenhum agente é capaz associar a identidade do emissor de uma mensagem à respetiva mensagem. Isto é, não deve ser possível, por exemplo, associar a identidade de um eleitor com a intenção do seu voto, isto é, não pode ser possível identificar o autor do voto exercido à favor de um determinado partido ou candidato concorrente. Portanto, o anonimato é um dos requisitos de segurança de uma eleição eletrónica, pois em qualquer ato eleitoral os eleitores devem ser mantidos anónimos.

Capítulo 2

Resultados iniciais

Neste capítulo aborda-se de forma simplificada os criptossistemas RSA e Elgamal, com intuito de permitir uma aplicação mais significativa das mesmas nos capítulos subsequentes. Mas antes, fez-se um tratamento da função de Euler por ser crucial na construção do algoritmo do RSA, e da noção de grupo, pois, exerce também um grande papel na análise, interpretação e construção dos algoritmos das técnicas criptográficas.

2.1 Noção de grupo

Definição 1: Seja o conjunto $G \neq \emptyset$, com operação binária $(\cdot)$,
$(a, b) \in G \times G \mapsto a \cdot b \in G$. Diz-se que $(G, \cdot)$ é um grupo se cumpre com as seguintes propriedades:

- $\forall\ x, y \in G : x \cdot y \in G$
- $\forall\ x, \exists 1 \in G : 1 \cdot x = x \cdot 1 = x$
- $\forall\ x, y, z \in G$, temos $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
- $\forall\ x \in G, \exists\ x^{-1} \in G : x \cdot x^{-1} = x^{-1} \cdot x = 1$.

Para simplificar a notação, omitiremos, sempre que possível, o símbolo " $\cdot$ ", ou seja, xy denota $x \cdot y$.

Nota: Se a operação binária for comutativa, G é designado por grupo abeliano.

Um subconjunto não vazio H de G não vazio diz-se subgrupo de G se H, com a operação binária que fornece a estrutura de grupo a G, é de novo um grupo. Consideramos de agora em diante, apenas grupos finitos, ou seja, G é um conjunto de um número finito de elementos. A ordem de um grupo G, denotada por $O(G)$, é o número de elementos de G.

Dado $g \in G$, o conjunto $< g >= \{g^i : i \in \mathbb{N}\}$ é um subgrupo de G. Este subgrupo é denominado grupo cíclico gerado por g. Onde g designa-se por gerador deste grupo.

Atente-se que o gerador não é único.

A ordem de $g \in G$, denotada por $o(g)$, é o menor natual k tal que $g^k = 1$.

Prova-se que a ordem de qualquer elemento divide a ordem do grupo; ou seja, $o(g)|O(G)$.

Exemplo 1, o conjunto $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n : mdc(a, n) = 1\}$ forma um grupo com a multiplicação módulo n. Se tomarmos $n = 7$, obtemos $\{1, 2, 3, 4, 5, 6\}$. Para este caso, $g = 3$, $o(3) = 6$, e portanto $\mathbb{Z}_7^* =< 3 >$.

Definição 2: Para o grupo $(\mathbb{Z}_n^*, \cdot)$, diz-se que $g \in \mathbb{Z}_n^*$ é raiz primitiva $\mod n$, caso exista, se $o(g) = O(\mathbb{Z}_n^*)$

Definição 3: Diz-se que $(G, \cdot)$ é um grupo cíclico se existir $g \in G$ tal que, para toda escolha $h \in G$, se tem $h = g^\alpha$, para algum $\alpha \in \mathbb{N}$. Ou seja, se $G =< g >$, para algum g em G.

Teorema 1: Seja p primo. Então $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$ como grupo multiplicativo é cíclico.

Demonstração (Stein, 2017):

Suponhamos p primo, queremos demonstrar que existe raiz primitiva $g \in \mathbb{Z}_p^*$.

Para $p = 2$, temos que $g = 1$.

Suponhamos $p > 2$, escrevamos $p - 1$ como produto de $q_i^{n_i}$ $(i \in \mathbb{N})$ potências primas:

$$p - 1 = q_1^{n_1} q_2^{n_2} \cdots q_r^{n_r} \tag{2.1}$$

A demonstração da seguinte implicação pode ser encontrada em (Stein, 2017, pg 41): seja p primo e d divisor de $p - 1$. Então, a função $f(x) = x^d - 1 \in \mathbb{Z}_p^*[x]$ tem exatamente d raizes em $\mathbb{Z}_p^*$.

Seja $q_i^{n_i}$ um divisor de $p - 1$, o polinómio $x^{q_i^{n_i}} - 1 \in \mathbb{Z}_p^*[x]$ tem exatamente $q_i^{n_i}$ raizes, e o polinómio $x^{q_i^{n_i-1}} - 1 \in \mathbb{Z}_p^*$ tem exactamente $q_i^{n_i-1}$ raizes.
Existem $q_i^{n_i} - q_i^{n_i-1} = q_i^{n_i-1}(q_i - 1)$ elementos $a \in \mathbb{Z}_p^*$ tais que para cada i, fixamos a_i (fatores da raiz primitiva) primos relativos entre si, em que cada a_i tem ordem $q_i^{n_i}$, e tomamos

$$a = a_1 \cdot a_2 \cdots a_r. \tag{2.2}$$

De acordo a proposição segundo a qual (Stein, 2017, pg 42): suponhamos que $a, b \in \mathbb{Z}_p^*$ têm ordens r e s, respectivamete, e que o mdc $(a, b) = 1$. Então ab tem ordem rs.

O elemento a tem ordem $q_1^{n_1} \cdot q_2^{n_2} \cdots q_r^{n_r} = p - 1$ (produto das ordens dos seus fatores). Logo, $a = g$ é a raiz primitiva de $\mathbb{Z}_p^*$.

Exemplo 2: Para $p = 13$, temos $p - 1 = 12 = 2^2 \cdot 3$.

Tomamos, então, $q_1 = 2,\ n_1 = 2,\ q_2 = 3,\ n_2 = 1$

O polinómio $x^4 - 1 = (x^2 - 1)(x^2 + 1)$ tem exatamente 4 raizes em $\mathbb{Z}_{13}^*$. Interessam-nos as que não são raizes de $x^2 - 1$, ou seja, as raizes de $x^2 + 1$.

Por exemplo, $a_1 = 5$ é raiz de $x^2 + 1$ e não é raiz de $x^2 - 1$.

O segundo polinómio que vamos considerar é $x^3 - 1$. Este polinómio tem exatamente 3 raizes em $\mathbb{Z}_{13}^*$. Queremos destas raizes, a que não é solução de $x^1 - 1 = x - 1$. Tomamos $a_2 = 9$.

Ora, a_1 e a_2 são primos relativos, sendo que a ordem de a_1 é igual a 4, e a ordem de a_2 é igual a 3.

Portanto, a ordem de $g = a = a_1 \cdot a_2 = 45 \mod 13 = 6$ é o produto das ordens, ou seja, 12.

2.2 Função de Euler

Definição 4 (função multiplicativa): Diz-se que $f : \mathbb{N} \to \mathbb{N}$ é uma função multiplicativa se para quaisquer $m, n \in \mathbb{N}$ primos relativos se tem:

$$f(m \cdot n) = f(m) \cdot f(n) \tag{2.3}$$

Definição 5: chama-se função $\phi : \mathbb{N} \to \mathbb{N}$ de Euler, ao número $\phi(m)$ de inteiros positivos menores ou iguais que m, que são primos relativos com m, para $m \in \mathbb{N}$ (Capello, 2007, p. 4).

$$\phi(m) = \#\{a \in \mathbb{N} : a \leq n \text{ e mdc } (a, m) = 1\}.$$

Exemplo 4: Calculemos a função de Euler para os casos,

$$\begin{aligned}
\phi(1) &= \#\{1\} = 1 \\
\phi(2) &= \#\{1\} = 1 \\
\phi(5) &= \#\{1, 2, 3, 4\} = 4 \\
\phi(16) &= \#\{1, 3, 5, 7, 9, 11, 13, 15\} = 8
\end{aligned}$$

Decorre da definição que $\phi(p) = p - 1$, se p for um número primo, visto que p é primo relativo com todos os números menores que ele.

Temos ainda que,

$$\begin{aligned}
&\phi(m) \leq m - 1 \text{ e que} \\
&\phi(m) = m - 1 \Leftrightarrow m \text{ é primo}
\end{aligned}$$

Para $m = p^\alpha$, $\alpha \geq 1 \in \mathbb{N}$, temos:

$$\phi(p^\alpha) = p^\alpha - \frac{p^\alpha}{p} = p^\alpha - p^{\alpha-1} = p^{\alpha-1}(p-1) \tag{2.4}$$

No caso de $m = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_n^{\alpha_n}$, temos:

$$\phi(m) = p_1^{\alpha_1-1}(p_1-1) \cdot p_2^{\alpha_2-1}(p_2-1) \cdots p_n^{\alpha_n-1}(p_n-1) \tag{2.5}$$

Se p e q são dois números primos distintos e $m = p \cdot q$, então:

$$\phi(m) = \phi(p \cdot q) = \phi(p) \cdot \phi(q) = (p-1) \cdot (q-1)$$

Pela definição, $\phi(n) = O(\mathbb{Z}_n^*)$

2.3 Criptossistema RSA

O RSA (abreviatura dos nomes de seus criadores: Ron Rivest, Adi Shamir e Leonard Adleman) é uma técnica criptográfica de chave pública criada para permitir a troca de mensagens seguras entre dois agentes, cuja segurança reside na dificuldade computacional da factorização de inteiros nos seus fatores primos (Robinson, 2003).

O RSA faz o uso de um par de chaves, onde uma é pública e outra privada. Pode ser usada para cifrar/decifrar mensagens e na assinatura digital de uma mensagem.

a) Geração de chaves:

- Escolhem-se dois números primos p e q distintos, distantes um do outro e de grande tamanho, calcula-se $n = p \cdot q$ e:

$$\phi(n) = (p-1) \cdot (q-1), \tag{2.6}$$

de seguida escolhe-se e primo relativo com $\phi(n)$ e calcula-se d tal que: $e \cdot d \equiv 1 \mod \phi(n)$, isto é $e \in \mathbb{Z}_{\phi(n)}^*$, e d é o inverso multiplicativo de $e \mod \phi(n)$. Temos:

d- é a chave privada;

(n, e)- é a chave pública. Os parâmetros e,d, e n, são designados por: expoente de cifração, expoente de decifração e módulo RSA, respetivamente. Na geração de chaves, o parâmetro de entrada (input) é o número de bits da chave, e o de saída (output) é o par de chaves.

b) Cifração:

- Tendo a mensagem M e a chave pública do destinatário, o emissor calcula e envia o criptograma C:

$$C \equiv M^e(\mod n) \tag{2.7}$$

c) Decifração:

Para a decifração do criptograma, os parâmetros de entrada são: o criptograma e o par de chaves, permitindo a recuperação do texto claro. Para tal, o recetor da mensagem calcula:

$$M \equiv C^d \equiv [(M^e)]^d(\mod n) \tag{2.8}$$

Quando o RSA é usado de forma indevida (não observando os pressupostos de geração de chave), ele torna-se sensível de sofrer um ataque por força bruta. Apesar da NIST numa das atualizações feitas em (21-11-2016) recomendar um tamanho da chave de, no mínimo 3072 bits, existem várias experiências apontando que, utilizando uma chave de 2048 bits, por exemplo, é impraticável através de um ataque por força bruta quebrá-la a um tempo razoável (Masthanamma e Preya, 2015).

Exemplo 5 (anexo 1): Vamos usar o RSA, cujo código foi construido no sagemath (python 2.0) para cifrar e decifrar a mensagem $M = 112341$, onde o número de bits é igual a 64.

Invocando a função >>> RSA (64) com 64 bits como parâmetro de entrada, calcula-se o valor de n, produto dos números primos p e q, e um par de chaves RSA, onde a chave pública tem dois parâmetros.

```
Sage: >>> p= random_prime (2^(bits/2))
2820828821
Sage: >>> q= random_prime (2^(bits/2))
1564769951
Sage: >>> n= p * q
Sage: >>> e= ZZ: random_element (phi)
Chave pública = (4413948176015557771, 2688550984574843963)
Sage: >>> d= power_mod (e, -1, phi)
Chave privada =60483247987657027
```

Cifração:

Invocando a função que recebe a mensagem a cifrar e a chave pública, obtemos o criptograma:

```
Sage: >>> Encrypt (Mensagem, pk)
16699382483954225 74
```

Decifração:

E por fim, o recetor da mensagem para recuperar o texto claro, tendo o criptograma, a sua chave privada e o primeiro elemento (n) da chave pública, recupera o texto claro invocando:

```
Sage: >>> M= Decrypt (Criptograma, chave)
112341.
```

2.4 Criptossistema Elgamal

A cifra Elgamal refere-se a um esquema de criptografia baseado no problema do logaritmo discreto. O mesmo pode ser usado para gerar assinaturas digitais, bem como para cifrar ou decifrar mensagens. Foi proposto por Taher Elgamal em 1984 (Meier, 2005).

Suponhamos que Alice e Bob são dois agentes com propósito de comunicarem entre si através de um canal. Segue-se o algoritmo:

1) Geração de chaves:

Alice escolhe um número primo p, e dois inteiros g e x, onde g é o gerador do grupo $\mathbb{Z}_p^*$ e $x \in \mathbb{Z}_p^* = \{1, 2, \cdots, p-1\}$. Em seguida, calcula $y \equiv g^x \mod p$.

x- é a chave privada;

(p, g, y)- é a chave pública.

2) Cifração:

Bob pretende enviar uma mensagem $M \in \mathbb{Z}_p^*$ (espaço de mensagens), escolhe um inteiro $k \in \{2, \cdots, p-1\}$, e com a chave pública de Alice, calcula:

$$\alpha \equiv g^k \mod p,\ \beta \equiv M \cdot y^k \mod p. \tag{2.9}$$

É enviado o criptograma $C = (\alpha, \beta)$ para Alice.

3) Decifração:

Depois de Alice ter recebido o criptograma C, e com a chave privada, calcula:

$h \equiv \alpha^x \equiv y^k \equiv g^{kx} \mod p$.

Para decifrar C, Alice aplica:

$$M \equiv h^{-1} \cdot \beta \equiv g^{(-kx)} \cdot g^{kx} \cdot M \mod p. \tag{2.10}$$

Desta forma, o texto claro M foi recuperado.

Exemplo 6 (anexo 2): Bob pretende enviar para Alice uma mensagem $m =$ 11002. Alice possui o seu par de chaves:

```
Sage: >>> Elgamal (64)
Chave privada =123337183323074633
Chave pública =(p: 5157254862981286937, g: 3, g^x: 2086489247241788091).
```

Sendo a chave pública acessível.

Cifração:

Bob com a chave pública de Alice e o texto claro, calcula o criptograma (α, β) invocando a função que recebe a chave pública e o texto claro tem:

```
Sage: >>> Encrypt (pk, Mensagem)
(270068037259658339, 4718182066144619931)
```

Decifração:

Alice por sua vez, recebe o criptograma, usando a chave pública e privada, invoca a função que recebe as duas chaves, e recupera o texto claro:

```
Sage: >>> Decrypt (pk, sk, Criptograma)

11002
```

Portanto, se uma terceira pessoa "Eva" intersetasse y, não conseguiria calcular x de forma eficiente, pois calcular x (equação 1.1), pressupõe resolver problema do logaritmo discreto, cujo cálculo é intratável computacionalmente. Este facto torna o Elgamal seguro (Hâstad, 2007).

2.4.1 Propriedade homomórfica multiplicativa do Elgamal

Sejam m_1 e m_2 duas mensagens cifradas utilizando a cifra Elgamal com a mesma chave pública (p, g, y), com os parâmetros aleatórios: $k1$ e $k2$.
Sejam $\alpha_1 \equiv \ g^{k_1} \mod p$ e $\alpha_2 \equiv g^{k_2} \mod p$. Existem dois criptogramas:

$c_1 = E(m_1) = (g^{k_1} \mod p, m_1 \cdot y^{k_1} \mod p) = (\alpha_1, \beta_1)$
$c_2 = E(m_2) = (g^{k_2} \mod p, m_2 \cdot y^{k_2} \mod p) = (\alpha_2, \beta_2)$. O produto desses criptogramas é igual ao criptograma do produto dos textos claros:

$$c_1 \cdot c_2 = E(m_1) \cdot E(m_2) = E(m_1 \cdot m_1) = C \tag{2.11}$$

Verificação:

Queremos verificar se:

$c_1 \cdot c_2 = E(m_1) \cdot E(m_2) = E(m_1 \cdot m_1) = C.$
$= (\alpha_1, \beta_1) \cdot (\alpha_2, \beta_2)$
$= (\alpha_1 \cdot \alpha_2, \beta_1 \cdot \beta_2)$

$$= (g^{k_1} \mod p \cdot g^{k_2} \mod p, m_1 \cdot g^{x^{k_1}} \mod p \cdot m_2 \cdot g^{x^{k_2}} \mod p)$$
$$= (g^{k_1} \cdot g^{k_2} \mod p, m_1 \cdot y^{k_1} \mod p) \cdot m_2 \cdot y^{k_2} \mod p)$$
$$= (g^{(k_1+k_2)} \mod p, m_1 \cdot m_2 \cdot y^{(k_1+ k_2)} \mod p)$$
$$= (g^{k} \mod p, m \cdot y^{k} \mod p), \text{ onde } k_1 + k_2 = k \text{ e } m_1 \cdot m_2 = m.$$

Logo, o produto de duas mensagens (diferentes) cifradas com a mesma chave pública é igual ao criptograma do produto dessas mensagens. Isto é:

$$E(m_1) \cdot E(m_2) = E(m_1 \cdot m_2) \tag{2.12}$$

Exemplo 7 (anexo 3): Dois eleitores A e B manifestaram seus votos, verifiquemos a propriedade multiplicativa homomórfica do Elgamal através do código construído no sagemath (python 2.0). Sejam $m_1 = 22110$ e $m_2 = 112277$, os votos.

Vamos calcular os criptogramas dos dois textos claros (diferentes), usando a mesma chave com 64 bits como parâmetro de entrada. Seja o grupo $\mathbb{Z}_p^*$, $g \in \mathbb{Z}_p^*$ gerador do grupo, Alice gera a chave privada, e a chave pública. Invocando a função que recebe o número de bits, tem-se:

```
Sage:  >>> Elgamal (64)
Chave privada = 5639733198545457430
Chave publica = 12095632392767911067, 2, 762127733165538539.
```

Bob escolhe k_1 e k_2, elementos de $\mathbb{Z}_p^*$, calcula $g^{xk_1}(\mod p)$ e $g^{xk2}(\mod p)$, dois valores diferentes para cifrar m_1 e m_2, e em seguida invoca as funções:

```
Sage: >>> Encrypt (pk, Mensagem1)
Criptograma1 = [7328698325914942318, 5585731434412057124]


Sage: >>> Encrypt (pk, Mensagem2)
Criptograma2= [11086177226526804309, 6959705628603275349]
```

Calculando o produto dos criptogramas:

O produto dos primeiros e dos segundos elementos dos respetivos criptogramas, permite obter o produto dos criptogramas. Com os criptogramas recebidos, Alice invoca a função que recebe os dois criptogramas:

```
Sage: >>> Encrypt3 (Criptograma1, Criptograma2)
Criptograma3= [1952137899274186311, 369731871242704007]
```

Calculando o criptograma do produto das mensagens:

multiplicam-se primeiramente os parâmetros α_1 por α_2, e β_1 por β_2, Bob invocando a função que tem, como parâmetros de entrada, a chave pública, a mensagem

resultante do produto das mensagens, e os inteiros $k1$ e $k2$ distintos. Obtém-se o criptograma do produto:

```
Sage: >>> Encrypt4 (pk, Mensagem, k1, k2)
Criptograma4= [1952137899274186311, 369731871242704007]
```

Portanto, Alice verifica se: $E(m_1) \cdot E(m_2) = E(m_1 \cdot m_2)$ invocando a função que recebe o produto dos criptogramas e o criptograma do produto.

`Sage: >>> Equal (Criptograma3, Criptograma4):`

Retorna True (verdade), e caso não haver correspondência retorna False (falso).

2.4.2 Elgamal exponencial

O esquema Elgamal pode ser representado na forma exponencial.

Geração de chaves:

O processo de geração de chaves é análogo ao anterior.

x – é chave privada e (p, g, y) – chave pública, com $y \equiv g^x \mod p$, onde g é uma raiz primitiva.

Cifração:

O criptograma é dado por:

$$C = (\alpha, \beta) = (g^k \mod p, g^m \cdot y^k \mod p) \tag{2.13}$$

Decifração:

O texto puro é recuperado:

$$g^m \equiv y^{(-k)} \cdot g^m \cdot y^k \mod p \tag{2.14}$$

2.4.3 Propriedade homomórfica aditiva do Elgamal

Sejam m_1 e m_2 duas mensagens cifradas utilizando o Elgamal exponencial com a mesma chave pública. Sejam c_1 e c_2 os criptogramas dessas mensagens:

$$c_1 = (g^{(k_1)} \mod , g^{(m_1)} \cdot y^{(k_1)} \mod p) \tag{2.15}$$

$$c_2 = (g^{(k_2)} \mod p, g^{(m_2)} \cdot y^{(k_2)} \mod p), \tag{2.16}$$

temos que o produto dos criptogramas é igual ao criptograma da soma das mensagens.

$$c_1 \cdot c_2 = E(m_1) \cdot E(m_2) = E(m_1 + m_2) = g^m \cdot y^k \mod p = C \tag{2.17}$$

Verificação:

Suponhamos que $c_1 = E(m_1)$ e $c_2 = E(m_2)$, queremos verificar se:
$E(m_1) \cdot E(m_2) = E(m_1 + m_2) = C$.

$$\begin{aligned} c_1 \cdot c_2 &= (g^{k_1} \mod p, g^{m_1} \cdot y^{k_1} \mod p) \cdot (g^{k_2} \mod p, g^{m_2} \cdot y^{k_2} \mod p) \\ &= (g^{k_1} \cdot g^{k_2} \mod p, g^{m_1} \cdot g^{m_2} \cdot y^{k_1} \cdot y^{k_2} \mod p) \\ &= (g^{k_1+k_2} \mod p,\ g^{m_1+m_2} \cdot y^{k_1+k_2} \mod p) \\ &= (g^{k} \mod p,\ g^{m} \cdot y^{k} \mod p) = C, \end{aligned}$$

onde $k_1 + k_2 = k$ e $m_1 + m_2 = m$.

Exemplo 8 (anexo 4): Consideremos para p primo o grupo $\mathbb{Z}_p^*$, com $g \in \mathbb{Z}_p^*$ gerador deste grupo. Suponhamos duas mensagens $m_1 = 173, m_2 = 411 \in \mathbb{Z}_p^*$. Vamos verificar a propriedade aditiva através da aplicação do código construido no sagemath (python 2.0).

Para gerar uma chave privada e outra pública de 64 bits temos de invocar a função:

```
Sage: >>> Elgamaladitprop (64)
Chave privada = 11859750610466699932
Chave pública = 151522339214100708211, 2, 11856649396572359821
```

O modo de invocar as funções para gerar os criptogramas não difere da propriedade anterior, pois as suas construções diferem-se fundamentalmente nos sinais: $\cdot$ e $+$.

Cálculo dos criptogramas

```
Criptograma1 = [4088660568012678654, 87312942481012772 99]
Criptograma2= [530094874280869351, 5787818172918183661]
```

Cálculo do produto dos criptogramas:

`[44842915018668594 06, 1140529719004063540]`

Cálculo do criptograma da soma:

`[4484291501866859406, 1140529719004063540]`

Verificação:

Invoca-se a função que recebe os dois criptogramas para verificar se:
$E(m_1) \cdot E(m_2) = E(m_1 + m_2)$. Que resulta em "**True**", caso contrário, resulta em **False**.

Capítulo 3

Eleição eletrónica

Neste capítulo falou-se de forma sintética acerca da eleição eletrónica, pois o capítulo a seguir faz o uso de muitos conceitos aqui tratadados, assim como, requisitos de seguraça, intevenientes, estilos de voto de uma eleição eletrónica, por exemplo.

Uma eleição eletrónica não é simplesmente uma troca de ferramentas e materiais. Não significa passar de urna de madeira, plástica, ou de metal para uma urna eletrónica. É muito mais que as possibilidades/ funcionalidades que o novo sistema oferece, pois permite redesenhar, corrigindo o sistema eleitoral (Prince, 2004).

Um sistema de eleição eletrónica é qualquer sistema de eleição eletrónica que utilize meios eletrónicos nas fases de votação, ou contagens dos votos. O processo de eleição eletrónica tem o objetivo de automatizar os diferentes processos eleitorais com intuito de se obter resultados eficientes, isto é, resultados justos e verdadeiros (Prince, 2004).

Os principais tipos de eleição eletrónica são (Prince, 2004):

a) Eleição eletrónica remota: o eleitor manifesta o seu voto através da internet mediante o uso de um meio tecnológico (com acesso à internet).

b) Eleição eletrónica presencial: este tipo de eleição implica o uso de captação eletrónica do voto com transmissão e escrutínio provisório através de uma urna eletrónica colocada nos lugares (físicos) onde se realizam a eleição.

3.1 Estilos de um voto eletrónico

Uma eleição pode ocorrer com diferentes propósitos, tendo como consequência as formas distintas de apresentar o voto. Ou seja, o número de opções ou candidatos possíveis a serem apresentados ao eleitor para que este possa realizar suas escolhas. Estas formas de apresentação do voto são denominadas de estilos de voto, e são descritas a seguir (Mursi, 2013):

a) Uma escolha de 2 opções de voto (sim/não): a resposta do eleitor é sim ou não. Cada voto emitido representa: 1 para sim e 0 para não.

b) Uma escolha de n opções de eleição: a partir de N opções o eleitor realiza uma escolha que é representada como 1, e as demais como 0.

c) k escolhas de n opções de eleição: o eleitor realiza diferentes k escolhas de um conjunto de n possibilidades. A ordem dos elementos selecionados não é importante.

d) k escolhas de n opções ordenadas: o eleitor põe em ordem diferentes k escolhas de um conjunto de n possibilidades. Neste caso, a ordem é importante.

e) Voto escrito em boletins: o eleitor formula sua própria resposta e escreve o voto. Votar é uma sequência de letras com um tamanho máximo, que representa o nome de uma pessoa, por exemplo.

3.2 Intervenientes da eleição eletrónica

Baseando-se nas eleições tradicionais, os atores do sistema de eleição eletrónica são:

a) Comissão eleitoral: é responsável por todo processo eleitoral, incluindo o recenseamento eleitoral, a gestão do sistema de voto e autenticação da informação publicada;

b) Auditores: correspondem, em parte, à figura dos delegados de listas nas eleições tradicionais, e são responsáveis pelo controlo no que respeita a privacidade do eleitor e integridade da eleição.

c) Entidades de Verificação: são responsáveis por, de forma independente, verificar a validade e correção da eleição. Verificam também os resultados eleitorais. São em geral, entidades diretamente interessadas nos resultados da eleição (partidos políticos).

d) Eleitor: Corresponde a qualquer cidadão com direito de voto, e cada eleitor terá de se recensear uma única vez juntamente da comissão eleitoral, sendo necessário registar-se para cada eleição antes do processo de eleição.

e) Adversário: um adversário é uma entidade maliciosa, que tenta manipular o processo eleitoral, sua apuração ou o eleitor. O adversário pode atuar em um momento isolado ou em vários momentos de uma eleição. É possível dividir sua atuação em dois tipos:

- Adversário externo: fazem parte deste grupo, entidades sem intervenção direta no processo eleitoral, capazes de realizar uma ação maliciosa; isto é, não elegem, organizam o processo, nem distribuem equipamentos eletrónicos. Exem-

plo deste tipo de adversário, pode ser qualquer pessoa com capacidade técnica de promover ataques.

- Adversário interno: refere-se a entidades com intervenção direta no processo eleitoral, capazes de realizar uma ação maliciosa. Por exemplo: pode ser um eleitor que queira expandir seu único voto para contabilizar 1.000 ou mais; um distribuidor de equipamento de hardware ou software com alguma vulnerabilidade que possa ser explorada num ataque futuro; ou uma autoridade corrupta (Mursi, 2013).

3.3 Requisitos de segurança

Uma eleição eletrónica deve cumprir com uma série de requisitos, dos quais os principais são (Prince, 2004):

a) Anonimato e confidencialidade: deve-se garantir o anonimato do eleitor e a confidencialidade do voto emitido. O responsável pelo escrutínio não deve saber quem votou em quem, e apenas agentes autorizados devem ter acesso ao voto.

b) Elegibilidade e Autenticidade: Só podem votar os agentes devidamente autorizados, garantindo que o voto emitido pertence ao eleitor esperado. Sempre que o voto for alterado, não será autêntico, logo será rejeitado.

c) Unicidade: deve-se garantir que cada eleitor só vota uma única vez.

d) Integridade: garantir que os votos não foram adulterados, nem tampouco eliminados. Deve-se procurar mecanismos para o correto armazenamento do voto e de toda informação auxiliar.

e) Verificável: todo processo deve ser verificável. O processo deve ser auditável (por terceiros).

f) Confiável: o sistema de eleição deve funcionar de maneira robusta, isto é, sem permitir a perca do voto, nem de dados ou informação.

Selker (2003) também apresenta 4 requisitos:

1) Assegurar o armazenamento do voto, através de uma arquitetura que permita a detecção e correção de erros.

2) Prevenção contra alterações externas, essencialmente as que envolvem a modificação do voto.

3) Prevenção contra alterações internas, que envolvem o desenvolvimento malicioso (vulnerabilidades) do sistema de eleição.

4) Permitir a detenção de falsificação dos conteúdos dos arquivos que o sistema utiliza ou gera.

Portanto, na eleição eletrónica, a garantia das propriedades criptográficas que permitem a obtenção de um voto seguro reveste-se de grande importância, visto que

a segurança neste processo é um fator primordial para a validação do processo. No processo de eleição eletrónica deve-se garantir o anonimato do eleitor, a confidencialidade, integridade, autenticidade e verificabilidade do voto, sob pena de vermos as políticas de segurança de uma eleição eletrónica comprometidas. E durante a abordagem em causa, procurou-se esclarecer de forma detalhada as técnicas que permitem este fato, culminando com a construção de um esquema criptográfico que visa garantir estes cinco requisitos.

Capítulo 4

Abordagem criptográfica a eleição eletrónica

Neste capítulo detalharam-se as técnicas criptográficas: assinatura cega, esquema de partilha secret verificável de Shamir e prova de conhecimento zero (nulo), bem como suas devidas aplicações numa eleição eletrónica. E no final do capítulo propôs-se um esquema criptográfico que permite garantir a segurança do voto numa eleição eletrónica.

4.1 Assinatura cega

A assinatura cega é uma forma de assinatura digital onde o conteúdo de uma mensagem é ofuscada antes de ser assinada. A assinatura cega resultante pode ser publicamente verificada em relação à mensagem original (Chaum, 1998).

Ofuscar uma mensagem significa tornar o conteúdo da mensagem inalcançável. Mas, assinar uma mensagem de forma cega consiste em solicitar a uma terceira entidade para a realização da assinatura sem que esta saiba o conteúdo da referida mensagem. Visto que, antes do envio, foi ofuscada pelo titular da mensagem.

Uma assinatura cega pode ser utilizada em protocolos em que o signatário e o autor da mensagem são partes diferentes. Consequentemente, uma assinatura cega tem utilidade em transações, tais como: moeda eletrónica e eleição eletrónica.

Um dos requisitos de segurança de um voto eletrónico, é precisamente a garantia do anonimato do eleitor, isto é, o destinatário não deve saber se Alice votou em quem. O anonimato do eleitor deve ser tomado em conta durante o processo. A aplicação da assinatura cega numa eleição eletrónica vem garantir este requisito, na medida em que a assinatura contida num voto é verificável, mas não pertence ao eleitor (Alice). Vejamos que nem o signatário, nem o verificador, são capazes de

identificar o autor do voto exercido a favor de um determinado partido (candidato) político.

Existem vários esquemas de assinatura cega versados em criptografia de chave pública, tais como: RSA, ECDSA e a versão de Schnorr (1989) .

A segurança da versão RSA (Rivest–Shamir–Adleman) consiste na dificuldade computacional da factorização de inteiros em números primos. A segurança das outras duas versões, basea-se na intratabilidade computacional do logaritmo discreto (equação 1.1).

No que diz respeito à assinatura cega versado no ECDSA (Elliptic Curve Digital Signature Algorithm), a sua segurança cinge-se fundamentalmente nas vantagens criptográficas das curvas elípticas sobre um corpo finito: ganhos em eficiência, isto é, para o mesmo nível de segurança, uma técnica com curvas elípticas ocupa pouco espaço e é frequentemente mais rápida do que uma técnica equivalente em $\mathbb{Z}_p^*$.

4.1.1 Assinatura cega versão RSA

Para realizar a assinatura cega, primeiro a mensagem é ofuscada tipicamente combinando-a (multiplicando) com um número inteiro k designado por fator de ofuscação.

Sejam Alice (eleitor), CA (autoridade de confiança) e Bob (responsável pelo escrutínio), que têm o papel de: 1) eleger, ofuscar/ desofuscar o voto, 2) assinar o voto e 3) verificar a autenticidade, respetivamente. O eleitor exerce seu direito de voto, e em seguida ofusca-o, solicita à CA para assinar o voto, e depois de assinar, a CA devolve-o para Alice, e esta por sua vez, desofusca-o usando o mesmo fator de ofuscação/ desofuscacão.

A mensagem (voto) assinada e ofuscado resultante pode ser verificada posteriormente pelo responsável pelo escrutínio em relação à chave pública da CA.

O algoritmo do esquema de assinatura cega consubstancia-se em:

a) Geração de chaves:

De acordo a alínea a) da secção 2.3, tem-se o par de chaves RSA:

$(d, (n, e))$, chave privada e chave pública, respetivamente.

b) Ofuscação:

O eleitor com o seu voto m, gera um inteiro $k_0 \in \mathbb{Z}_n^*$ que é partilhado de antemão com o verificador da assinatura, e calcula:

$$m_0 = k_0^e \cdot m \mod n, \tag{4.1}$$

envia a mensagem m (voto) ofuscada para a CA.

c) Assinatura

O signatário assina m_0 (voto ofuscado) usando d:

$$S' = m_0^d \mod n, \tag{4.2}$$

envia S' para o eleitor.

d) Desofuscacão:

O eleitor desofusca o voto assinado para ter certeza que o assinado pertence-lhe, a seguida volta ofusca-lo e envia-o:

$$S = k_0^{(-1)} \cdot S' \mod n \tag{4.3}$$

Substituindo (4.1) e (4.2) em S:

$$\begin{aligned} S &= k_0^{(-1)} \cdot (k_0^e \cdot m)^d \mod n \\ &= k_0^{(-1)} \cdot k_0^{ed} \cdot m^d \mod n \\ &= m^d \mod n \end{aligned}$$

Alice deposita o seu voto numa urna eletrónica que será aberta posteriormente para o escrutínio.

e) Verificacão:

Se $S^e \mod n = m$, então o destinatário aceita a mensagem. E se for diferente, rejeita-a, pois isto significa que o mesmo foi adulterado, ou substituido.

Obs: Como a verificação da assinatura S permite ter acesso ao conteúdo do voto de Alice, mas a entidade do escrutínio não associa o voto ao seu correspondente eleitor, visto que o voto não assinado pelo eleitor, e também não enviado diretamente para a entidade do escrutínio.

Exemplo 9 (anexo 5): suponhamos que temos a mensagem $m = 709980$, e queremos aplicar a assinatura cega.

Calculando n, produto dos números primos p e q, e o par de chaves RSA, invocando a função que recebe 64 bits:

```
Sage: >>> BlindSigRSA (64)
Chave privada = 7898280744143511092764637749 3
Chave pública = (792281627909654987550217996 19,  169583053666264493085329903 17)
```

- Ofuscação:

O autor da mensagem, de modo a preservar o seu anonimato, ofusca a mensagem invocando a função, onde o texto claro e a chave pública são os parâmetros de entrada.

```
Sage: >>> Blind (m, pk)
907071777250742915349169524 1
```

- Assinatura:

O signatário gera a assinatura no resultado da ofuscação, invocando a função, que recebe a mensagem ofuscada e o par de chaves.

```
Sage: >>> BlindSign (bmess, sk, pk)
261655983940725955107358626 83
```

- Desofuscação:

O signatário devolve a mensagem assinada para o emissor, e este por sua vez, efetua a desofuscação para obter a assinatura. Para tal invoca a função cujos parâmetros de entrada são: a assinatura da mensagem ofuscada, o fator de ofuscação e a chave pública.

```
Sage: >>> UnBlind (bsig, bfactor, pk)
 532575483334280074210576335 06
```

- Verificação:

O recetor da mensagem, verifica a autenticidade da mensagem, invocando a função cujos parâmetros de entrada são: a mensagem, chave pública e a assinatura. Se a potência da assinatura de expoente e (chave pública) for igual ao texto claro, então ela é autêntica, senão rejeita-se a mensagem.

`Sage: >>> verify (m, pk, desof)`

True ou **False**.

4.2 Partilha da chave secreta

Num esquema criptográfico, a sensibilidade da chave secreta (decifração) obriga que Bob (recetor da mensagem) adote um mecanismo para o armazenamento da chave observando os requisitos de segurança, visto que o acesso à chave privada por um agente não autorizado, por exemplo, comprometeria o processo em causa. Daí que o seu armazenamento deve ser criteriosamente bem tratado. Isto é, deve ter um alto nível de segurança. A partilha de uma chave (informação) secreta adotado no trabalho, foi proposto independentemente por Adi Shamir em 1979 (Shamir, 1979).

Existem várias maneiras de aumentar a confiabilidade de uma chave (Shoenmakers, 2018):

(1) Manter uma única cópia da chave em um local de sigilo máximo.

(2) Mantendo várias cópias da chave em locais diferentes para maior confiabilidade. Mas, aumentar a confiabilidade da chave armazenando várias cópias, reduz a confidencialidade, criando vetores de ataque.

(3) Dividir o segredo em várias partes e distribuí-las entre um grupo de participantes.

Em geral, partilhar uma informação secreta pressupõe a aplicação de métodos para distribuir esta informação (segredo) entre um grupo de participantes, onde cada um deles, recebe uma porção (parte) do segredo.

Portanto, é necessário que numa eleição eletrónica por exemplo, a chave usada para a decifração dos votos seja partilhada, pois para a sua reconstrução pressupõe um prévio acordo entre os elementos do grupo. Qualquer iniciativa isolada (individual) que visa obter o segredo não pode ser eficiente. Importa realçar que esta é uma forma de armazenamento da chave, pois existem outros mecanismos que podem ser usados para o armazenamento da chave, isto é, pode-se cifrar a chave usando um algoritmo criptográfico forte (envelope digital, por exemplo).

Na próxima seccção abordamos o polinómio interpolador de Lagrange:

(1) O esquema de Shamir usa polinómios em um corpo finito.

(2) Na reconstrução das partes secretas é necessário calcular o polinómio interpolador de Lagrange.

4.2.1 Polinómio interpolador de Lagrange

Sejam $(x_0, y_0), (x_1, y_1), \cdots, (x_{(n-1)}, y_{(n-1)}), (x_n, y_n)$ conjunto de pares ordenados. O problema da interpolação consiste na determinação de uma função polinomial f, designada por polinómio interpolador, onde $f(x_i) = y_i$, com $i = 1, 2, \cdots, n$ (Matos, 2005).

As abcissas $x_0, x_1, \cdots, x_{(n-1)}, x_n$ designam-se por nós da interpolação, tais que: $i \neq j$ isto é, devem ser todos diferentes, e as ordenadas $y_0, y_1, \cdots, y_{(n-1)}, y_n$ designam-se por valores modais.

Definição 6: Sejam $x_0, \cdots, x_{n-1}, x_n$ nós distintos, o polinómio definido pela expressão

$$l_j(x) = \prod_{(i=0, i\neq j)}^{n} \frac{x - x_i}{x_j - x_i}, \tag{4.4}$$

onde, $j = 0, 1, \cdots, n$. $l_j(x)$ designa-se por polinómio de Lagrange relativo ao j.

Da definição, decorre que os índices i e j variam de 0 a n, tais que (5.1).

Definição 7: O polinómio interpolador de Lagrange define-se por:

$$P(x) = \sum_{j=0}^{n} y_j \cdot l_j(x), \tag{4.5}$$

ou seja:

$$P(x) = y_0 \cdot \ l_0(x) + y_1 \ \cdot \ l_1(x) + y_2 \cdot l_2(x) + \cdots + y_n l_n(x)$$

Vejamos o que acontece para $n = 2$:

Sejam $(x_0, y_0), (x_1, y_1), (x_2, y_2)$, calculam o os polinómios relativos ao valores $j = 0, 1, 2$.

$$\begin{aligned} l_0(x) &= \tfrac{(x-x_1)}{(x_0-x_1)} \cdot \tfrac{(x-x_2)}{(x_0-x_2)} \ (*) \\ l_1(x) &= \tfrac{(x-x_0)}{(x_1-x_0)} \cdot \tfrac{(x-x_2)}{(x_1-x_2)} \ (**) \\ l_2(x) &= \tfrac{(x-x_0)}{(x_2-x_0)} \cdot \tfrac{(x-x_1)}{(x_2-x_1)} \ (***) \end{aligned}$$

O polinómio interpolador de Lagrange para $n = 2$ é:

$$P(x) = y_0 \cdot \tfrac{(x-x_1)}{(x_0-x_1)} \cdot \tfrac{(x-x_2)}{(x_0-x_2)} + y_1 \cdot \tfrac{(x-x_0)}{(x_1-x_0)} \cdot \tfrac{(x-x_2)}{(x_1-x_2)} + y_2 \cdot \tfrac{(x-x_0)}{(x_2-x_0)} \cdot \tfrac{(x-x_1)}{(x_2-x_1)}$$

Exemplo 10: Determinemos em $\mathbb{R}$ o polinómio interpolador de Lagrange para os valores:
$(x_0, y_0) = (1, 2), (x_1, y_1) = (2, 1), (x_2, y_2) = (3, 0)$.

Calculando os polinómios de Lagrange relativos aos nós, $l_j(x)$:

$$l_j(x) = \prod_{(i=0, i\neq j)}^{n} \frac{(x - x_i)}{(x_j - x_i)} \tag{4.6}$$

$$\begin{aligned} l_0(x) &= \tfrac{(x-x_1)}{(x_0-x_1)} \cdot \tfrac{(x-x_2)}{(x_0-x_2)} \\ &= \tfrac{(x-2)}{(1-2)} \cdot \tfrac{(x-3)}{(1-3)} \\ &= \tfrac{(x^2-5x+6)}{2} \\ &= \tfrac{1}{2}x^2 - \tfrac{5}{2}x + 3 \end{aligned}$$

$$\begin{aligned} l_1(x) &= \tfrac{(x-x_0)}{(x_1-x_0)} \cdot \tfrac{(x-x_2)}{(x_1-x_2)} \\ &= \tfrac{(x-1)}{(2-1)} \cdot \tfrac{(x-3)}{(2-3)} \\ &= \tfrac{(x^2-4x+3)}{-1} \\ &= -x^2 + 4x - 3 \end{aligned}$$

$$\begin{aligned} l_2(x) &= \tfrac{(x-x_0)}{(x_2-x_0)} \cdot \tfrac{(x-x_1)}{(x_2-x_1)} \\ &= \tfrac{(x-1)}{(3-1)} \cdot \tfrac{(x-2)}{(3-2)} \\ &= \tfrac{(x^2-3x+2)}{2} \\ &= \tfrac{1}{2}x^2 - \tfrac{3}{2}x + 1 \end{aligned}$$

Então, o polinómio interpolador de Lagrange é:

$$\begin{aligned} P(x) &= \textstyle\sum_{j=0}^{n} y_j \cdot l_j(x) \\ &= y_0 \cdot l_0(x) + y_1 \cdot l_1(x) + y_2 \cdot l_2(x) \\ &= 2(\tfrac{1/2x^2-5}{2x+3}) + (-x^2 + 4x - 3) \\ &= x^2 - 5x + 6 - x^2 + \tfrac{3}{2} \\ &= -x + 3 \end{aligned}$$

Teorema 2: Um polinómio $P(x)$ de grau menor ou igual a n, que interpola o conjunto de valores $y_0, y_1, \cdots, y_{(n-1)}, y_n$ nos nós $x_0, x_1, \cdots, x_n$, respetivamente é dado por:

$$P(x) \;=\; \sum_{j=0}^{n} y_j \cdot l_j(x), \tag{4.7}$$

onde $l_j(x)$ (confome 5.1) .

Demonstração:

Queremos demonstrar que $P(x)$ é um polinómio de grau menor ou igual a n.

Note que $l_j(x)$ é um polinómio de grau menor ou igual a n, porque $l_j(x)$ é o produto de n polinómios de grau 1. O polinómio $l_j(x)$ tem grau exatamente igual a n.

Como $P(x)$ é a soma de $y_j \cdot l_j(x)$ $(j = 0, \cdots, n)$, polinómios de grau menor ou igual a n, então $P(x)$ é um polinómio de grau menor ou igual a n.

Como o grau de $l_j(x)$ é igual a n, então $kl_j(x)$, tem grau igual a n, se $k \neq 0$, ou então $kl_j(x)$ é um polinómio nulo.

Para cada x_i se tem:

$$P(x_i) \;=\; \sum_{j=0}^{n} y_j \cdot l_j(x_i) = y_i. \tag{4.8}$$

Exemplo 11: Determinemos em $\mathbb{R}$ o polinómio interpolador de Lagrange para os seguintes valores:

$(x_0, y_0) = (1, 1), (x_1, y_1) = (2, 8), (x_2, y_2) = (3, 27),$
$(x_3, y_3) = (4, 81)$

Calculando os polinómios de Lagrange relativos aos nós, $l_j(x)$:

$$l_j(x) = \prod_{(i=0, i\neq j)}^{n} \frac{(x - x_i)}{(x_j - x_i)} \tag{4.9}$$

$$l_0(x) = \frac{-1}{6}x^3 + \frac{3}{2}x^2 - \frac{13}{3}x + 4$$

$$l_1(x) \ = \frac{1}{2}x^3 + 4x^2 + \frac{19}{2}x - 6$$

$$l_2(x) = \frac{-1}{2}x^3 + \frac{7}{2}x^2 - 7x + 4$$

$$l_3(x) = \frac{1}{6}x^3 - x^2 + \frac{11}{6}x - 1$$

Então, o polinómio interpolador resultante é:

$$\begin{aligned} P(x) &= \textstyle\sum_{j=0}^{n} \ y_j \cdot l_j(x) \\ P(x) &= y_0 \ \cdot l_0(x) + y_1 \cdot l_1(x) + y_2 \cdot l_2(x) + y_3 \cdot l_3(x) \\ P(x) &= \tfrac{23}{6}x^3 + 47x^2 + 29x - 17 \end{aligned}$$

Teorema 3 (**unicidade do polinómio interpolador:**) Sejam $P_1(x)$ e $P_2(x)$ polinómios de iguais graus menores ou iguais a n, que assumem os mesmos valores num conjunto de nós $x_0, x_1, \cdots, x_{(n-1)}, x_n$ distintos. Então, estes polinómios são iguais.

Demonstração:

Queremos demonstrar que $P_1(x) = P_2(x)$, ou seja, o polinómio diferença

$d(x) = P_1(x) - P_2(x) = 0$.

Por redução ao absurdo, suponhamos que: $0 \leq$ grau $d(x) \leq n$.

Do teorema temos que, $P_1(x)$ e $P_2(x)$ assumem valores $x_0, x_1, \cdots, x_{(n-1)}, x_n$. E sabe-se que x_i é raiz de um polinómio $Q(x)$ se $Q(x_i) = 0$.

Como $P_1(x_i) - P_2(x_i) = d(x_i) = 0$, então: $x_0, x_1, \cdots x_{(n-1)}, x_n$ são raizes de $d(x)$, e assim, $d(x)$ é dado por:

$$d(x) = (x - x_0) \cdot (x - x_1) \cdots (x - x_{(n-1)}) \cdot (x - x_n) \cdot t(x) \tag{4.10}$$

Suponhamos que $t(x)$ é um polinómio de grau $m \geq 0$.

Multiplicando os n-ésimos fatores de $d(x)$, seu grau será $n + 1 + m \geq n$.
Chegamos a uma contradição, pois, $d(x)$ tem grau menor ou igual a n.
Então, $t(x)$ não tem grau $m \geq 0$, isto é, $t(x) = 0$.

Logo, $d(x) = 0$.

Conclui-se que, o polinómio interpolador é único, sendo o de Lagrange interpolador (teorema 2) então não há mais nenhum.

4.2.2 Esquema de Shamir

O esquema de partilha de segredo de Shamir é um algoritmo em criptografia criado por Adi Shamir (1979).

Definição 8: o esquema de Shamir com parâmetros (b, n), consiste em dividir um segredo S em n pedaços $S_1, \cdot, S_n$, de modo que:

- O parâmetro n indica o número total de participantes;

- O parâmetro b indica o número de partes secretas necessárias para reconstruir S.

- O segredo S seja reconstruído a partir de qualquer combinação de b partes do segredo.

- O segredo S seja difícil de ser reconstruído com menos de b partes.

Esta é uma forma de partilha secreta, onde um segredo é dividido em n partes e distribuído a cada participante uma parte única. Se $b = n$, então é necessário juntar todas as partes secretas de S para reconstruí-lo (Shoenmakers, 2018).

O segredo só pode ser reconstruído quando um número k menor ou igual a n de elementos reunirem seus segredos. Com k partes secretas deve ser possível reconstruir o segredo S.

Suponhamos que, usando (b, n), queremos partilhar um segredo S, onde, $1 < k \leq n$. Temos o algoritmo:

a) Sejam n, k S e $G = \mathbb{Z}_p^*$ um grupo cíclico de ordem $p-1$, com $g \in G$ gerador do grupo.

b) Escolhem-se $b-1$ elementos $a_i \in G$ tais que: $i = 1, 2, \cdots, b-1$, e escolhe-se $a_0 = S$.

c) Constrói e fixa-se o polinómio:

$$P(x) = S + \sum_{i=1}^{b-1} a_i x^i = a_0 + a_1 x + \cdots + a_{b-1} x^{b-1} \tag{4.11}$$

d) Calcula-se os pares ordenados resultantes da substituição de $t = 1, 2, \cdots, n$ em $P(x)$, isto é, calculam-se as partes secretas:

$$S_{t-1} = (t, P(t)_i), \tag{4.12}$$

com $i = 1, \cdots, n$

A partilha é feita distribuindo a cada participante uma parte única $S_{(t-1)}$ correspondente.

e) A reconstrução do segredo é feita mediante o cálculo do polinómio que interpola as b partes secretas. Isto é, as b partes $(S_{(t-1)})$ escolhidas entre as n. Em seguida, calcula-se o polinómio interpolador de Lagrange, obtendo assim, $S = a_0$.

Exemplo 12 (anexo 6): suponhamos que queremos partilhar o segredo $S = 9870000$ entre 5 elementos, sendo necessário apenas 3 deles para reconstruí-lo.

Usando o código construído no sagemath, temos: $n = 5, b = 3$ e $a_0 = S$. Escolhe-se $b - 1 = 2$ números inteiros $a_i \in Z_p^*$, que serão coeficientes do polinómio interpolador, invocando a função, cujos parâmetros de entrada são: o segredo, n, b e p:

```
Sage: >>> p= random prime(2**bits)
11737979611798134169
Sage: >>> Coeficientes (p, secret, n, b)
[a0= 9780000, a1= 2345550274137265079, a2= 2961391170802148976]
```

O polinómio $P(x)$ é construído invocando, função que recebe os coeficientes:

```
Sage: >>> polinomio (coefpol)
```

```
P(x)= 2961391170802148976*x^2 + 234555027413726507 9*x + 9780000
```

Constroem-se as $n = 5$ partes a serem partilhadas. Basta invocar a função que recebe parâmetros n e o polinómio:

```
Sage: >>>  shared(n, polconst)
[1, 5306941444949194055];
[2, 16536665231492906062];
[3, 33689171359640916021];
[4, 56764459829393223932];
[5, 85762530640749829795].
```

Então, temos a partilha feita. Para reconstruir $P(x)$, de modo a recuperar o segredo S, basta escolher $b = 3$ partes secretas entre as 5.

Havwndo necessidade de obter a chave privadapara decifrar os votos, escolhem-se b partes secretas, e invocando a função que recebe os coeficientes e as 3 partes secretas, tem-se:

```
Sage: >>> polrec (coefpol, partes)
P(x)= 2961391170802148976*x^2 + 234555027413726507 9*x + 9780000
```

Portanto, temos o segredo `a0= 9780000` recuperado.

4.2.3 Esquema de verificação de Feldman

O esquema de Shamir para além de nos permitir partilhar e reconstruir uma informação secreta, também é verificável. Isto é, também é possível verificar se as partes distribuidas são válidas.

Um esquema de partilha de um segredo deve satisfazer dois requisitos (Feldman, 1987):

a) Restrição de verificação: ao receber uma parte do segredo, um participante deve ser capaz de testar se é ou não uma parte válida (consistente). Se as partes secretas são válidas, então existe um único segredo que será recuperado quando forem combinadas k partes válidas e distintas.

b) Imprevisibilidade: não existe uma estratégia que permite a escolha de partes do segredo, de modo que elas possam ser usadas para parecer ao segredo. Em particular, as partes secretas são diferentes do segredo.

A verificação das partes secretas centra-se no facto de saber se todas as partes partilhadas são consistentes. Isto é, cada participante pode verificar se a parte

secreta recebida, quando combinada com outras partes secretas (de outros participantes) permite recuperar a informação secreta (esperada).

O esquema de Feldman é definido em $\mathbb{Z}_p^*$ para p primo, onde g é o gerador do grupo.

A distribuem-se simultâneamente as partes secretas e as $b-1$ potências:

$$C_i = g^{(a_i)} \mod p, \tag{4.13}$$

onde a_i, são os coeficientes de $P(x)$, e $i = 0, \cdots, b-1$. Onde a_0 é a chave privada.

Cada participante pode verificar se a parte secreta recebida $S_t - 1$ é válida, calculando:

$$g^{(S_{t-1})} = C_j^{t^j} \cdot (C_{j+1})^{t^{(j+1)}} \ldots C_{b-1}^{t^{(b-1)}} = g^{P(t)_i}, \tag{4.14}$$

onde $j = 0, \cdots, (b-1)$, e $i = t = 1, \cdots, n$, representa a posição de cada participante. Em $C_0 = g^{S_t - 1}$, não é possível de forma eficiente obter alguma informação sobre o segredo, pelo problema do logaritmo discreto (equação 1.1).

Exemplo 13 (anexo 6): vamos verificar se todas as partes secretas vistas no exemplo anterior são consistentes, e recorrendo ao código construido no sagemath (python 2.0).

Com os parâmetros de entrada: g gerador do grupo $\mathbb{Z}_p^*$ e os a_i coeficientes do polinómio interpolador, calculam-se $b-1$ potências que são distribuídas juntamente com as partes secretas aos participantes do esquema.

```
Sage: >>> potencia(g, coefpol):
[2763625451175465918];
[263867074193122205];
[1031831021283536856].
```

Com estas potências, o gerador do grupo, a parte secreta $P(t)$ e t (posição de cada participante na lista), invoca-se:

```
Sage: >>> g= Z_p. multiplicative generator()
29
Sage: >>> feldmanverify (t, g, ptt, partes[t-1][1])
```

para verificar a partes distribuída a cada elemento, retornando $True$ ou $False$.

Repare que é necessário fazer $t-1$, pois, o elemento t (posição de $1 \cdots, n$), a sua parte secreta na lista (de $0, \cdots, n-1$) está na posição $t-1$.

Portanto, outra função não menos importante, é a verificação da unicidade do polinómio de Lagrange, isto é, a função que permite verificar se os polinómios, construído e o reconstruído são iguais. Função esta, que recebe os dois polinómios e compara-os.

`Invocado >>> verifypols(polconst, polreconst)`

Temos a confirmação de **True** se $P_1(x) = P_2(x)$, caso contrário, imprime **False**.

4.3 Prova de conhecimento zero

Numa eleição eletrónica, é fundamental que o voto seja confidencial. A grande problemática reside no facto de, às vezes ser necessário provar ao destinatário que o voto emitido e enviado pelo eleitor foi formado corretamente, sem revelar para o destinatário qualquer informação inerente ao voto. Para este caso, é necessário aplicar a prova de conhecimento zero (nulo) "proof zero-knowlodge"(Shoenmakers, 2018).

Definição 9: Prova de conhecimento zero é um método interativo que possibilita a uma parte provar para outra que uma declaração (informação) é verdadeira, sem revelar qualquer informação da mesma (Shoenmakers, 2018).

Exemplo 14:

Paula (P) descobriu a chave secreta da porta de uma caverna. A caverna tem a entrada por um lado e uma porta bloqueando o lado oposto, isto é, se a pessoa entrar por uma porta não consegue ter acesso ao outro lado. Veigas (V) diz que vai pagar-lhe pelo segredo, mas só se for confirmado que Paula realmente sabe o segredo. Paula diz que vai contar o segredo, mas, só quando ele (V) lhe entregar o dinheiro.

P pode provar a V que conhece a chave secreta da porta da caverna, sem que V saiba da chave antes de pagar.

- Sejam A e B os dois caminhos (esquerda/ direita) da caverna, V espera fora da caverna (atento nas saídas de A e B), enquanto P entra na caverna pelo caminho A ou B aleatoriamente.

- V grita o nome do caminho A ou B, pedindo que P entre ou retorne pelo respetivo caminho.

- Uma vez que P conhece o segredo, abre a porta (se necessário) que bloqueia o acesso do lado oposto, e retorna ao longo do caminho desejado.

- Se P não soubesse a chave secreta e, visto que V solicita o caminho aleatoriamente, P teria uma probabilidade de 50% para acertar caso o procedimento se repita várias vezes. Isto é, P só conseguiria retornar pelo caminho solicitado (desejado) por

V, se fosse o mesmo escolhido para entrar. Portanto, sendo o segredo conhecido por P, ele sai pelo caminho escolhido por V, quantas vezes ele quisesse, provando que conhece a chave secreta sem revelar o segredo antes que V pague.

Ainda (Shoenmakers, 2018) na sua obra intitulada "Lecture Notes Crytographic Protocols": é possível provar para um agente cético que o voto está corretamente formado ou não. Na prática o eleitor para além de autenticar seu voto com uma assinatura cega, o voto deve ser enviado juntamente com uma prova tal que um agente autorizado possa verificar se o voto está corretamente formado, sem, no entanto, saber o conteúdo do voto.

Provar que um voto está corretamente formado, significa:

a) Verificar se todos os votos cifrados $E(m_i)$ utilizados para calcular o produto dos votos $(E(m_i) \cdot E(m_{i+1}))$ foram formados corretamente. Isto é, de acordo à propriedade homomórfica aditiva do Elgamal, e no caso de uma eleição eletrónica que adote o estilo de voto de duas escolhas "sim/ não"$(0/1)$ entre as n opções, somam-se 0's e 1's, em seguida são cifrados os resultados;

b) Cada $E(m_i)$ foi corretamente assinado;

c) Verificar se o resultado da decifragem corresponde ao respetivo criptograma, onde:

$i = 1, 2, \cdots, n$;

m_i- o voto;

$E(m_i)$- o voto cifrado;

n- o total de eleitores.

Propriedades

Sejam os dois agentes, provador e verificador, são válidas as seguintes propriedades:

a) Integralidade: se a afirmação é verdadeira, então, um verificador V será convencido deste facto por um provador.

b) Sanidade: se a afirmação é falsa, nenhum falso provador poderá convencer o verificador que ela é verdadeira.

c) Conhecimento zero: se a afirmação é verdadeira, nenhum verificador consegue obter qualquer informação, além de que a afirmação é verdadeira.

4.3.1 Esquema de identificação de Feige-Fiat-Shamir

O esquema de identificação Feige-Fiat-Shamir é um tipo de prova de conhecimento zero desenvolvido por Uriel Feige, Amos Fiat e Adi Shamir em 1988.

Tal como vimos anteriormente, um provador pode provar a um verificador que ele possui informações secretas sem revelar ao verificador o conteúdo desta informação. O esquema de identificação de Feige-Fiat-Shamir usa a aritmética modular e um processo de verificação paralela que limita o número de comunicações entre os dois, isto é, verificador e provador (Feige, Fiat, Shamir, 1988).

O algoritmo é dado da forma que se descreve de seguida:

Sejam Paula- provadora, e Veigas- verificador.

a) Inicialização:

1- Escolhe-se dois números primos p e q distintos, e calcula-se $n = p \cdot q$. Paula e Veigas recebem n.

2- Com os números secretos s_i, $i \in \mathbb{N}$, Paula calcula:

$$v_i = (s_i)^2 \mod n \tag{4.15}$$

e envia os números v_i para Veigas (comunicação).

b)Procedimento:

3- Paula escolhe um inteiro $r \in \mathbb{Z}_n^*$ calcula e envia:

$$x = r^2 \mod n \tag{4.16}$$

4- Veigas escolhe aleatoriamente os números $w_i \in \{0, 1\}$, $i \in \mathbb{N}$ isto é, $w_1, \cdots, w_i$, e envia-os para Paula.

5- Paula calcula e envia para Veigas:

$$y = r \cdot s_1^{w_1} \cdot s_2^{w_2} \cdots s_i^{w_i} \mod n \tag{4.17}$$

6- Veigas verifica se:

$$y^2 = x \cdot v_1^{w_1} \cdot v_2^{w_2} \ldots v_i^{w_i} \mod n,\ x \neq 0 \tag{4.18}$$

Exemplo 15 (anexo 7): Paula quer provar para Veigas que conhece o número secreto
$S = 210001111012239915973$, sem revelá-lo este número.

Solução:

Inicialização:

Seja n, o produto dos parâmetros p e q, o valor de n que será tornado público é obtido invocando a função que recebe 64 bits como parâmetro de entrada.

```
Sage:  >>> genPQ (64)
```

```
n= 11021574356268210131
```

Procedimento:

Prova:

Paula obtém os números secretos separando os algarismos do segredo, números que serão usados durante a comunicação com V:

$S = 2100011 - 1101223^{-} 9915973$, isto é: $s1 = 2100011$; $s2 = 1101223$; $s3 = 9915973$. Invocando a função que recebe o valor de n, e os números secretos, Paula envia a prova:

```
Sage: >>> Proof (num, s1, s2, s3)

v1= 4410046200121

v2= 1212692095729

v3= 98326520536729

x= 7135802740278375468
```

Veigas interage com Paula os números w_i para Paula, e esta por sua vez calucula y:

```
y= 4458054991700864221
```

Verificação:

Veigas recebe os elementos da prova enviados por Paula, v_1, v_2, v_3, y, x, e com a função que recebe para além dos valores da prova, os números w_i, verifica invocando:

```
Sage: >>> Verify (prova, binarios)
```

V é convencido por P verificando que realmente P conhece o segredo, retornando um **True** (verdade), caso contrário, retorna **False** (falso).

Portanto, este procedimento poderia repetir-se quantas vezes V quisesse com diferentes valores r e w_i até que ele esteja convencido de que P realmente possui raízes quadradas (s_i) modulares dos quadrados v_i, isto é, Paula prova a Veigas que conhece o segredo
$S = 210001111012239915973$, sem revelar-lhe nenhuma informação a cerca do mesmo.

Por outra, se um intruso intercetar v_i, não consegue recuperar os números secretos s_i, devido a dificuldade de calcular a raiz quadrada modular quando a fatorização do módulo é desconhecida. Logo, pela mesma razão, o verificador V não recupera nenhuma informação sobre o segredo de P.

Este fato, do cálculo da raiz quadrada modular, é abordado por Eric Bach e Klaus Huber (1999, p. 807-808), num artigo científico intitulado "Note on Taking Square- -Roots Modulo N". Os autores mostram que, dada:

$$x^2 \equiv a \mod N, \tag{4.19}$$

onde a é um número inteiro, e N um número composto não é eficiente calcular x quando a fatorização de N não é conhecida. Isto é, calcular x implica fatorizar N. Problema este, que é baseada a segurança do RSA.

4.4 Esquema criptográfico para eleição eletrónica

O presente esquema criptográfico proposto, é um procedimento resultante da combinação de técnicas criptográficas que visam garantir a confidencialidade, integridade, autenticidade e não repúdio, anonimato e verificabilidade numa eleição eletrónica.

4.4.1 Intervenientes

São intervenientes do esquema as seguintes entidades:

a) Eleitor (Alice): tem a função de eleger, ofuscar, desofuscar e cifrar o voto (mensagem).

b) Autoridade da comissão eleitoral (Bob): tem a função de escolher o par de chaves Elgamal e partilhar a chave privada.

c) Autoridade de confiança (CA): tem a função de gerar um par de chaves RSA e assinar o voto.

d) Membros da comissão eleitoral (MEC:) participam na partilha da chave privada e verificam a consistência das partes secretas recebidas.

d) Provador: tem a função de provar que os votos foram corretamente formados.

e) Responsável pelo escrutínio (RE): é um dos MEC, e tem a função de reconstruir a chave privada, decifrar os votos, verificar a autenticidade e verificar se os votos foram corretamente formados.

4.4.2 Técnicas criptográficas envolvidas

O algoritmo do esquema faz o uso das seguintes técnicas criptográficas:

a) Elgamal: é utilizada para a cifração/ decifração do voto, garantindo a confidencialidade do voto.

b) Assinatura cega: é utilizada para garantir a autenticidade, não repúdio e o anonimato do eleitor.

c) Partilha da chave secreta: o esquema de Shamir permite realizar a partilha da chave privada.

d) Prova de conhecimento zero: utilizada para a entidade provadora provar ao RE (verificador) que os votos foram corretamente formados.

e) Esquema de verificação de Feldman: usado para verificar a consistência das partes secretas da chave privada distribuídas a todos participantes envolvidos no esquema de partilha.

4.4.3 Algoritmo do esquema

Sejam Alice, Bob, CA, MEC, Provador e RE. De acordo às abordagens feitas anteriormente. Segue-se o algoritmo:

1) Seja p primo, tal que $\mathbb{Z}_p^*$ um grupo com gerador g, Bob escolhe um par de chaves Elgamal:
$x \in \mathbb{Z}_p^*$: chave privada
(p, g, g^x): chave pública.

Em seguida, Bob define n- total de participantes no esquema de partilha da chave privada, e k- total de partes secretas necessárias para reconstrução do segredo.

2) Tendo os elementos para realizar a partilha, a chave privada e $\mathbb{Z}_p^*$, Bob inicia o processo de partilha da chave:

- Escolhe os $k-1$ inteiros (coeficientes) de $\in \mathbb{Z}_p^*$ do polinómio $P(x)$ de grau $k-1$, onde a_0 é a chave privada.

- Constrói o polinómio em função dos coeficientes escolhidos, e fixa-o.

- Tendo a função polinomial $P(x)$, calcula as partes secretas $(t, P(t)_i)$, onde $t = 1, \cdots, n$.

3) Bob com os coeficientes do polinómio $P(x)$, calcula as potências $c_i = g^{a_i}$, $(i \in \mathbb{N})$, onde a_i são os coeficientes do polinómio. As partes secretas $S_t - 1$ e as potências são enviadas em simultâneo para os MEC.

4) Cada MEC verifica se: $g^{P(t)_i} = C_0^{t^0} \dots C_{b-1}^{t^{(b-1)}}$, onde b é o número de elementos necessários para recuperar a chave privada.

5) Depois de Alice ter votado, inicia o processo de assinatura cega:

- A CA gera um par de chaves RSA (secção 2.3), e com a sua chave privada assina o voto.

- Alice escolhe um inteiro $k_0 \in \mathbb{Z}_p^*$, com a chave pública de Bob, calcula: $k_0^e \cdot m$ (ofuscação),onde m- é o voto.

- Alice multiplicando o voto assinado pelo inverso de k_0 obtém a assinatura da mensagem original. Assim, Alice sabe que o voto assinado lhe pertence.

6) Alice usando a chave pública Elgamal de Bob, cifra o seu voto.

7) O responsável (RE) pelo escrutínio, tendo necessidade de decifrar o voto de Alice, em $(t, P(t)_i)$, escolhe um grupo de k_0 pessoas das n, e com as suas partes secretas $P(t)_i$ (somente as imagens de t), $i = 1, \cdots, n$, reconstrói-se $P(x)$ (polinómio interpolador de Lagrange), onde a chave privada é o termo a_0 de $P(x)$.

8) RE, usando a chave privada Elgamal reconstruída, decifra o voto de Alice.

9) RE com a chave pública da CA, verifica a autenticidade do voto. Isto é, se a potência de base assinatura digital e expoente e (chave pública RSA) for igual ao voto m, então conclui-se que o voto é autêntico.

10) Sejam p e q, dois números primos, e $n = p \cdot q$, o provador que pode ser a CA, ou Bob. No caso de ser a CA, usa a sua chave privada como segredo para realizar a prova, e no caso de ser Bob, usa os coeficientes do polinómio para realizar a prova. O provador escolhe seus números secretos s_i, calcula os quadrados modulares v_i de base s_i. Escolhe o inteiro $r \in \mathbb{Z}_n^*$, e calcula:

$x \equiv r^2 \mod n$.

11) RE escolhe os números $w_i \in \{0, 1\}$.

12) Com os números w_i, o provador calcula $y = r \cdot s_1^{w_1} \cdots s_i^{w_i} (\mod n)$. Para realizar a prova, envia-se para o verificador o seguinte: v_i, x, y. onde

$$v_i = (s_i)^2 \mod n$$

.

13) E finalmente o verificador RE efetiva a verificação comparando:

$$y^2 = r^2 \cdot (s_1^{a_1} \ldots s_i^{a_i})^2 \mod n$$
$$y^2 = x \cdot s_1^{2a_1} \ldots s_i^{2a_i} \mod n$$
$$y^2 = x \cdot v_1^{a_1} \ldots v_i^{a_i} \mod n;$$

Onde, $r^2 = x \mod n$ e $v_i = s_i^2 \mod n$.

4.4.4 Aplicação

Para fazer a aplicação do esquema, vamos adotar o estilo de eleição eletrónica, "uma escolha de N opções possíveis", onde o eleitor faz uma escolha na lista de N candidatos/ partidos concorrentes, enumerados de 1 a N.

Exemplo 16 (anexo 8): suponhamos que temos 30 eleitores, e o voto $m = 17$, isto é, Alice escolheu o concorrente número 17. Apliquemos o nosso esquema

criptográfico para dar um tratamento que garanta segurança ao voto desde a origem até ao escrutínio.

Solução:

Usemos 64 bits.

- Bob (uma entidade da comissão eleitoral) escolhe um par de chave Elgamal, partilha a chave privada, construindo o polinómio para obter as partes (a partilhar) secretas, invoca as funções.

  - Para construir o polinómio, invoca-se a função que recebe os coeficientes:

  ```
  Sage: >>> polinomioconstruido (coefpol)
  13990770389270254 7x^2 + 180288090840436691 5x+ 164593337622566279
  ```

  - Para obter as partes secretas, invoca-se a função que recebe o número total de participantes e os coeficientes do polinómio:

  ```
  Sage: >>> partespartilhadas (n, polconst)

          [1, 2107381949919635741];
          [2, 4329985970002110297];
          [3, 6832405397869989947];
          [4, 9614640233523274691];
          [5, 12676690476961964529].
  ```

- Depois da partilha, usando os coeficientes do polinómio como expoente, calcula as potências que têm o gerador g como base, e invoca-se a função que recebe o gerador e os coeficientes do polinómio:

  ```
  Sage: >>> potenciasverificacao (gerador, coefpol)
  1725094208757685000;
  1244971207896640836;
  1218387893865471964.
  ```

- Cada participante da partilha recebe a sua parte secreta, as potências e verifica se a parte secreta recebida é válida, invocando a função que tem como parâmetros de entrada: o número de ordem do participante, gerador do grupo, as potências e as partes secretas.

```
Sage: >>> verificacaopartes (t, gerador, ptt, partes[t-1][1])
```

Retornando *True* ou *False*.

- O eleitor começa por ofuscar o seu voto, envia-o para o signatário, depois da assinatura, o eleitor desofusca e obtém a assinatura invocando a função que recebe a assinatura ofuscada, fator de ofuscação e a chave pública:

```
Sage: >>> assinaturalimpa (bsig, bfactor, pk)
502310287232279324050113765 94
```

- Em seguida, cifra o seu voto, e obtém o criptograma invocando a função que recebe o voto e a chave pública:

```
Sage: >>> cifracaovoto (pkEG, m)
[8, 1556434553424576841]
```

- Com intuito de recuperar a chave privada para decifrar os votos, RE escolhe b partes secretas de modo a obter o polinómio interpolador, onde a chave privada é o termo independente do polinómio reconstruido. Então, invoca a função que recebe as partes secretas e os coeficientes construidos inicialmentemente:

```
Sage: >>> recostrucaopolinomio (coefpol, partes)
139907703892702547x^2 + 180288090840436691 5x+ 164593337622566279
```

- RE aplica a prova de conhecimento zero para saber se os mesmos estão corretamente formados. Neste caso, como prova, o provador (CA/ Bob) envia $v_i, \cdots, v_k$ e (y, x). O RE interage com o provador, enviando a_i. Para verificar, invoca a função que recebe os elementos da prova e os números w_i escolhidos por ele aleatoriamente.

```
Sage: >>> Verificacaoproof (prova, binarios)
```

Então, ele é convencido, e diz *True*.

- RE decifra o voto de Alice, invocando a função que recebe a chave pública, privada e o texto cifrado:

```
Sage: >>> Decifracaovoto (pkEG, skEG, Criptograma)
17
```

- RE abre a urna eletrónica, e confirma que a integridade e confidencialidade do voto foi preservada, em seguida, verifica a autenticidade, para tal, invoca a função que recebe o voto, chave pública e a assinatura.

  ```
  Sage: >>> verificacaoassinatura (m, pk, sigmes)
  ```

  Retorna True, confirmando que não houve anomalias no processo do tratamento do voto.

4.4.5 Segurança

A segurança do esquema, baseia-se intrinsecamente em:

a) Dificuldade da fatorização de inteiros: desvendar a chave privada (RSA) do signatário, pois conjentura-se que quebrar o sistema RSA implique fatorizar $n = p \cdot q$.

b) Problema do logaritmo discreto: a segurança da chave privada usada para decifrar, baseia-se no problema do logaritmo discreto, e na impossibilidade da reconstrução da chave de forma isolada (uma pessoa), isto é, sem o acordo mútuo de k partes secretas, tal que, para $1 < k \leq n$ não seja possível recuperar o segredo (desejado). A partir da potência (partilhada) onde o expoente é a chave privada a_0, não eficiente calcular a_0 (expoente), também pelo problema do logaritmo discreto. Portanto, é difícil reconstruir a chave privada com um número de segredos menor ou maior que k.

c) Ofuscação: nem o signatário, nem qualquer outra entidade consegue desvendar o anonimato de Alice, quer dizer, não é possível saber quem votou a quem, pois o voto não é assinado pelo seu respetivo proprietário, e por outra, o signatário não consegue ter acesso ao conteúdo do voto por não conhecer a constante (secreta) de ofuscação. Portanto, o voto assinado revela somente a identidade da autoridade de confiança.

d) Conhecimento zero: nenhuma entidade com propósito de realizar a verificação, consegue saber o segredo conhecido pelo provador, para além de ser convencido que realmente o provador conhece o segredo (chave privada RSA, ou coeficientes do polinómio), pela dificuldade de calcular a raiz quadrada modular quando a fatorização do módulo é desconhecida. Sendo o módulo, o produto de dois números primos p e q de grande tamanho, distintos e distantes um do outro.

Portanto, uma vez que no esquema se inclui a chave pública para assinar o voto, deve-se incluir também a certificação digital para garantir a associação entre chave

pública e identidade. Isto é, para garantir que o voto foi assinado por uma entidade autorizada, ou provém de uma fonte fidedigna. O Elgamal garantiu a integridade e a confidencialidade; a assinatura cega permitiu garantir a autenticidade, não repúdio e o anonimato e a prova de conhecimento zero e o esquema de Feldman, garantiram a verificabilidade do processo.

Capítulo 5

Conclusões

Portanto, no âmbito dos objetivos preconizados inicialmente, e de acordo com as teorias abordadas no trabalho, o tratamento matemático realizado, e os resultados obtidos, culminámos este estudo com a construção de um esquema criptográfico que visa a garantia da integridade, confidencialidade, autenticidade, anonimato, não repúdio e verificabilidade de uma eleição eletrónica. Temos em seguida algumas conclusões.

1) A aplicação do esquema criptográfico proposto numa eleição eletrónica permite garantir o anonimato do eleitor, integridade, confidencialidade, autenticidade e verificabilidade do voto eletrónico.

2)É possível verificar a autenticidade e não repúdio de um voto eletrónico, sem revelar a identidade do eleitor.

3) A segurança do esquema criptográfico proposto baseia-se fundamentalmente no problema da fatorização de números inteiros em primos, logaritmo discreto, assinatura cega, partilha do segredo e na dificuldade de calcular a raiz quadrada modular quando a fatorização do módulo é desconhecida.

4) É necessário que a partilha da chave privada seja verificável, visto que, erros pela partilha ou ações mal-intencionadas podem ocorrer durante o processo. Logo, cada participante deve estar habilitado para verificar a consistência da parte secreta que lhe foi atribuída.

5) O esquema de identificação de Feige-Fiat-Shamir é viável para provar se os votos estão corretamente formados, na medida em que, a partir dos valores da prova $v_1, v_2, \cdots, v_i$, quer seja o verificador, ou um intruso qualquer, não é eficiente descobrir o segredo (chave privada RSA, ou coeficientes do polinómio) pela dificuldade de calcular a raiz quadrada modular quando a fatorização do módulo é desconhecida.

5.1 Recomendações

1) O uso do RSA para cifração/ decifração, ou assinatura digital, continua ser recomendável, pois, apesar da segurança do RSA centrar-se na dificuldade de fatorizar números inteiros de grande em números primos, não se conhece solução em tempo polinomial para a factorização de números com tamanho grande em números primos. A geração de chaves é realizada em tempo polinomial, mas a quebra da chave por força bruta exige tempo exponencial, e este fato é o que suporta a segurança do algoritmo do RSA (Masthanamma e Preya, 2015).

2) A segurança da assinatura cega reside na garantia da cegueira da mensagem antes do signatário assiná-la. Mas, está sujeita ao ataque man-in-the-middle, e por isso deve-se incluir certificado digital para garantir a associação da chave pública à identidade.

3) A partilha da chave secreta verificável constitui uma das ferramentas para o armazenamento de uma chave secreta. O esquema pode ser utilizado para o armazenamento de informações com acesso exclusivo, por exemplo, a chave secreta de um cofre bancário.

(4) Os códigos apresentados em anexos não contêm o princípio de identação, recomenda-se ao leitor interessado em implementar os respetivos códigos, a ter em conta este aspecto, ou consultar o repositório github (*https://github.com/Agente2019/Implementation-E-voting-Cripto*).

Capítulo 6

Referências bibliográficas

Allen, B. (2008). Implementing Several Attacks Plain ElGamal Encrypt. Graduate Theses and Dissertations. 11535. Iowa State University: Digital Repository.

Alves, P. (2014). Aplicação Conceitual de Criptografia Homomórfica, MO422-Algoritmos Criptográficos. Campinas: Brasil.

Bach, E., Huber, K. (1999). Note on Taking Square-Roots Modulo N: Transations Information Theory, vol. 45. Nº 2, Member, IEEE.

Boneh, D., Shoup, V. (2017). Graduate Course in Applied Cryptography.

Burmester, M., Desmedt, Y., Beth, T. (1991). Efficient Zero-Knowlodge Identification Schemes for Smart Cards. Department of Mathematics, University of London-RHBNC, Egham, Surrey TW20 OEX.

Capello, A., Leal, I. (2007). Teoria Aritmética dos Números e Criptografia RSA. RA 061533.

Cerqueira, M. (2016). O Estudo da Criptorafia RSA no Ensino Básico com Auxílio de Softwares Computacionais.Universidade Federal de Alagoas: Brasil.

Cramer, R., Gennaro, R., Shoenmakers, B. (1997). In Advances in Cryptology-EUROCRYPT"97, Vol.1233 of Lecture Notes in Computer Science, Springer-Verlag. Damgård, I., Groth, J. e Salomonsen, G. (2002). The Theory and Implementation of an Eletronic Voting System.

Brosenne, H. (2014). University of Goettingen: Institut of Computer Science.

Diffie, W., Hellman, M. (1976), New Directions in Cryptography. Invited Paper. Transations in Iformation Theory, Vol. It-22, Nº 6.

Espuelas, A., José, G. (2017). Estudio y Análisis de Esquemas de Votación Electrónica Mediante Protocolos de Firmas Digitales Anónimas. Universidad Autónoma de Madrid: Espanha.

Feige, U., Fiat, A., Shamir, A. (1988). Zero-knowledge Proofs of Identily. The Weizmann Institute of Science: Israel.

Feldman, P. (1987). A Pratical Sheme for Non-Interactive Verifiable Secret Sharing. Fiat, A. e Shamir, A. (1987). How to prove youself:pratical solutions to identiication and signature problems. Departament of Applied Mathematics the Weizmann Institute of Science Rehovot 76100, Israel.

Giacomelli, I. (2014). Verifiable Secret-Sharing Shemes. AARHUS UNIVERSITY: Aalborg. Goldwasser, S. e Bellare, M. (2008). Lecture Notes on Cryptography. MIT Computer Science and Artificial Intelligence Laboratory, The Stata Center, Building 32, 32 Vassar Street, Cambridge, MA 02139, USA.

Hashim, H. (2014). the Discrete Logarithm Problem in the ElGamal Cryptosystem over the Abelian Group U(n) whwre $n = p^m$, or $2p^m$. Assistant Lecturer Department of Mathematics, Faculty of Mathematics e Computer Science, University of Kufa, Iraq.

Håstad, J. (2006). Lecture 7: Elgamal and Discrete Logarithms. Transcribed by Johan Linde. KTH NADA. 2D1449 Foundation of Cryptography.

Kaw, A., Keteltas, M. (2009). Lagrangian Interpolation: Textbook notes on the Lagrangian method of interpolation. General Engineering.

Martins, A. (2005). Elementos de Criptologia: uma aplicação da Álgebra.UMINHO. Braga: Portugal.

Masthanamma, V., Preya, G. (2015). Efficient Data Security in Cloud Computing Using the RSA Encryption Process Algorithm. Vol. 4. International Journal of Innovative Research in Science, Engineering and Technology (An ISO 3297: 2007 Certified Organization): UG Scholar, Department of Information Technology, Saveetha School of Engineering. Saveetha University, Chennai, Tamil Nadu, India.

Meier, A. (2005). Joint Advanced Students Seminar 2005.

Mortensen, M. (2007). Secret Sharing and Secure Multy-Party Computation. N-5020 Bergen.

Rabin, T., Ben-Or, M. (1989). Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstrat). Jerusalem. Israel.

Schoenmakers, B. (2000). Fully Auditable Electronic Secret-Ballot.

Shoenmakers, B. (2010). "Voting Shemes" in Algorithms and Theory pf Computations Theory (Second Edition), Mikhail Atallah and Marina Blanton (Eds), Chapman e Hall/ CRC.

Shoenmakers, B. (2018). Lecture Notes Cryptographic Protocols. Eindhoven: Netherlands.

Stadler, M. (1996). Publicly Verifiable Secret Sharing. In Advances Cryptography-EUROCRYPT"96, volume 1070 of Lecture Notes in Computer Science Zurich: Suça.

Stein, W. (2017). Elementary Number Theory: Primes, Congruences, and Se-

crets.

Capítulo 7

ANEXOS

7.1 ANEXO 1

```
    # Criptossistema RSA
# Geração de chaves

def RSA (bits):
    """ Função que invoca o cálculo do par de chaves RSA.
Input: recebe o número de bits.
Output: retorna um par de chaves.
Uso: escolhe-se dois números primos p e q, calcula-se a função de Euler φ(n) = (p− 1),
isto é, d é o inverso multiplicativo de e    mod  n, Z_n é o grupo, tal que n é o produto
de dois números primos p e q, e ambos de grande tamanho.
d- chave privada e (n, e)- chave pública.
Exemplo: Seja 128 o número de bits, suponhamos p = 6076863405864212999 e
q = 1562809530454321421, então, o par de chaves é:
chave privada
d = 491397063228403561872764881059505611 1,
chave pública:
(n, e) = (9496980045953699178562209452752351579, 4042594120986213764416215937187774 31).
Atenção: Se os parâmetros p e q forem de tamanhos pequenos, um ataque por força
é trivial. A NIST recomenda no mínimo 3072 bits.
    """


p, q= random_prime (2**(bits/2)), random_prime (2**(bits/2))
n= p* q
Z_n= IntegerModRing(n)
```

```
phi= (p-1)* (q-1)
e= ZZ.random_element (phi)
```

while

```
 gcd(e, phi) != 1:
e= ZZ. random_element (phi)
d= power_mod (e, -1, phi)
Chave privada= d
Chave publica= (n, e)
```

return (Chave privada, Chave publica)

```
    # Cifração
def Encrypt(M, Chave publica):
    """ Função que invoca a cifração do texto puro.
Input: mensagem (texto puro) a enviar e a chave pública.
Output: mensagem cifrada.
Uso: o criptograma C é calculado, efetuando: C = M^e (mod n)
Exemplo : SejaM = 112, a mensagem a cifrar, o criptograma é:
780352686748774311441387705090707592 1.
    """
```

```
n, e= Chave publica
Criptograma= power_mod (M, e, n)
```

return Criptograma

Decifração

```
def Decrypt (Criptograma, sk, pk):
    """
    Função que invoca a decifração do texto puro.
Input: criptograma e o par de chaves.
Output: texto puro.
Uso: o texto puro é recuperado, efetuando: M = C^d = [(M^e)]^d (mod n)
    """
```

```
d= sk
(n,_) = pk
Texto puro= power_mod (Criptograma, d, n)
```

```
return Texto puro
```

7.2 ANEXO 2

```
    # Criptossistema Elgamal
# Geração de chaves

def Elgamal (bits):
    """
    Função que invoca o cálculo das chaves, privada e pública Elgamal.
Input: recebe o número de bits.
Output: retorna a as duas chaves como parâmetros de saída.
Uso: Alice calcula o p primo, onde Z_p^* é grupo, escolhe a chave privada x ∈ Z_p^*, g
gerador do grupo, calcula y = g^x mod p. A chave a pública é: (y ; g ; p).
Exemplo: Suponhamos que pretende-se gerar uma chave de 128 bits, então, a chave
privada: 190422571672400112311789107296726056505,
Chave pública: (22475769130059677661640730068541036135 3,
65773925495537165919969078067390052602)
    """


p= random_prime (2**bits)
Z_p= IntegerModRing(p)
g= Z_p. multiplicative_generator()
x= randint(2, p- 1)
y= g**x
PuKey= (p, g, y)
PrKey= x

return PuKey, PrKey


    # Cifração
def Encrypt (PuKey, m):
    """
    Função que invoca a cifração do texto puro.
Input: a chave pública e o texto puro.
Output: retorna o criptograma.
Uso: Bob calcula o criptograma (α ; β), efetuando, α = g^k mod p, e
β = M . y^k mod p, onde k é um inteiro pertecente ao grupo.
Exemplo: Suponhamos que queremos cifrar M = 11002, então, o criptograma
```

resultante é:
[2186333659918013616281193712444930365 81, 655619131077072917206333566472546844 60]
"""

```
p, g, y= PuKey
Z_p= IntegerModRing(p)
k= randint(2, p- 1)
Alfa= g**k
Beta= Mensagem*(y**k)
Criptograma= (Alfa, Beta)
```

return Criptograma
Decifração

def Decrypt(PuKey, PrKey, Criptograma):
"""
Função que invoca a decifração do texto cifrado.
Input: a chave pública, privada e o criptograma.
Output: retorma a mensagem original.
Uso: Alice decifra a mensagem: Texto puro = M . α^{-x} . α^{x} mod p, sendo, $\alpha = g^{k}$ enviado por Bob.
"""

```
p, g, y= PuKey
x= PrKey
Z_p= IntegerModRing(p)
s= (Criptograma[0]**x)
Mensagem= (1/s)*Criptograma[1]
```

return Mensagem

7.3 ANEXO 3

```
    # Verificação da propriedade multiplicativaElgamal
# Geração de chaves

def Elgamalmultprop (bits):
    """ Função que invoca o cálculo das chaves, privada e pública Elgamal.
Input: recebe o número de bits.
Output: retorna a as duas chaves como parâmetros de saída.
Uso: Alice calcula o p primo, onde Z_p^* é grupo, escolhe a chave privada x ∈ Z_p^*, g
gerador do grupo, calcula y = g^x mod p. A chave pública é: (p ; g ; y).
Exemplo: Suponhamos que, pretende-se gerar uma chave de 128 bits, então,
a chave privada é:125803504335772840918183276333021836l3,
chave pública:
(117975757404720837759613660827752801947 , 2 , 10153998274581155281795044029414688548 9)
    """


    p= random _prime (2**bits)
    Z_p= IntegerModRing(p)
    g= Z_p. multiplicative_generator()
    x= randint(2, p- 1)
    y= g**x
    PuKey= (p, g, y)
    PrKey= x

    return PuKey, PrKey
# Cifração
    def Encrypt1 (PuKey, Mensagem1):
    """ Função que invoca a cifração do primeiro texto puro.
Input: a chave pública e o 1º texto puro.
Output: retorna o o 1º criptograma e o respetivo inteiro k_1 que é usado para calcular
o criptograma do produto.
Uso: Bob calcula o criptograma (α; β), efetuando, α = g^{k1} mod p e β = M . y^{k1} mod p,
onde k_1 é um inteiro pertecente ao grupo.
Exemplo: Seja a Mensagem1 = 22110, então o criptograma1 é:
[822075750300803973786448407417124582 86, 19598283097754323803422926241038401477]
    """
```

```
p, g, y= PuKey
Z_p= IntegerModRing(p)
k1= randint(2, p- 1)
Alfa1= g**k1
Beta1= Mensagem_1*(y**k_1)
Criptograma1= (Alfa_1, Beta_1)
```

return Criptograma1 , k_1

```
    # Segundo criptograma
def Encrypt2 (PuKey, Mensagem2):
    """
```

Função que invoca a cifração da segunda mensagem.
É análogo ao Decrypt1.
Examplo: Seja Mensagem2 = 112277, então o criptograma2 é:
[660921442295453614481192227030925832 40, 280236265034895311285878552115486 4222]

```
    """
```

```
p, g, y= PuKey
Z_p=IntegerModRing(p)
k2= randint(2, p- 1)
Alfa2= g**k2
Beta2= Mensagem 2*(y**k2)
Criptograma2= (Alfa2, Beta2)
```

return Criptograma2, k2

```
    # Criptograma do produto dos textos claros
def Encrypt3 (PuKey, Mensagem, k1, k2):
    """
```

Função que invoca a cifração do produto das mensagens.
Input: a chave pública o produto das mensagens e os inteiros k1, k2.
Output: retorna o o criptograma do produto das mensagens.
Uso: Bob calcula o criptograma $(\alpha; \beta)$, efetuando, $k = k1 + k2, M = M1.M2$, $\alpha = g^{k1} \ mod \ p$, e $\beta = M.y^k 1 mod p$.
Exemplo: Sejam as mensagens: Mensagem1 = 22110 e Mensagem2 = 112277, multiplicando as mensagens, o criptograma do produto é:
[10235679208102051508838505733599120 4787, 3637146108662286672300093493703954 7360]

```
    """
```

```
p, g, y= PuKey
```

```
k= k1+ k2
Alfa3= g**k
Beta3= Mensagem*(y**k)
Criptograma3= (Alfa3, Beta3)

return Criptograma3
# produto dos criptogramas
def Encrypt4 (Criptograma1, Criptograma2):
    """
    Função que invoca o cálculo do produto de criptogramas.
Input: recebe os criptogramas a serem multiplicados.
Output: retorna o produto dos criptogramas.
Uso: Calcula-se os produtos dos componentes dos criptogramas. Isto é, Sendo
C1 = [Alfa1 , Beta1] e C2 = [Alfa2 , Beta2]: C = [Alfa1 . Alfa2, Beta2 . Beta2]
. Exemplo: Seja os dois criptogramas:
C1 = [822075750300803973786448407417124582 86, 19598283097754323803422926241038401477]
e C2 = [660921442295453614481192227030925832 40, 728023626503489531128587855211548642 22]
O produto dos criptogramas é:
[102356792081020515088385057335991204787, 363714610866228667230009349370395473 60]
Atenção: Cuidado ao achar o produto, multiplicar diretamente C1 . C2 é diferente
de multiplicar componente por componente dos criptogramas, porque o produto é
vetorial, e não interno, pois, o criptograma é representado como um vetor de dois
parâmetros.
    """

(Alfa1, Beta1)= Criptograma1
(Alfa2, Beta2)= Criptograma2
Alfa4= Alfa1*Alfa2
Beta4= Beta1*Beta2
Criptograma4= (Alfa4, Beta4)

return Criptograma4
# Verificação da propriedade

def Equal (Criptograma3, Criptograma4):
    """
    Função que invoca a verificação da propriedade aditiva do Elgamal.
Input: recebe o criptograma do produto das mensagens e o produto dos criptogra-
mas.
```

Output: a função retorna valores lógicos. True, se for verdade e False, se for falso.
Uso: Verifica-se somente se a condição: $C_3 == C_4$ acontece.
“““

if Criptograma3== Criptograma4:
return True
else:
return False

““““ Função que invoca a verificação da propriedade aditiva do Elgamal.
Input: recebe o criptograma do produto das mensagens e o produto dos criptogramas.
Output: a função retorna valores lógicos. True, se for verdade e False, se for falso.
Uso: Verifica-se somente se a condição: $C_3 == C_4$ acontece.
“““

if Criptograma3== Criptograma4:
return True
else:
return False

7.4 ANEXO 4

```
    # Verificação da propriedade aditiva Elgamal
# Geração de chaves

def Elgamaladitprop (bits):
    """
    Função que invoca o cálculo das chaves, privada e pública Elgamal, cujo objetivo
principal é permitir verificar a propriedade multiplicativa homomórfica do Elgamal.
Input: recebe o número de bits.
Output: retorna a as duas chaves como parâmetros de saída.
Uso: Alice calcula o p primo, onde Z_p^* é grupo, escolhe a chave privada x ∈ Z_p^*, g
gerador do grupo, calcula y = g^x mod p.
A chave a pública é: (y; g; p).
Exemplo: Suponhamos que pretende-se gerar uma chave de 128 bits, então:
a vhave privada é: 41390711041409397171247429663529836471,
Chave pública: (114945940316038462204978194754054439339, 2,
92173770494054960342624284607689848 32)
    """

    p= random_prime (2**bits)
    Z_p= IntegerModRing(p)
    g= Z_p. multiplicative_generator()
    x= randint(2, p -1)
    y= g**x
    PuKey= (p, g, y)
    PrKey= x

    return PuKey, PrKey
# Cifração
    def Encrypt1 (PuKey, Mensagem1):
    """
    Função que invoca a cifração do primeiro texto puro.
Input: a chave pública e o 1º texto puro.
Output: retorna o o 1º criptograma e o respetivo inteiro k1 que é usado para calcular
o criptograma do produto.
Uso: A mensagem é representada na forma exponencial.
Bob calcula o criptograma (α ;β), efetuando, α = g^k1 mod p, e β = g^M . y^k1 mod p,
onde k1 é um inteiro pertecente ao grupo.
```

Examplo: Seja Mensagem1 = 22110, então o Criptograma1 é:
[97549997930649425488484075819228841 99, 23767648256769374292261276579892055 8]
“““

```
p, g, y= PuKey
Z_p=  IntegerModRing(p)
k1= randint(2, p- 1)
Alfa1= g**k_1
Beta1= (g**Mensagem1)*(y**k1)
Criptograma1= (Alfa1, Beta1)
```

return Criptograma1, k1
# Segundo criptograma
def Encrypt2 (PuKey, Mensagem2):
“““ Função que invoca cifração da segunda mensagem.
É análogo ao Decrypt1.
Examplo: Seja Mensagem2 = 112277, então o Criptograma2 é:
[10214436313214439332522057435537949264 4, 90099834863259559715252233316499138 35]
“““

```
p, g, y= PuKey
Z_p= IntegerModRing(p)
k_2= randint(2, p- 1)
Alfa_2= g**k_2
Beta_2= (g**Mensagem2)*(y**k_2)
Criptograma2= (Alfa_2, Beta_2)
```

return Criptograma2, k2
Criptograma do produto dos textos claros

def Encrypt3 (PuKey, Mensagem, k1, k2):
“““
Função que invoca a cifração do produto das mensagens.
Input: a chave pública o produto das mensagens e os inteiros k1, k2.
Output: retorna o o criptograma da soma das mensagens.
Uso: Bob calcula o criptograma $(\alpha; \beta)$, efetuando, $k = k_1 + k_2$, $M = M_1 + M_2$ e $\alpha = g^{k_1} mod\ p$, e $\beta = g^M . y^k \ mod\ p$, onde k também é um inteiro pertecente ao grupo.

Exemplo: Sejam as mensagens: Mensagem1 = 22110 e Mensagem2 = 112277, multiplicando as mensagens, o criptograma do produto ica:
[85727612272791696525782194249328670 27, 513978390185355451421008275426606516 96]

"""

```
p, g, y= PuKey
k= k_1+ k_2
Alfa_3= g**k
Beta_3=(g**Mensagem)*(y**k)
Criptograma_3= (Alfa_3, Beta_3)
```

return Criptograma3
# produto dos criptogramas
def Encrypt4 (Criptogram1, Criptograma2):

"""

Função que invoca o cálculo do produto de criptogramas.
Input: recebe os criptogramas a serem multiplicados.
Output: retorna o produto dos criptograma.
Uso: Calcula-se os produtos dos componentes dos criptogramas. Isto é, sendo
$C_1 = [Alfa_1 \ , \ Beta_1]$ e $C_2 = [Alfa_2 \ , \ Beta_2] : C = [Alfa_1.Alfa_2 \ , \ Beta_1 \ . \ Beta_2]$
Exemplo: Seja os dois criptogramas:
$C_1 = [97549997930649425488484075819228841 99,$
23767648256769374292261276579892055 8] e $C_2 = [10214436313214439332522057435537949 2644,$
90099834863259559715252233316499138 35] O produto dos criptogramas é:
[85727612272791696525782194249328670 27,
513978390185355451421008275426606516 96]
Atenção: cuidado ao achar o produto, multiplicar diretamente $C_1 \ . \ C_2$ é diferente em multiplicar componente por componente dos criptogramas, porque o produto é vetorial, e não interno, pois, o criptograma é representado como um vetor de dois parâmetros.

"""

```
(Alfa_1, Beta_1)= Criptograma1
(Alfa_2, Beta_2)= Criptograma2
Alfa_4= Alfa_1*Alfa_2
Beta_4= Beta_1*Beta_2
Criptograma_4= (Alfa_4, Beta_4)
```

return $Criptograma_4$

Verificação da propriedade

def Equal (Criptograma3, Criptograma4):

“““

Função que invoca a verificação da propriedade aditiva do Elgamal.

Input: recebe o criptograma do produto das mensagens e o produto dos criptogramas.

Output: a função retorna valores lógicos. True, se for verdade e False, se for falso.

Uso: Verifica-se se a condição: $C_3 == C_4$ acontece.

“““

if Criptograma3== Criptograma4:

return True

else:

return False

7.5 ANEXO 5

```
    # Assinatura cega
def BlindSigRSA (bits):

    """ Função que invoca a geração do par de chaves RSA.
Input: número de bits.
Output: chave privada e pública.
Uso: escolhe-se dois números primos p e q, calcula-se a função de Carmichael
φ(n) = (p-1), isto é, d é o inverso multiplicativo de e mod n, Z_n é o grupo, tal
que n é o produto de dois números primos p e q e ambos de grande tamanho.
d- chave privada e (n , e)- chave pública.
Exemplo: Seja 128 o número de bits, suponhamos p = 6076863405864212999 e
q = 1562809530454321421, então, o par de chaves é:
chave privada d=491397063228403561872764881059505611,
Chave pública:
(n , e) = (9496980045953699178562209452752351579, 4042594120986213764416215937187774 31).
Atenção: Se os parâmetros p e q forem de tamanhos pequenos, um ataque por força
bruta é trivial. A NIST recomenda no mínimo 3072 bits.
    """


p, q= next_prime (2**(bits/2)), next_prime (2**(bits))
n= p*q
phi= (p- 1)*(q- 1)
e= ZZ. random_element(phi)

while gcd(e, phi) != 1:

e= ZZ. random_element(phi)
d= power_mod (e, -1, phi)
Chave privada= d
Chave publica= (n, e)

return Chave privada, Chave publica
# Ofuscação

def Blind (m, Chave publica):
    """
    Função que invoca a ofuscação da mensagem antes da assinatura.
Input: mensagem e chave pública.
```

Output: constante de ofuscação e a mensagem ofuscada.
Uso: escolhe-se k pertencente a grupoZ_n, e com k e a chave pública, o emissor calcula $(k^{**}e) * m$ (ofusca), e envia ao signatário a mensagem ofuscada m0.
Exemplo: suponhamos que queremos ofuscar m= 70998000111870, então, m0 = 7173497316937668328273499868468851149947609388261382141 08

"""

```
n, e= Chave publica
Z_n= IntegerModRing(n)
k_0= Z_n(randint(2, n- 1))
m0= (k_0**e)*m
```

return (k , m0)
Momento da assinatura

def BlindSign (m0, Chave privada, Chave publica):

```
(n, e)= Chave publica
d= Chave privada
sig= m0**d
```

return sig
Desofuscação

def UnBlind (sig, bfactor, Chave publica):

"""

Função que invoca a desofuscação do texto ofuscado.
Input: mensagem ofuscada ofuscada, factor de ofuscação e a chave pública.
Output: texto claro assinado.
Uso: O emissor multiplica-se o texto ofuscado assinado pelo inverso da constante de ofuscação.

"""

```
(n, _)= Chave_publica
sigm= sig*(1/bfactor)
```

return sigm
verificação da autenticidade

def verify (m, Chave publica, sigm):

“““

Função que invoca a verificação da autenticidade.

Função que invoca a verificação da autenticidade. Input: recebe a mensagem, a chave pública e a mensagem assinada. Output: retorna dois valores lógicos: True-caso a auteticidade for válida e False-caso a autenticidade for inválida.

Atenção: caso a autenticidade não for confirmada, o recetor rejeita a mensagem, isto, a mensagem foi alvo de ataque.

“““

```
(n, e)= Chave publica
if sigm**e == m:
```

return True

else:

return False

7.6 ANEXO 6

```
    # Esquema de partilha de chaves de Shamir
# Inicialização
def main (bits):
    """
    Função principal que invoca as outras.
Input: número de bits no primo p.
Output: retorna os parametros n, b, g, p, a0, onde n-total de participantes ao esquema, b-elementos necessários para reconstruir o segredo, g-gerador e a0 o segredo.
    """

p= random_prime (2**bits)
Zp= IntegerModRing(p)
g= Zp. multiplicative_generator()

a0 = input("Digite o segredo a ser compartilhado:")
n = input("Quantos participantes terá o esquema¿')
k = input("Quantos deles serão necessários para recuperar o segredo¿')
return (n, b, g, p, a0)
 # Coeficientes do polinómio

def Coeficientes (p, a0, n, b):
    """
    Função que invoca a escolha dos coeficientes.
Input: recebe o primo p, o segredo, o número de participantes e o número de elementos para recuperar.
Output: retorna os coeficientes do polinómio interpolador.
Uso: o parâmetro a, corresponde aos b − 1 números aleatórios.
    """

Z_p= IntegerModRing(p)
t= b- 1
coef= [a0]

for  r  in range (1 ,  t  +  1):


a= Z_p. random_element()
coef. append(a)
```

return coef
Construção do polinómio def polinomio(coef):

"""
Função que invoca a geração do polinómio interpolador de lagrange.
Input: recebe os coeficientes.
Output: retorna o polinómio.
Exemplo: suponhamos que temos o segredo 2000, k = 2.
Escolhemos k - 1 = 1 coeficiente [2000, 167497848475266771127746087824805103 92],
então o polinómio é:
P(x)= 167497848475266771127746087824805103 92*x + 2000
"""

```
Z_p= coef[0]. parent()
Pol. <x> = PolynomialRing(Z_p)
polinomio= Pol(coef)
```

return polinomio
Partes à partilhar

def shared (n, polinomio):
"""
Função que invoca o cálculo das partes secretas.
Input: recebe n e o polinómio interpolador de Lagrange.
Output: retorna pares ordenados (1, polinomio(1)),...,(n, polinomio(n)).
Uso: calcula-se as as imagens dos n números na função polinómial, e retorna-se os pares ordenados secretos, e são distribuidos.
Exemplo: Seja $bits = 128$, suponhamos que queremos compartilhar 2000, as partes secretas compartilhadas são:
[[1, 13762403652574382957828797503551451 7986],
[2, 27524807305148765915657595007102903 3972] ,
[3, 41287210957723148873486392510654354 9958]]
"""

```
pares= []
```

for x in range (1, n+1):
pares.append ([x, polinomio(x)])
return pares

```
 # Recuperação do polinómio

def polrec (coef, pares):
    """
    Função que invoca a recuperação do segredo compartihado.
Input: recebe k e as partes secretas compartilhadas.
Output: retorna o polinómio interpolador.
Uso: calcula-se os polinómios de Langrange l_j e em seguida, calcula o polinómio
interpolador pol.
então, a0 de pol é o segredo reconstruido.
    """
    Z_p= coef[0]. parent()
Pol. <x>= PolynomialRing(Z_p)
b= len(coef)
pol= Pol(0)
for j in range(k):
l_j = Pol(Z_p(1))
for i in range(k):
if j != i:
l_j =  l_j * ((x-pares[i][0])/(pares[j][0]- pares[i][0]))
pol= pol+ l_j * pares[j][1]
return pol
 # Cálculo das potências

i = 0
pot= [ ]
def potencia (g, coef):
    """
    Função que invoca o cálculo das potências (parâmetros públicos) para a verifi-
cação.
Input: recebe o gerador e os coeficientes do polinómio.
Output: retorna as potências.
Uso: calcula-se e distribui-se as b potencias a todos os participantes.
    """

pot= [ ]
k= len(coef)

for i in range (k):
```

```
pot.append (g**coef [i])
return pot
 # Verificação da consistência das partes secretas

def feldmanverify (i, g, pot):
    """
    Função que invoca a verificação da consistência das partes secretas de cada interveniente.
Input: recebe i-posição de cada participante, o gerador e as potencias.
Output: retorna dois valores, True ou False, caso positivo ou negativo, respetivamente.
Atenção: Se o resultado for False, implica dizer que, o responsável pela partilha distribuiu maliciosamente a parte secreta ou por engano assim, a fez.
    """

prod= 1
k= len(pot)

for j in range (b):
prod  = prod * (pot[j]**(i**j))
if g** secret== prod: return True
else:
return False
 # Verificar a unicidade do polinomio interpolador de Lagrange

def verifypols (polc, polr):
    """
Função que invoca a verificação do teorema da unicidade do polinómio interpolador de Lagrange.
Input: o polinómio construído e o reconstruído.
Output: True ou False
    """
    if polc== polr:
return True
else:
return False
```

7.7 ANEXO 7

```
    # Prova de conhecimento zero
# Geração de chaves
def genPQ(bits=32):
    """
Função que invoca a inicialização da prova de conhecimento zero. Input: recebe o
número de bits. Output: retorna o parâmetro público n.
Uso: escolhe-se p e q, númros e calcula-se n= p . q.
    """


p, q=  random_prime (2**(bits/2)),
random_prime (2**(bits/2))
n = p*q

return n
# Interação entre Paula e Veigas
def Comunication (n, s_1, s_2, s_3) :
    """
    Função que invoca a interação entre o provador e o verificador.
Input: parâmetro público n e números secretos s1, s2, s3, ..., si.
Onde k é um inteiro positivo.
Output: retorna r*(s1**a1)*(s2**a2)*(s3**a3), s*r**2 e a lista de números binários,
onde n pertence ao grupo Zn e r um número inteiro.
Uso: seja Zn, ecolhe-se um número inteiro r. Calcula-se em seguida x=r**2, escolhe-
se também os números binários w1, w2, w3, ..., wi e calcula-se y = r.(s_1^a1).(s_2^a2).(s_3^a3).
Atenção: a quantidade de números secretos e de números binários devem ser iguais.
    """

Z_n= IntegerModRing(n)
r= Z_n. random_element()
x= s*r**2
w1= randint(0, 1)
w2= randint(0, 1)
w3= randint(0, 1)
y = r*(s_1**a1)*(s_2**a2)*(s_3**a3)

return y, x, [w1, w2, w3]
# Momento da prova
def Proof (n, s_1, s_2, s_3):
```

“““

Função que invoca a prova diante do verificador.

Input: Input: parâmetro público n e números secretos s1, s2, s3, ..., si. Onde i é um inteiro positivo.

Output: retorna os quadrados modulares $v1, v2, v3, y, x$ e a lista dos números binários. Onde y e x são os dois primeiros parâmetros resultantes da invocação da função Comunication(n, s_1, s_2, s_3) :.

“““

```
Z_n= IntegerModRing(n)
v_1 = Z_n(s_1)**2
v_2 = Z_n(s_2)**2
v_3 = Z_n(s_3)**2
comun = Comunication(n, s_1, s_2, s_3)
```

return $(v_1, v_2, v_3,$ comun$[0:2])$, comun$[2]$

Verificação

def Verify (proof, listaa):

“““

Função que invoca a verificação da prova.

Input: receeb os elementos da prova: v1, v2, v3, y, xe os némeros binários.

Output: retorna dois valores, um True, outro False.

Uso: o verificador calcula e depois verifica se: $y^2 = x * (v_1^{**}a1) * (v_2^{**}a2) * (v_3^{**}a3)$. Se sim, Paula passa da prova, ao contrário, Paula reprova.

“““

```
v_1, v_2, v_3, (y, x)= proof
[a1, a2, a3]= listaa
u = x*(v1**a1)*(v2**a2)*(v3**a3)
```

if $(y^{**}2 == u$

return True

else:

return False

7.8 ANEXO 8

def Elgamalinicial (bits):

“““

Função que invoca a escolha da chave privada e cálculo da chave pública Elgamal.
Input: recebe o número de bits.
Output: retorna um par de chaves, total de participantes ao esquema de partilha, total de elementos para recuperação do segredo, o grupo e o seu respetivo gerador.
Uso: Alice define o total de participantes na partilha da chave privada, o número de elemetos necessários para a reconstrução do segredo, calcula o p primo, onde Zp* é grupo, escolhe a chave privada $x \in Zp^*$, g gerador do grupo, calcula y= $g^x \ mod \ p$.
A chave a pública é: $(y; g; p)$.
Exemplo: Suponhamos que pretende-se gerar uma chave de 128 bits, então: a chave privada: 190422571672400112311789107296726056505,
Chave pública: (22475769130059677661640730068541036135 3, 3,
65773925495537165919969078067390052602)

“““

```
p= random_prime (2**bits)
Zp= IntegerModRing(p)
g= Zp. multiplicative_generator()
```

n=input(“Quantos participantes terá o esquema de partilha da CHAVE PRIVADA?“)
b=input(“Quantos deles terão de juntar seus segredos para RECUPERÁ-LA?“)
a0= input(“Digite o segredo a ser partilhado:“)

```
y= g**a0
PuKey= (p, g, y)
PrKey= a0
```

return PuKey, PrKey, n, k, Zp, g

ASSINATURA

def voto ():

“““

Função que invoca a geração de uma mensagem.
Input: nenhum parâmetro é usado.
Output: retorna a mensagem.
Uso: o eleitor escolhe um número que representa o concorrente.

```
    """
    m= input ("Digite um numero para eleger o seu candidato/ partido:")
return m

def chavesblindsig (bits):
    """
    Função que invoca a geração do par de chaves RSA.
Input: número de bits.
Output: chave privada e pública.
Uso: escolhe-se dois números primos p e q, calcula-se a função de Euler φ(n) = (p − 1),
isto é, d é o inverso multiplicativo de e mod n, Zn é o grupo, Exemplo: Seja 128 o nú-
mero de bits, suponhamos p = 6076863405864212999 e q = 1562809530454321421,
então o par de chaves é: chave privada:
d = 491397063228403561872764881059505611 1,
chave pública: (n, e) = (94969800459536991785622094527523515 79,
40425941209862137644162159371877743 1).
Atenção: Se os parâmetros p e q forem de tamanhos pequenos, um ataque por força
é trivial. A NIST recomenda no mínimo 3072 bits.
    """

p, q= nextprime (2**(bits/2)), next_prime (2**(bits))
n1= p*q
phi= (p-1)*(q-1)
e= ZZ. random_element (phi)

while gcd(e, phi) ! = 1:


e= ZZ. random_element(phi)
d= power_mod (e, -1, phi)
Chaveprivada= d
Chavepublica= (n1, e)

return Chaveprivada, Chavepublica

# Ofuscação

def ofuscacao (m, Chavepublica):
    """
```

Função que invoca a ofuscação da mensagem antes da assinatura.
Input: mensagem e chave pública.
Output: constante de ofuscação e a mensagem ofuscada.
Uso: escolhe-se k pertencente a Zn, e com k e a chave pública, o emissor calcula $(k**e)*m$ (ofusca), e envia ao signatárioa mensagem ofuscada m0.
Exemplo: suponhamos que queremos ofuscar $m = 70998000111870$, então,
m= 717349731693766832827349986846885114994760938826138214108
"""

```
n1, e= Chavepublica
Zn1= IntegerModRing(n1)
k0= Zn1(randint(2, n1- 1))
m0= (k0**e)*m
```

return (k0, m0)

Momento da assinatura

def assinaturacega (m0, Chaveprivada, Chavepublica):
"""
Função que invoca a assinatura da mensagem.
Input: mensagem ofuscada e o par de chave RSA.
Output: assinatura da mensagem ofuscada.
Uso: a assinatura é igual a potência cuja base é a mensagem ofuscada e o expoente é a chave privada.
"""

```
n1,_= Chavepublica
d= Chaveprivada
sig= m0**d
```

return sig

Desofuscação

def assinaturalimpa (sig, k0, Chavepublica):
"""
Função que invoca a desofuscação do texto ofuscado.
Input: mensagem ofuscada ofuscada, factor de ofuscação e a chave pública.
Output: texto claro assinado.

Uso: O emissor multiplica-se o texto ofuscado assinado pelo inverso da constante de ofuscação.

"""

```
n1, e= Chavepublica
sigm= (sig)*(1/k0)
```

return sigm

PARTILHA

def Coefpolinomio (Zp, Prkey, n, b):

"""

Função que invoca a escolha dos coeficientes.
Input: recebe o primo p, o segredo, o número de participantes e o número de elementos para recuperar.
Output: retorna os coeficientes do polinómio interpolador de Lagrange.
Uso: o parâmetro a_i, corresponde aos $k-1$ números aleatórios.

"""

```
a0= Prkey
t= b- 1
coef= [a0]
```

for r in range (1, t+1):
a = Zp. `random_element`()
coef.append(a)
return coef

Construção do polinómio

def polinomioconstruido (coef):

"""

Função que invoca a geração do polinómio interpolador de lagrange.
Input: recebe os coeficientes.
Output: retorna o polinómio.
Exemplo: suponhamos que temos o segredo 2000, $k = 2$. Escolhemos $k - 1 = 1$ coeficiente [2000, 167497848475266771127746087824805103 92], então o polinómio é: $P(x) = 167497848475266771127746087824805103 92 * x + 2000$

"""

```
Zp= coef[0].parent()
Pol.<x>= PolynomialRing(Zp)
polinomio= Pol(coef)

return polinomio

# Partes à partilhar

def partespartilhadas (n, polinomio):
    """
    Função que invoca o cálculo das partes secretas.
Input: recebe n e o polinómio interpolador de Lagrange.
Output: retorna pares ordenados (1, polinomio(1)),...,(n, polinomio(n)).
Uso: calcula-se as as imagens dos n números na função polinómial, e retorna-se os
pares ordenados secretos, e são distribuidos.
Exemplo: Seja bits = 128, suponhamos que queremos compartilhar 2000, as partes
secretas compartilhadas são:
[[1, 137624036525743829578287975035514517986],
[2, 275248073051487659156575950071029033972],
[3, 412872109577231488734863925106543549958]]
    """

pares= []
for x in range (1, n+1):
pares.append([x, polinomio(x)])

return pares

# VERIFICAÇÃO DE CONSISTÊNCIA

# Calculo das potencias

i=0
pot = []
def potencia (g, coef):
    """
    Função que invoca o cálculo das potências (parâmetros públicos) para a verifi-
cação.
Input: recebe o gerador e os coeficientes do polinómio.
```

Output: retorna as k potencias.
Uso: calcula-se e distribui-se as k potencias a todos os participantes.
"""

```
pot= []
k= len(coef)
```

for i in range (b):

```
pot.append(g**coef[i])
```

return pot

Verificação da consistência das partes secretas

def verificacaopartes (i, g, pot, partesecreta):
"""
Função que invoca a verificação da consistência das partes secretas de cada interveniente.
Input: recebe i-posição de cada participante, o gerador e as potencias.
Output: retorna dois valores, True ou False, caso positivo ou negativo, respetivamente.
Atenção: Se o resultado for False, implica dizer que, o responsável pela partilha distribuiu maliciosamente a parte secreta ou por engano assim, a fez.
"""

```
prod= 1
b= len(pot)
for j in range (b):
prod= prod*(pot[j]**(i**j))
```

if $g * *$partesecreta == prod:
return True
else:
return False

RECONSTRUÇÃO DO POLINÓMIO

def recostrucaopolinomio (coef, pares):
"""

Função que invoca a recuperação do segredo compartihado.
Input: recebe k e as partes secretas compartilhadas.
Output: retorna o polinómio interpolador.
Uso: calcula-se os polinómios de Langrange lj e em seguida, calcula o polinómio interpolador pol, então, a0 de pol é o segredo reconstruido.
“““

```
Zp= coef[0] . parent()
Pol. <x>= PolynomialRing(Zp)
k= len(coef)
pol= Pol(0)
```

for j in range(k):
$lj =$ Pol(Zp(1))
for i in range(k):
if j! = i :
lj = lj * ((x − pares[i][0])/(pares[j][0] - pares[i][0]))
pol = pol + lj * pares[j][1]
return pol

Cifração

def cifracaovoto (PuKey, m):
“““
Função que invoca a cifração do texto puro.
Input: a chave pública e o texto puro.
Output: retorna o criptograma, e o inteiro escolhido por Alice.
Uso: Bob calcula o criptograma (α; β), efetuando, $\alpha = g^k mod\ p$, e $\beta = M.y^k mod\ p$, onde $k1$ é um inteiro pertecente ao grupo.
Exemplo: Suponhamos que queremos cifrar $M = 11002$, então, o criptograma resultante é: [21863336599180136162811937124449303658 1 , 655619131077072917206333566472546844 60]
“““

```
p, g, y= PuKey
Zp= IntegerModRing(p)
k1= randint(2, p- 1)
Alfa =g**k
Beta= m*(y**k)
Criptograma= (Alfa, Beta)
```

return Criptograma, $k1$

Decifração

def Decifracaovoto (PuKey, PrKey, Criptograma):

“““

Função que invoca a decifração do texto cifrado.
Input: a chave pública, privada e o criptograma.
Output: retorma a mensagem original.
Uso: Alice decifra a mensagem: Texto puro = $M \; . \; \alpha^{-x} \; . \; \alpha^{x} mod \; p$, sendo, $\alpha \; = \; g^{k}$ recebido de Bob.

“““

```
p, g, y= PuKey
a0= PrKey
Zp= IntegerModRing(p)
skey= (Criptograma[0]**a0)
tp= (1/skey)* Criptograma[1]
```

return tp

VERIFICAÇÃO DA AUTENTICIDADE

def verificacaoassinatura (m, Chavepublica, sigm):

“““

Função que invoca a verificação da autenticidade.
Input: recebe a mensagem, a chave pública e a mensagem assinada.
Output: retorna dois valores lógicos: True-caso a auteticidade for válida e False-caso a autenticidade for inválida.
Atenção: caso a autenticidade não for confirmada, o recetor rejeita a mensagem, isto, a mensagem foi alvo de um ataque.

“““

$n1, \; e \; = \;$ Chavepublica
if sigm$* * e \; = =$m:
return True
else:
return False

Prova de conhecimento zero ALICE VS BOB

INICIALIZAÇÃO

def parametropublico (bits):

"""

Função que invoca a inicialização da prova de conhecimento zero.

Input: recebe o número de bits.

Output: retorna o parâmetro público $n2$.

Uso: escolhe-se p e q, númros e calcula-se $n2 = p \,.\, q$.

"""

$p,\ q\ =$ `random_prime` $(2 * *(bits/2))$,

`random_prime` $(2 * *(bits/2))$

$n2 = p * q$

return $n2$

Escolha dos segredos

def secretsproof $(n2)$:

"""

Função que invoca a escolha de segredos.

Input: parâmetro público $n2$.

Output: retorna os segredos.

Uso: Alice escolhe os números secretos pertencentes em $Zn2$.

"""

```
Zn2= IntegerModRing(n2)
s1= Zn2. random_element()
s2= Zn2. random_element()
s3= Zn2. random_element()
```

return $s1,\ s2,\ s3$

Interação entre Paula e Veigas

def Comunicacaoproof $(n2,\ s1,\ s2,\ s3)$:

"""

Função que invoca a interação entre o provador e o verificador.

Input: parâmetro público $n2$ e números secretos $s1,\ s2,\ s3, ...,\ sk$. Onde k é um inteiro positivo.

Output: retorna $r * (s1 * *a1) * (s2 * *a2) * (s3 * *a3),\ s * r * *2$ e a lista de números

binários, onde n pertence ao grupo Zn e r é 1 ou -1.
Uso: seja $Zn2$, ecolhe-se um número inteiro r e escolhe-se um números entre 1 ou -1. Calcula-se em seguida $x = s.r**2$, escolhe-se também os números binários $a1, a2, a3$, ..., k e calcula-se $y = r.(s1^{a1}).(s2^{a2}).(s3^{a3})$.
Atenção: a quantidade de números secretos e de números binários devem ser iguais, devido o cálculo de y.
"""

```
Zn2= IntegerModRing(n2)
r= Zn2.random_element()
x= s*r**2
w1= randint(0, 1)
w2= randint(0, 1)
w3= randint(0, 1)
y= r*(s1**a1)*(s2**a2)*(s3**a3)
```

return y, x, $[a1, a2, a3]$

Momento da prova

def Proof $(n2, s1, s2, s3)$:
"""
Função que invoca a prova diante do verificador.
Input: Input: parâmetro público $n2$ e números secretos $s1$, $s2$, $s3$, ..., k. Onde k é um inteiro positivo.
Output: retorna os quadrados modulares $v1$, $v2$, $v3$, y, x e a lista dos números binários. Onde y e x são os dois primeiros parâmetros resultantes da invocação da função Comunicacaoproof.
"""

```
Zn2= IntegerModRing(n2)
v1= Zn2(s1)**2
v2= Zn2(s2)**2
v3= Zn2(s3)**2
comun = Comunicacaoproof (n2, s1, s2, s3)
```

return $(v1, v2, v3, \text{comun}[0:2]), \text{comun}[2]$

Verificação

def Verificacaoproof (proof, listaa):

“““

Função que invoca a verificação da prova.

Input: receeb os elementos da prova: $v1$, $v2$, $v3$, y, x e os némeros binários. Output: retorna dois valores, um True, outro False.

Uso: o verificador calcula $w = -x * (v1 * *a1) * (v2 * *a2) * (v3 * *a3)$ k $= x * (v1 * *a1) * (v2 * *a2) * (v3 * *a3)$, e depois verifica se y^2 é igual a k ou w. Se sim, Paula passa da prova, ao contrário, Paula reprova.

“““

```
v1, v2, v3, (y, x)= proof
[a1, a2, a3]= lista a
u= x*(v1**a1)*(v2**a2)*(v3**a3)
if y**2 == u:
```

return True

else:

return False