

# Módulo 8 - Aulas 3 e 4

## Módulo 8: Conceitos de criptografia

### Aula 3: Funções Hash

#### Objetivos

- ☒ Compreender o conceito de função hash e sua importância na criptografia e integridade de dados.
- ☒ Explorar diferentes algoritmos de função hash e suas propriedades, como uniformidade na distribuição de valores de hash.
- ☒ Aplicar funções hash em exemplos práticos, como verificação de integridade de arquivos e autenticação de dados, para fortalecer a compreensão teórica.

#### Conceitos

- ☒ Função hash.
- ☒ Integridade de dados.
- ☒ Colisões em funções hash.

#### Introdução

As funções hash desempenham um papel fundamental em diversos aspectos da segurança e integridade de dados. São algoritmos poderosos e versáteis, amplamente utilizados em criptografia, verificação de integridade de arquivos, autenticação de dados e muito mais. Nesta aula, vamos explorar os conceitos e aplicações das funções hash, entendendo como elas funcionam e por que são tão importantes na proteção e validação de informações.

Durante a aula, vamos mergulhar nos fundamentos das funções hash, compreendendo seu propósito e funcionamento básico. Vamos examinar diferentes algoritmos de função hash e suas características, incluindo a distribuição uniforme de valores de hash e a resistência a colisões. Ao entender essas propriedades, poderemos avaliar a qualidade e a segurança de diferentes funções hash, escolhendo a mais adequada para cada situação.

Além de explorar a teoria por trás das funções hash, esta aula também terá um enfoque prático. Vamos aprender a aplicar funções hash em situações reais, como a verificação de integridade de arquivos ou a autenticação de dados. Veremos como calcular o hash de um arquivo e compará-lo com um valor de referência para verificar se houve qualquer modificação. Com isso, estaremos preparados para utilizar as funções hash de maneira efetiva em diferentes contextos, fortalecendo a segurança e a confiabilidade dos dados.



Função Hash aplicada.

## Características das Funções Hash

Uma função hash é um algoritmo matemático que recebe um conjunto de dados de entrada e produz uma sequência de caracteres ou valores de tamanho fixo, chamados de "hash" ou "valor de hash". Essa função tem a propriedade de ser rápida de calcular e determinística, ou seja, a mesma entrada sempre produzirá o mesmo hash. O objetivo principal de uma função hash é mapear um conjunto de dados de tamanho arbitrário para um valor de hash de tamanho fixo, que representa de forma única e compacta os dados originais.

As funções hash são amplamente utilizadas em diversas áreas da computação, como criptografia, verificação de integridade de dados, autenticação e indexação de informações. Elas desempenham um papel fundamental na segurança e na eficiência de muitos sistemas. Uma função hash de qualidade deve ser capaz de produzir valores de hash únicos para diferentes conjuntos de dados, evitar colisões e ser resistente a tentativas de reversão.

No contexto da criptografia, as funções hash desempenham um papel importante na proteção de senhas e na verificação da integridade dos dados transmitidos. Elas também são utilizadas para indexar informações em bancos de dados, acelerando a busca e recuperação de dados.

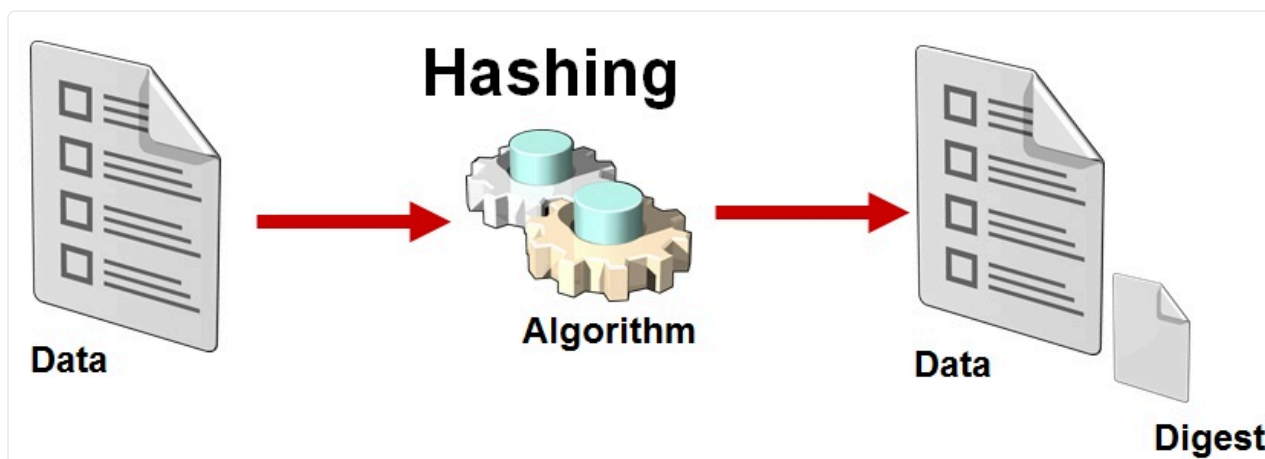
### **Resistência a tentativas de inversão**

A resistência a tentativas de inversão é uma propriedade fundamental de uma função hash. Ela se refere à dificuldade de recuperar ou reverter os dados originais a partir do valor de hash gerado por essa função.

Em uma função hash segura, mesmo que dois conjuntos de dados diferentes possam produzir o mesmo valor de hash (o que é conhecido como colisão de hash), encontrar a entrada original a partir do valor de hash deve ser computacionalmente difícil e impraticável.

Essa resistência é essencial para garantir a segurança de diversas aplicações. Por exemplo, em criptografia, quando uma senha é armazenada em um sistema, ela não é armazenada diretamente, mas sim o seu valor de hash. Quando um usuário tenta fazer login, a senha inserida é novamente convertida em um valor de hash e comparada com o valor armazenado anteriormente. Se ambos os valores de hash coincidirem, a senha é considerada correta.

A resistência a tentativas de inversão garante que mesmo que um atacante obtenha acesso aos valores de hash armazenados, ele terá uma tarefa árdua para determinar a senha original correspondente. O atacante precisaria testar diferentes combinações de entrada e calcular os valores de hash correspondentes até encontrar uma colisão válida. Com uma função hash segura e resistente a tentativas de inversão, isso é extremamente difícil e consome uma quantidade significativa de tempo e recursos computacionais.



Função Hash com resultado (Digest).

## Colisões em Funções Hash

Colisões em funções hash ocorrem quando dois conjuntos de dados distintos produzem o mesmo valor de hash. Em outras palavras, é quando duas entradas diferentes são mapeadas para o mesmo resultado de hash. Essa situação é indesejável, pois uma das principais propriedades de uma função hash é a unicidade, ou seja, cada conjunto de dados de entrada deve gerar um valor de hash único.

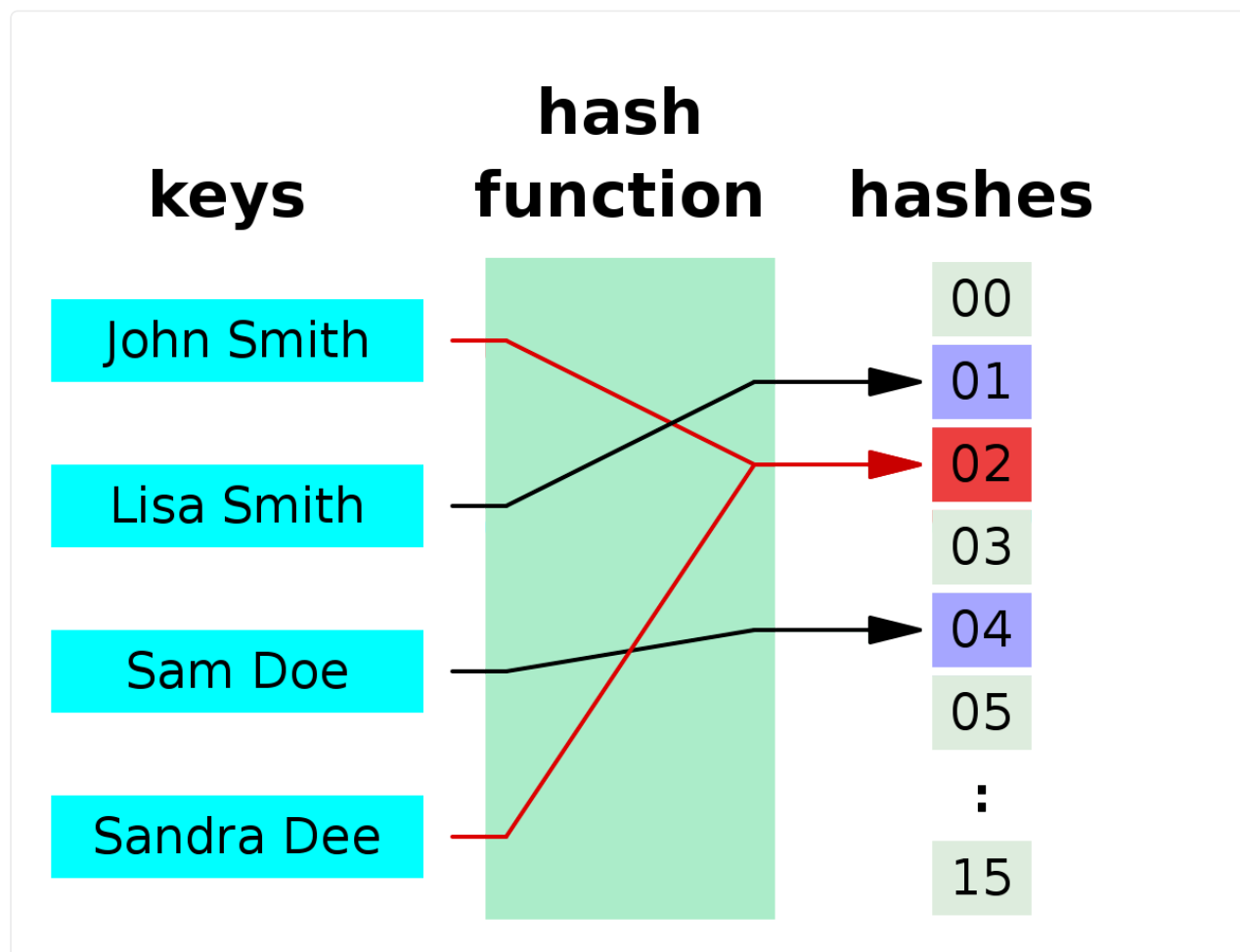
As colisões podem ser classificadas em duas categorias: colisões acidentais e colisões intencionais.

- Colisões acidentais são ocorrências não planejadas e imprevisíveis, resultado de características das funções hash e do espaço limitado de valores de hash. Embora seja extremamente improvável que uma função hash perfeitamente distribuída e de alta qualidade apresente colisões acidentais, é possível que, em casos reais, elas ocorram devido à natureza probabilística das funções hash.
- Colisões intencionais são uma preocupação maior, especialmente em contextos de segurança. São o resultado de ataques deliberados, nos quais um adversário procura encontrar duas entradas diferentes que produzem o mesmo valor de hash. O objetivo de um ataque de colisão intencional pode ser comprometer a integridade dos dados ou até mesmo encontrar uma fraqueza na função hash.

As colisões são indesejáveis porque podem levar a problemas de segurança e confiabilidade. Em aplicações como criptografia, assinatura digital e autenticação, as colisões podem ser exploradas para falsificar informações ou comprometer a

integridade dos dados. Portanto, é fundamental utilizar funções hash que apresentem uma resistência adequada a colisões, dificultando a ocorrência tanto de colisões acidentais quanto de ataques de colisão intencionais.

Para garantir a segurança e a confiabilidade das funções hash, os algoritmos utilizados devem ser cuidadosamente projetados e analisados para minimizar as chances de colisões e resistir a tentativas de encontrar colisões intencionais.



Colisão na saída 02

### Distribuição uniforme

A distribuição uniforme é uma propriedade importante em funções hash. Ela se refere à maneira como os valores de hash são distribuídos no espaço de saída da função. Em outras palavras, uma função hash com uma distribuição uniforme garante que todos os possíveis valores de hash tenham a mesma probabilidade de serem gerados.

Uma distribuição uniforme é desejada porque evita a ocorrência de colisões excessivas. Se a distribuição dos valores de hash não for uniforme, pode haver agrupamentos ou regiões no espaço de saída com uma maior concentração de valores de hash, o que aumenta a probabilidade de colisões.

Para ilustrar, imagine uma função hash que mapeia números inteiros em valores de hash de tamanho fixo. Se a distribuição não for uniforme, é possível que haja uma maior quantidade de valores de hash próximos a zero, por exemplo, resultando em uma concentração não uniforme. Isso tornaria mais provável a ocorrência de colisões para os conjuntos de dados que mapeiam para valores próximos a zero.

Uma distribuição uniforme garante que, independentemente dos dados de entrada, a função hash espalhe os valores de hash de forma equilibrada em todo o espaço de saída. Isso é especialmente importante em aplicações que dependem de identificadores únicos ou que precisam evitar colisões para garantir a integridade ou a segurança dos dados.

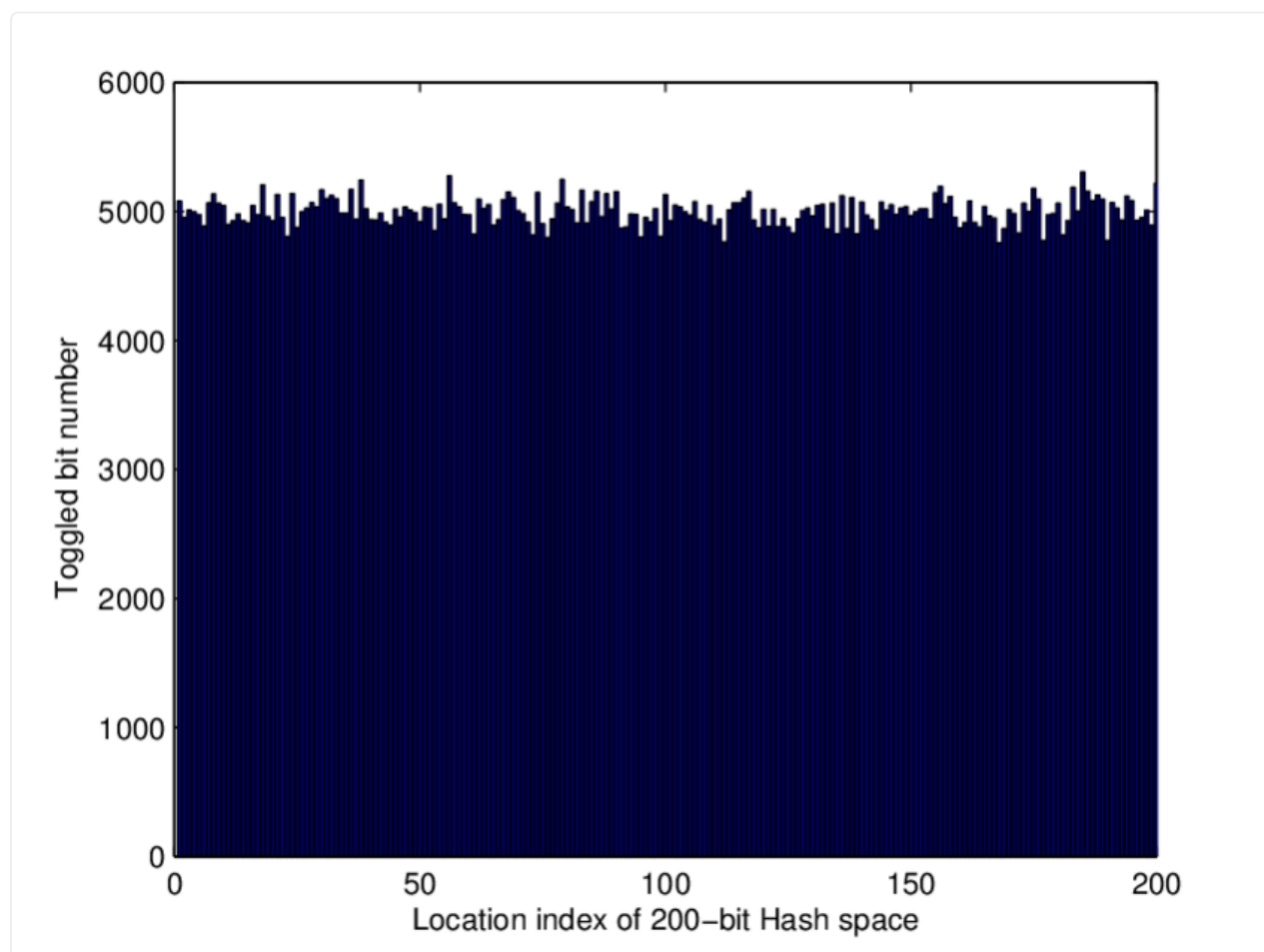


Gráfico de uma distribuição uniforme.

## Unicidade em Funções Hash

Unicidade é uma propriedade fundamental das funções hash, que garante que cada conjunto de dados de entrada produza um valor de hash único. Em outras palavras, dois conjuntos de dados diferentes não devem gerar o mesmo valor de hash.

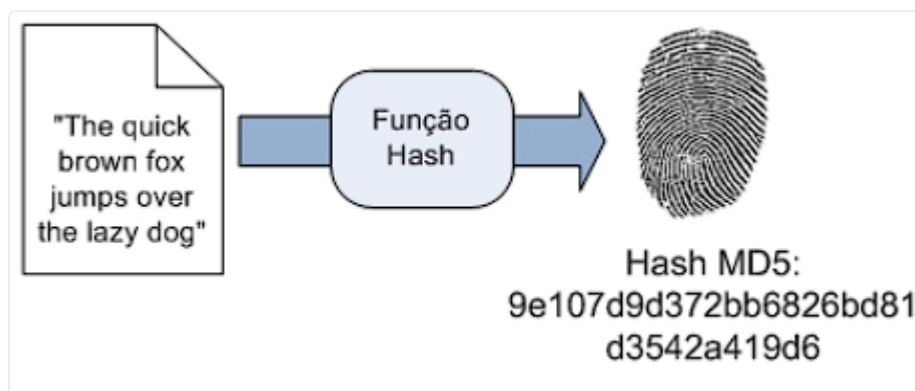
A unicidade é importante porque permite identificar e distinguir de forma única conjuntos de dados específicos por meio de seus valores de hash correspondentes. Isso é essencial em várias aplicações, como armazenamento e recuperação de dados, indexação, detecção de duplicatas, verificação de integridade e muito mais.

Quando uma função hash é projetada com unicidade, ela minimiza as chances de colisões, onde dois conjuntos de dados distintos geram o mesmo valor de hash. No entanto, é importante ressaltar que, em teoria, a existência de colisões é inevitável devido ao espaço limitado de valores de hash e ao potencial infinito de conjuntos de dados possíveis. No entanto, funções hash de alta qualidade têm uma probabilidade extremamente baixa de gerar colisões em casos práticos.

Para avaliar a unicidade de uma função hash, é comum utilizar métricas como o tamanho do espaço de valores de hash e a distribuição dos valores de hash. Quanto maior o espaço de valores de hash, menor a probabilidade de colisões. Além disso, uma função hash com uma distribuição uniforme dos valores de hash ao longo do espaço de saída também reduz a probabilidade de colisões.

No contexto da segurança, a unicidade desempenha um papel crítico. Por exemplo, em criptografia de senhas, é essencial que diferentes senhas sejam convertidas em valores de hash diferentes. Dessa forma, mesmo que um invasor tenha acesso aos valores de hash armazenados, ele terá dificuldade em determinar as senhas originais correspondentes.





Dados únicos.

## Integridade em Funções Hash

Refere-se à capacidade de verificar se os dados permaneceram inalterados durante a sua transmissão, armazenamento ou processamento. Uma função hash é usada para calcular um valor de hash a partir de um conjunto de dados e, ao comparar esse valor com um valor de referência conhecido, é possível detectar qualquer modificação ou corrupção nos dados.

A integridade é uma propriedade crítica em várias aplicações, como transferência de arquivos, verificação de integridade de dados, detecção de adulteração e prevenção de ataques maliciosos. Ela garante que os dados não tenham sido alterados acidentalmente ou intencionalmente, fornecendo um mecanismo para validar a sua integridade.

Quando um conjunto de dados é processado por uma função hash, ele gera um valor de hash único, que é como uma "impressão digital" dos dados originais. Qualquer alteração nos dados, mesmo que seja mínima, resultará em um valor de hash completamente diferente. Ao comparar o valor de hash calculado com o valor de referência previamente conhecido, é possível determinar se os dados permaneceram intactos.

A integridade é particularmente relevante em sistemas de segurança, como assinaturas digitais e autenticação. Por exemplo, em um sistema de assinatura digital, uma função hash é usada para calcular o valor de hash de uma mensagem assinada. Qualquer alteração nos dados da mensagem após a assinatura resultará em um valor de hash diferente, tornando evidente que a mensagem foi adulterada.



Além disso, a integridade também é importante em sistemas de armazenamento e transferência de dados, onde é essencial garantir que os dados permaneçam íntegros ao longo do tempo. Ao verificar periodicamente os valores de hash dos dados armazenados e compará-los com os valores de referência, é possível detectar quaisquer modificações não autorizadas ou corrupções nos dados.



Integridade de dados.

### **Confusão em Funções Hash**

A confusão em funções hash busca tornar o relacionamento entre a entrada e a saída da função complexo e difícil de ser analisado. Isso é alcançado através da aplicação de operações matemáticas não reversíveis nos dados de entrada. Em outras palavras, uma função hash com uma boa confusão produzirá um valor de hash radicalmente diferente comparado ao valor de entrada.

A confusão é necessária para garantir que pequenas alterações nos dados de entrada causem grandes efeitos nos valores de hash resultantes. Isso dificulta a previsão do valor de hash ou a inferência de informações sobre os dados originais com base no valor de hash.

Uma função hash confusa é projetada para espalhar as propriedades dos dados de entrada de forma equilibrada e caótica em todo o processo de cálculo do valor de hash. Isso é alcançado através do uso de operações criptográficas e técnicas de embaralhamento que amplificam as diferenças nas entradas e propagam essas diferenças ao longo de cada etapa do algoritmo. A confusão dificulta a manipulação

ou a falsificação dos dados sem alterar drasticamente o valor de hash correspondente.

Além disso, a confusão ajuda a evitar correlações entre os dados de entrada e os valores de hash, o que seria prejudicial para a segurança. Se houvesse padrões ou dependências facilmente identificáveis entre os dados de entrada e os valores de hash, um adversário poderia explorar essas informações para encontrar vulnerabilidades ou ataques.



Confusão de dados.

## Difusão em Funções Hash

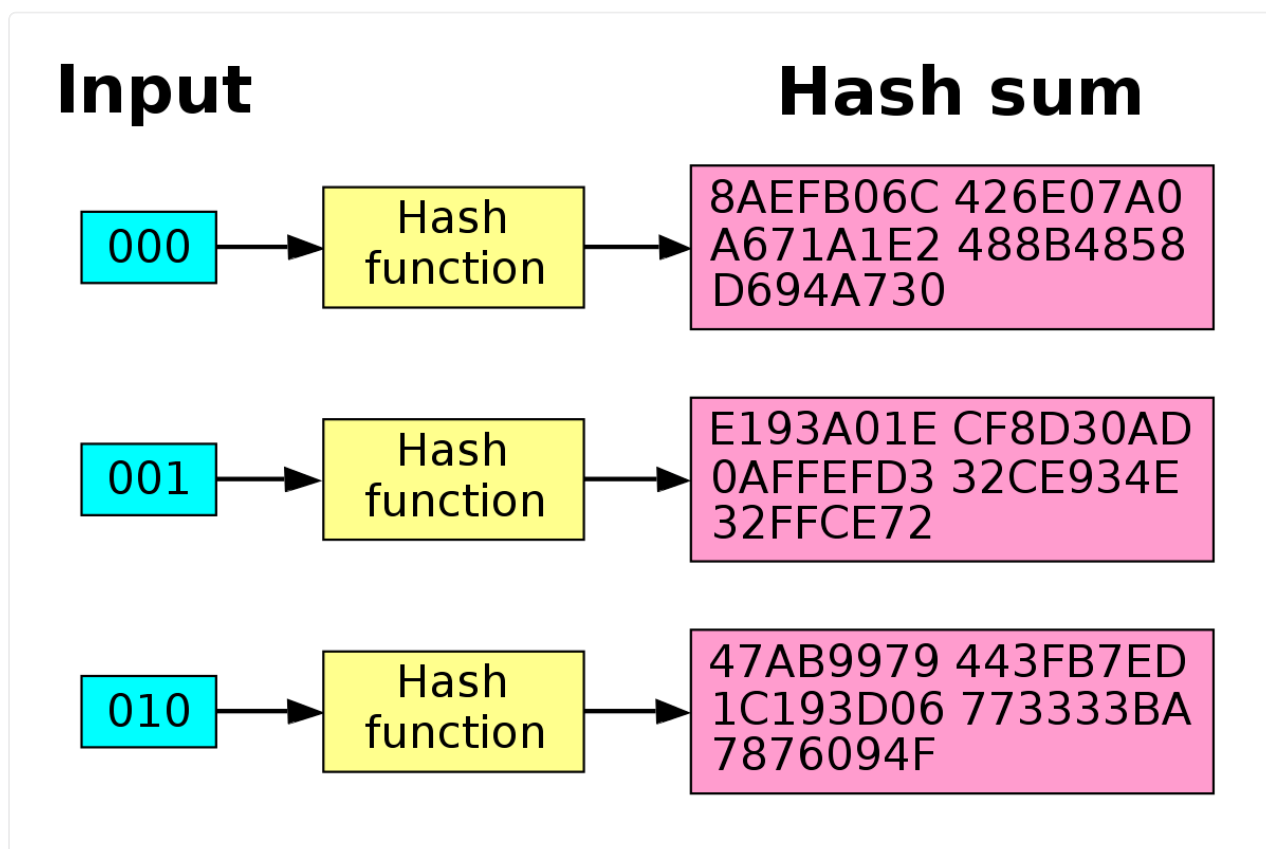
Refere à propriedade de uma função em espalhar as características dos dados de entrada por todo o valor de hash resultante. Em outras palavras, uma função hash com uma boa difusão garante que qualquer mudança nos dados de entrada afete significativamente todos os bits do valor de hash.

A difusão é alcançada por meio do processamento iterativo dos dados de entrada em uma série de transformações. Cada etapa do algoritmo de função hash aplica operações que misturam e embaralham os bits dos dados, propagando as mudanças por todo o valor de hash. Essas operações incluem combinações lógicas, deslocamentos, substituições não lineares e outras técnicas criptográficas.

A propriedade de difusão é fundamental para a segurança de uma função hash, pois garante que qualquer alteração nos dados de entrada produza um efeito significativo em todos os bits do valor de hash. Isso significa que mesmo uma pequena modificação nos dados resultará em um valor de hash completamente diferente.

A difusão também é responsável por garantir que os dados de entrada sejam amplamente distribuídos no espaço de saída do valor de hash. Isso significa que pequenas variações nos dados de entrada devem levar a mudanças imprevisíveis e extensas no valor de hash. Dessa forma, a difusão contribui para minimizar as correlações e os padrões nos valores de hash, tornando a função mais resistente a ataques criptográficos.

Uma função hash com uma boa difusão impede que um adversário possa inferir informações sobre os dados originais com base no valor de hash, dificultando a reversão do processo de hashing. Além disso, a difusão ajuda a espalhar quaisquer propriedades estatísticas dos dados de entrada, dificultando a identificação de padrões ou dependências.



Difusão em Hash.

## Paradoxo do aniversário em Funções Hash

O Paradoxo do Aniversário é um fenômeno estatístico que ilustra a probabilidade de ocorrência de colisões em um conjunto de elementos, como os valores de hash em funções hash. Embora o termo "paradoxo" seja utilizado, ele não se refere a uma contradição real, mas sim a uma surpreendente propriedade estatística.

No contexto das funções hash, o paradoxo do aniversário é aplicado para ilustrar a probabilidade de duas entradas diferentes produzirem o mesmo valor de hash. Apesar do tamanho potencialmente grande do espaço de saída de valores de hash, a probabilidade de colisões aumenta à medida que o número de elementos aumenta.

Em outras palavras, o paradoxo do aniversário mostra que, para um espaço de valores de hash de tamanho fixo, a probabilidade de colisões se torna significativa quando o número de elementos (ou entradas) se aproxima da raiz quadrada do espaço de valores de hash. Por exemplo, se o espaço de valores de hash tiver 10.000 valores possíveis, a probabilidade de pelo menos uma colisão ocorrer aumenta consideravelmente quando houver cerca de 100 entradas.

Isso tem implicações importantes para a segurança das funções hash. Uma função hash deve ter um tamanho de espaço de valores de hash adequado e ser projetada de forma a minimizar as colisões mesmo quando o número de elementos a serem processados aumenta. Caso contrário, a probabilidade de colisões pode ser explorada por adversários para falsificar dados ou comprometer a integridade de sistemas que dependem da unicidade dos valores de hash.

## Aplicações de Funções Hash

As funções hash são amplamente utilizadas em diversas tecnologias e aplicações. Algumas das principais áreas em que as funções hash desempenham um papel fundamental são:

- **Criptografia:** As funções hash são essenciais em algoritmos criptográficos para garantir a segurança e a integridade dos dados. Elas são usadas em algoritmos de assinatura digital, como o RSA e o DSA, para calcular resumos criptográficos dos dados e garantir a autenticidade e a integridade das informações

transmitidas. Além disso, as funções hash são usadas em algoritmos de hash de senha, como o bcrypt e o scrypt, para armazenar senhas de forma segura.

- **Armazenamento e verificação de integridade de dados:** Em sistemas de armazenamento de dados, as funções hash são usadas para verificar a integridade dos arquivos e garantir que não tenham sido alterados ou corrompidos. Elas geram valores de hash para os arquivos e, posteriormente, esses valores são verificados para garantir que os dados permaneçam íntegros durante o armazenamento ou a transferência.
- **Tabelas de dispersão (hash tables):** As estruturas de dados conhecidas como tabelas de dispersão (ou hash tables) utilizam funções hash para armazenar e recuperar dados de forma eficiente. Elas usam os valores de hash como índices para mapear os dados em uma tabela, permitindo o acesso rápido aos elementos. As funções hash são cruciais para distribuir uniformemente os dados na tabela e minimizar colisões.
- **Verificação de integridade de arquivos e downloads:** Ao baixar arquivos da Internet, as funções hash são usadas para verificar a integridade dos arquivos baixados. Por exemplo, muitos sites fornecem valores de hash dos arquivos originais, e os usuários podem calcular os valores de hash dos arquivos baixados e compará-los aos valores fornecidos. Se os valores de hash coincidirem, isso significa que o arquivo foi baixado corretamente e sem modificações.
- **Verificação de integridade de mensagens e comunicações:** As funções hash também são utilizadas para verificar a integridade das mensagens e comunicações em redes. Ao calcular o valor de hash das mensagens transmitidas, é possível verificar se elas foram alteradas durante o transporte. Isso é fundamental para garantir a autenticidade e a integridade das comunicações, especialmente em protocolos de segurança, como o IPsec e o SSL/TLS.





Funcionamento de Hash.

## Principais algoritmos de Funções Hash

Os principais algoritmos de funções hash são ferramentas fundamentais para garantir a segurança e a integridade dos dados em diversas aplicações. Esses algoritmos são projetados para calcular resumos criptográficos dos dados de entrada, gerando valores de hash únicos e de tamanho fixo. Cada algoritmo possui suas próprias propriedades, tamanhos de saída e características específicas, que determinam sua adequação em diferentes contextos:

### MD5 (Message Digest Algorithm 5)

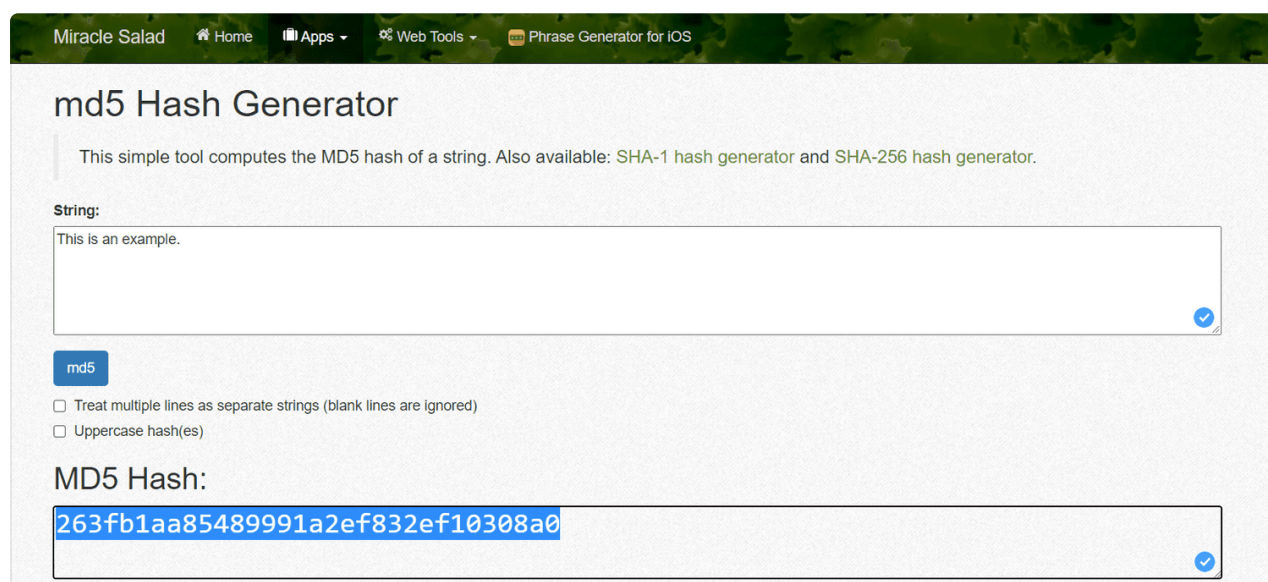
O algoritmo MD5 (Message Digest Algorithm 5) é um dos algoritmos de função hash criptográfica mais conhecidos e amplamente utilizados. Ele foi desenvolvido por Ronald Rivest em 1991 e é comumente usado para calcular resumos de mensagem de 128 bits.

O MD5 opera em blocos de dados de tamanho fixo (512 bits) e utiliza uma série de operações bitwise, aritméticas e lógicas para processar os dados de entrada. O algoritmo passa por quatro etapas principais: inicialização, processamento de blocos, finalização e geração do valor de hash.

Durante o processamento, o algoritmo divide os dados de entrada em blocos de 512 bits e aplica várias operações em cada bloco para misturar os bits e propagar as mudanças por todo o valor de hash. Essas operações incluem rotações, adições módulo  $2^{32}$ , funções lógicas e tabelas de constantes pré-definidas.

Uma característica importante do MD5 é a sua propriedade de resistência a colisões, que significa que é extremamente difícil encontrar duas mensagens diferentes que produzam o mesmo valor de hash MD5. No entanto, devido a vulnerabilidades descobertas, o MD5 não é mais considerado seguro para aplicações criptográficas críticas, pois é possível gerar colisões de forma prática com ataques computacionais modernos.

O MD5 ainda é utilizado em algumas aplicações não criptográficas, como verificação de integridade de arquivos e armazenamento de senhas. No entanto, para aplicações que exigem maior segurança, é recomendado o uso de algoritmos mais robustos, como SHA-256 ou SHA-3.

The image shows a web browser window with a dark green header bar. The header contains the text "Miracle Salad" and navigation links: "Home", "Apps", "Web Tools", and "Phrase Generator for iOS". The main content area has a title "md5 Hash Generator" and a subtitle "This simple tool computes the MD5 hash of a string. Also available: [SHA-1 hash generator](#) and [SHA-256 hash generator](#)." Below this is a "String:" label and a text input field containing "This is an example." with a blue checkmark icon on the right. Under the input field is a blue button labeled "md5". Below the button are two checkboxes: "Treat multiple lines as separate strings (blank lines are ignored)" and "Uppercase hash(es)". Below the checkboxes is the label "MD5 Hash:" and a text output field containing the hash "263fb1aa85489991a2ef832ef10308a0" with a blue checkmark icon on the right.

Funcionamento de Hash MD5.

## SHA-1 (Secure Hash Algorithm 1)

O algoritmo SHA-1 (Secure Hash Algorithm 1) é um algoritmo de função hash criptográfica que foi amplamente utilizado, mas agora é considerado inseguro para aplicações criptográficas críticas. Ele foi desenvolvido pelo National Institute of Standards and Technology (NIST) dos Estados Unidos em 1995.

O SHA-1 opera em blocos de 512 bits e gera um valor de hash de 160 bits. Assim como outros algoritmos de função hash, ele passa por várias etapas para processar os dados de entrada e produzir o valor de hash.



Durante o processamento, o algoritmo SHA-1 divide os dados em blocos de 512 bits e realiza uma série de operações bitwise, lógicas e aritméticas em cada bloco. Essas operações incluem rotações, combinações de bits com operadores lógicos (como AND, OR e XOR) e adições módulo  $2^{32}$ .

O objetivo principal do SHA-1 é gerar um valor de hash único para cada conjunto de dados de entrada. No entanto, em 2005, foram divulgadas vulnerabilidades teóricas no SHA-1, indicando que é possível encontrar colisões, ou seja, encontrar duas mensagens diferentes que produzam o mesmo valor de hash. Como resultado, o SHA-1 não é mais considerado seguro para aplicações que exigem alta resistência a ataques criptográficos. Devido às vulnerabilidades conhecidas, o uso do SHA-1 tem sido gradualmente desencorajado em favor de algoritmos mais robustos.

## **SHA-2 (Secure Hash Algorithm 2)**

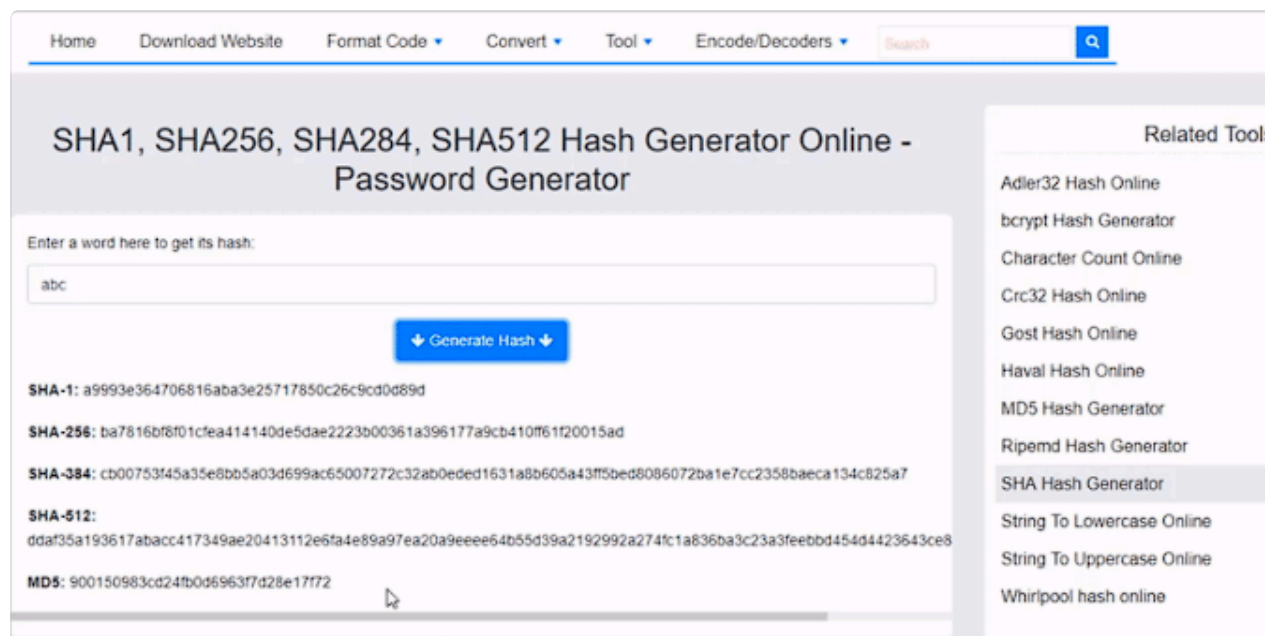
O algoritmo SHA-2 (Secure Hash Algorithm 2) é uma família de algoritmos de função hash criptográfica que foi projetada como uma melhoria do SHA-1. Foi desenvolvida pelo National Institute of Standards and Technology (NIST) dos Estados Unidos e é amplamente utilizada em várias aplicações criptográficas.

A família SHA-2 inclui vários tamanhos de saída, como SHA-224, SHA-256, SHA-384 e SHA-512. Esses números representam o tamanho dos valores de hash produzidos pelo algoritmo: 224, 256, 384 e 512 bits, respectivamente. Quanto maior o tamanho de saída, maior é a resistência a ataques criptográficos.

O processo de geração de hash do SHA-2 é semelhante ao do SHA-1. Os dados de entrada são divididos em blocos de 512 bits e passam por uma série de operações bitwise, lógicas e aritméticas em cada bloco. Essas operações incluem rotações, combinações de bits e adições módulo  $2^{32}$  ou  $2^{64}$ , dependendo do tamanho do valor de hash.

Uma das principais melhorias do SHA-2 em relação ao SHA-1 é o tamanho do valor de hash e o número de iterações realizadas durante o processamento. Esses fatores tornam os valores de hash do SHA-2 mais robustos e menos propensos a colisões.

Atualmente, o SHA-256 é amplamente utilizado e considerado seguro para a maioria das aplicações. Ele é utilizado em sistemas de autenticação, certificados digitais, protocolos de segurança (como SSL/TLS) e outros cenários em que a integridade e a autenticidade dos dados são fundamentais.



The screenshot shows a web interface for a hash generator. At the top, there is a navigation bar with links: Home, Download Website, Format Code, Convert, Tool, and Encode/Decoders. A search bar is also present. The main heading is "SHA1, SHA256, SHA384, SHA512 Hash Generator Online - Password Generator". Below this, there is a text input field with the value "abc" and a blue button labeled "Generate Hash". The results are displayed below the button:

- SHA-1:** a9993e364706816aba3e25717850c26c9cd0d89d
- SHA-256:** ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
- SHA-384:** cb00753f45a35e8bb5a03d699ac65007272c32ab0e0ded1631a8b605a43ff5bed8086072ba1e7cc2358baeca134c825a7
- SHA-512:** dda735a193617abacc417349ae20413112e6fa4e89a97ea20a9e9ee64b55d39a2192992a274fc1a836ba3c23a3feebd454d4423643ce8
- MD5:** 900150983cd24fb0d6963f7d28e17f72

On the right side, there is a "Related Tools" section with a list of other online tools: Adler32 Hash Online, bcrypt Hash Generator, Character Count Online, Crc32 Hash Online, Gost Hash Online, Haval Hash Online, MD5 Hash Generator, Ripemd Hash Generator, SHA Hash Generator (highlighted), String To Lowercase Online, String To Uppercase Online, and Whirlpool hash online.

SHA-1 e SHA-2 online.

### SHA-3 (Secure Hash Algorithm 3)

O algoritmo SHA-3 (Secure Hash Algorithm 3) é uma família de algoritmos de função hash criptográfica que foi selecionada pelo National Institute of Standards and Technology (NIST) dos Estados Unidos em 2012. Ele foi projetado como uma alternativa aos algoritmos SHA-2, oferecendo uma maior segurança e resistência a ataques criptográficos.

A família SHA-3 inclui vários tamanhos de saída, como SHA3-224, SHA3-256, SHA3-384 e SHA3-512. Esses algoritmos geram valores de hash de 224, 256, 384 e 512 bits, respectivamente. O SHA-3 é baseado em uma construção chamada Esponja (Sponge) que utiliza uma função de absorção e uma função de espremer.

A função de absorção do SHA-3 divide os dados de entrada em blocos de tamanho fixo e os absorve em um estado interno chamado de esponja. Em seguida, a função de espremer extrai o valor de hash do estado da esponja. Essa abordagem torna o SHA-3 resistente a diversos ataques criptográficos, incluindo ataques de diferenças e lineares.

O SHA-3 oferece uma maior segurança em relação ao SHA-2 e é recomendado para aplicações que exigem um alto nível de proteção criptográfica. Ele é amplamente utilizado em sistemas de autenticação, assinaturas digitais, sistemas de segurança em redes e outros cenários em que a integridade, autenticidade e confidencialidade dos dados são críticas.

Uma das vantagens do SHA-3 é a sua flexibilidade em termos de tamanho de saída, permitindo que se adapte às necessidades específicas de cada aplicação. Além disso, o SHA-3 é menos suscetível a ataques criptográficos conhecidos e oferece uma segurança de longo prazo. É o algoritmo de Função Hash mais seguro entre todos os apresentados nesta aula.

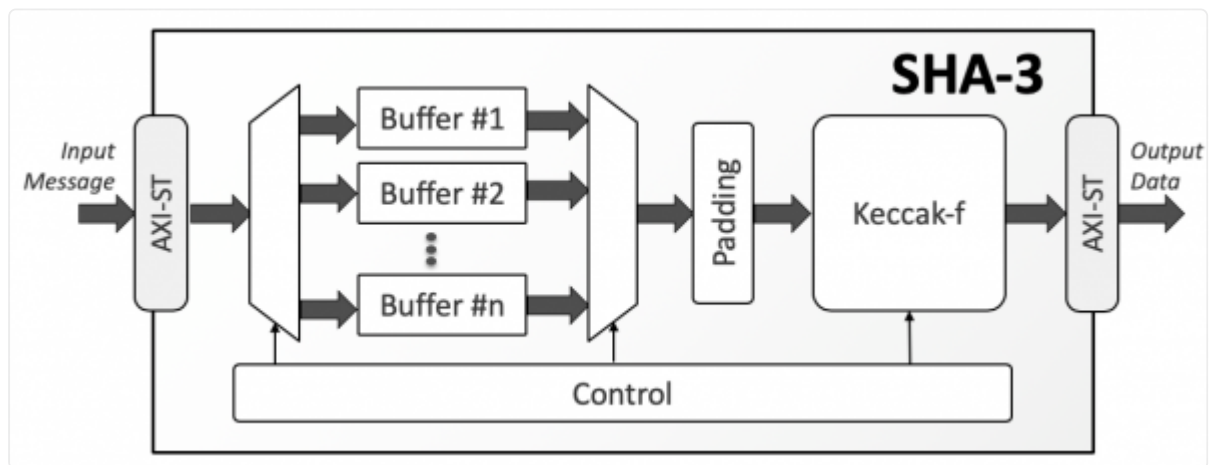


Diagrama de blocos do SHA-3.

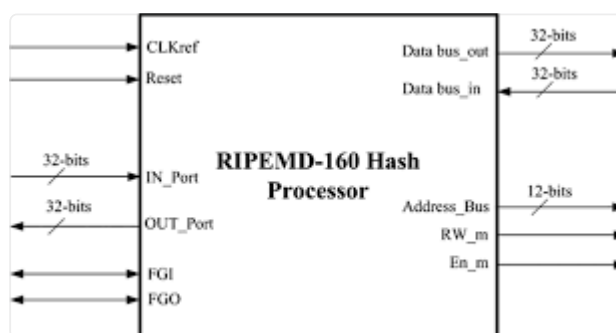
### **RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest-160)**

O algoritmo RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest-160) é um algoritmo de função hash criptográfica desenvolvido em 1996 por Hans Dobbertin, Antoon Bosselaers e Bart Preneel. Ele foi projetado como uma extensão do algoritmo RIPEMD original, com um tamanho de saída de 160 bits.

O objetivo principal do RIPEMD-160 é calcular um valor de hash único para um conjunto de dados de entrada. Ele opera em blocos de 512 bits e passa por várias etapas para processar os dados e gerar o valor de hash. Essas etapas incluem permutações, substituições não-lineares e combinações de bits utilizando funções lógicas e aritméticas.

O RIPEMD-160 foi desenvolvido como uma alternativa aos algoritmos SHA-1 e MD5, que demonstraram vulnerabilidades teóricas. Ele é amplamente utilizado em aplicações que exigem um valor de hash de tamanho moderado e uma boa resistência a colisões.

Uma das características do RIPEMD-160 é a sua capacidade de oferecer um nível aceitável de segurança em comparação com algoritmos mais recentes, como SHA-256. O RIPEMD-160 é utilizado em várias áreas, como criptografia, integridade de dados, assinaturas digitais e autenticação de arquivos. Ele continua sendo uma opção válida em casos onde um valor de hash de tamanho moderado é suficiente e não há requisitos específicos de segurança mais elevados.



Processador dedicado do RIPEMD-160.

## Blake2

O algoritmo Blake2 é uma família de algoritmos de função hash criptográfica que foi desenvolvido em 2012 por Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn e Christian Winnerlein. Ele foi projetado como uma evolução do algoritmo Blake, com melhor desempenho e segurança.

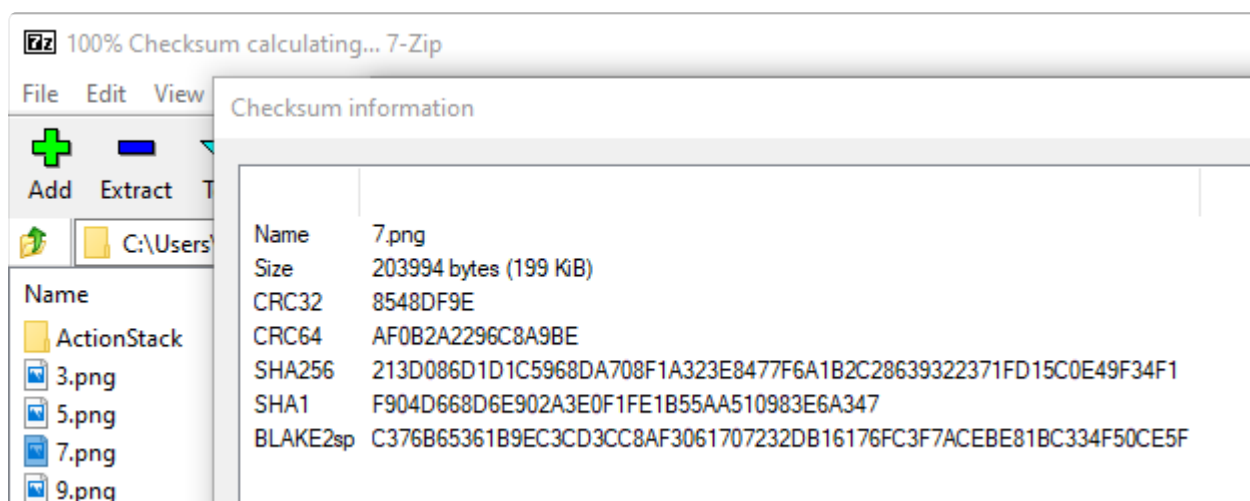
O Blake2 é altamente versátil, oferecendo diferentes versões com tamanhos de saída variáveis, incluindo Blake2b (com tamanho de saída de 512 bits) e Blake2s (com tamanho de saída de 256 bits). O algoritmo utiliza uma construção baseada em esponja (sponge construction), semelhante ao SHA-3, que permite absorver e espremer os dados de entrada.

Uma das principais características do Blake2 é seu desempenho excepcionalmente rápido. Ele é otimizado para processadores modernos e oferece velocidade de cálculo de hash significativamente mais rápida do que muitos outros algoritmos de

função hash. Além disso, ele possui uma implementação paralela eficiente, permitindo o processamento em paralelo de múltiplos blocos de dados.

Em termos de segurança, o Blake2 é considerado altamente resistente a ataques criptográficos conhecidos. Ele foi projetado para oferecer uma resistência robusta a colisões, pré-imagem e ataques diferenciais. O algoritmo possui propriedades de difusão e confusão bem equilibradas, que garantem a propagação das mudanças nos dados de entrada para o valor de hash gerado.

O Blake2 é amplamente utilizado em várias aplicações, como criptografia de dados, assinaturas digitais, autenticação de mensagens e integridade de arquivos. Sua eficiência e segurança tornaram-no uma escolha popular em ambientes que exigem um processamento de hash rápido e seguro.



Resultado de múltiplos algoritmos de Hash.

## Comparativo entre algoritmos de Funções Hash

Ao comparar os algoritmos de funções hash, é importante considerar vários aspectos, incluindo o tamanho dos resumos hash gerados pelos algoritmos. Vamos analisar alguns dos principais algoritmos e compará-los nessas categorias:

- **MD5 (Message Digest Algorithm 5):** O MD5 produz resumos hash de 128 bits, o que significa que o resultado do algoritmo é uma sequência de 128 bits que representa o dado original.
- **SHA-1 (Secure Hash Algorithm 1):** O SHA-1 gera resumos hash de 160 bits. Assim como o MD5, o resultado do algoritmo é uma sequência de 160 bits que

representa o dado original.

- **SHA-256 (Secure Hash Algorithm 256-bit):** O SHA-256 produz resumos hash de 256 bits, oferecendo um nível de segurança mais alto em comparação com o MD5 e o SHA-1.
- **SHA-3 (Secure Hash Algorithm 3):** O SHA-3 fornece diferentes tamanhos de resumos hash, incluindo 224, 256, 384 e 512 bits. A escolha do tamanho do resumo depende dos requisitos específicos de segurança e da aplicação.
- **BLAKE2:** O BLAKE2 oferece diferentes tamanhos de resumos hash, desde 1 até 512 bits, permitindo uma flexibilidade maior em relação ao tamanho do resumo desejado.

É importante observar que o tamanho dos resumos hash não é o único fator a ser considerado na escolha de um algoritmo de função hash. A segurança, a eficiência e a resistência a ataques criptográficos também são aspectos importantes a serem levados em conta.

## Conclusão

Em conclusão, as funções hash desempenham um papel fundamental na segurança e integridade dos dados. Elas fornecem um meio eficiente de gerar um valor de hash único para qualquer conjunto de dados, permitindo a verificação da integridade dos dados e detectando qualquer modificação ou corrupção. Além disso, as funções hash também são amplamente utilizadas em criptografia, autenticação e assinaturas digitais, fornecendo uma camada adicional de segurança.

Parabéns pela conclusão da aula de Funções Hash! Você demonstrou um excelente interesse e dedicação em aprender sobre esse importante conceito na área de segurança da informação. Ao entender os fundamentos das funções hash e suas características, você está adquirindo conhecimentos valiosos para garantir a integridade, autenticidade e segurança dos dados.

## Aula 4: Criptografia assimétrica



## Objetivos

- ☒ Compreender os princípios e conceitos fundamentais da criptografia assimétrica, incluindo chaves públicas e privadas.
- ☒ Explorar os algoritmos de criptografia assimétrica mais comuns, como RSA e ElGamal, e entender como eles são utilizados para proteger a comunicação e autenticar as partes envolvidas.
- ☒ Aprender a implementar corretamente a criptografia assimétrica em aplicações e sistemas, garantindo a confidencialidade, integridade e autenticidade dos dados transmitidos.

## Conceitos

- ☒ Chave pública e chave privada.
- ☒ Criptografia de chave assimétrica.
- ☒ Assinatura digital.

## Introdução

A aula de criptografia assimétrica é fundamental para compreender os princípios e técnicas avançadas de criptografia. Nessa área da segurança da informação, a criptografia assimétrica desempenha um papel essencial ao permitir a comunicação segura entre partes, garantindo a confidencialidade, a integridade e a autenticidade dos dados transmitidos. Durante essa aula, exploraremos os conceitos de chaves públicas e privadas, algoritmos de criptografia assimétrica e assinaturas digitais. Compreender esses conceitos é essencial para implementar soluções de segurança robustas e proteger informações sensíveis contra ameaças.

Durante a aula, iremos explorar as características únicas da criptografia assimétrica, que utiliza um par de chaves distintas para criptografar e descriptografar dados. Essa abordagem oferece uma camada adicional de segurança ao permitir a distribuição da chave pública para qualquer pessoa, enquanto mantém a chave privada em posse exclusiva do proprietário. Através da



criptografia assimétrica, as partes podem se comunicar de forma segura sem compartilhar a chave secreta, garantindo a confidencialidade dos dados transmitidos.

## Funcionamento da criptografia assimétrica

### Componentes

A criptografia assimétrica, também conhecida como criptografia de chave pública, utiliza um par de chaves distintas para criptografar e descriptografar dados. Esse par de chaves consiste em uma chave pública e uma chave privada. Vamos explorar como funciona o processo de criptografia assimétrica:

- **Geração do par de chaves:** O primeiro passo é gerar o par de chaves, composto pela chave pública e pela chave privada. Essas chaves são matematicamente relacionadas, mas computacionalmente inviáveis de serem derivadas uma da outra. A chave pública é divulgada publicamente, enquanto a chave privada é mantida em sigilo pelo proprietário. Cada parte da comunicação (emissor e receptor) pode ter um par de chaves pública e privada.



Chaves de Alice.

- **Criptografia:** Para enviar uma mensagem segura para um destinatário, o remetente utiliza uma das chaves (pública ou privada) para criptografar a mensagem. Isso é feito aplicando uma função matemática específica à mensagem original, juntamente com a chave usada. O resultado é a mensagem criptografada, que só pode ser descriptografada com a outra chave (privada ou pública) correspondente.

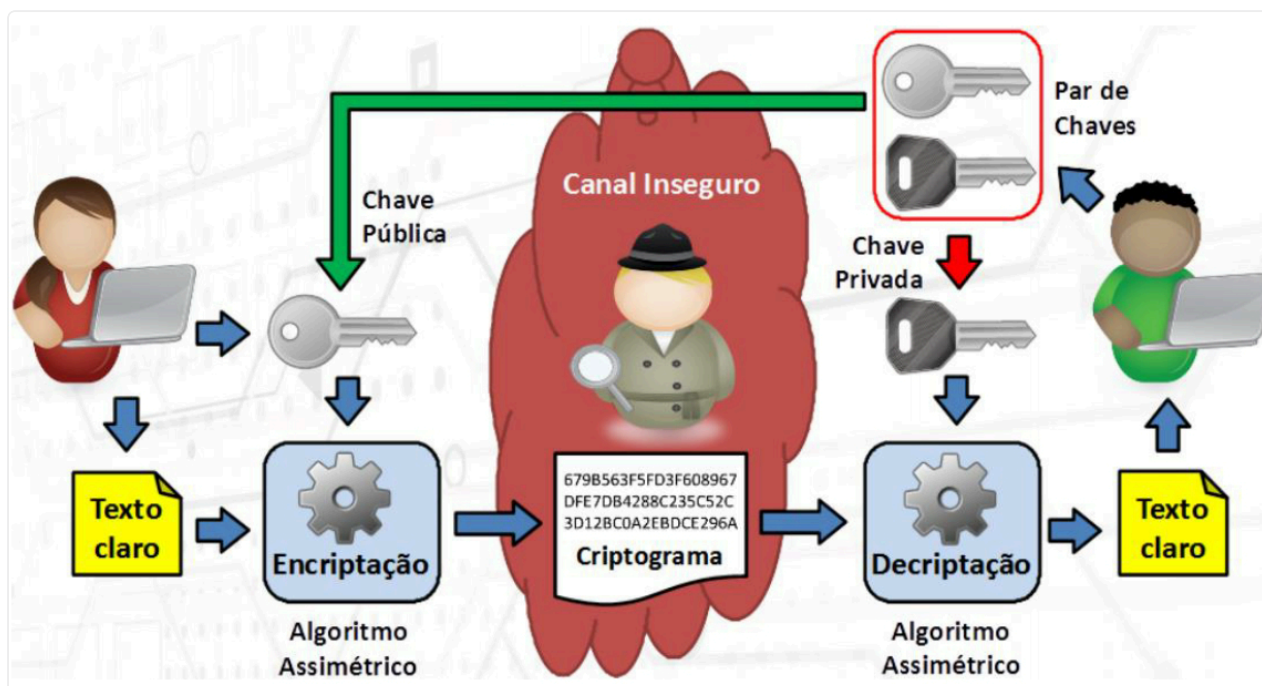
- **Transmissão da mensagem criptografada:** A mensagem criptografada é transmitida e somente o destinatário que possui a chave correspondente poderá descriptografá-la.
- **Descriptografia:** Ao receber a mensagem criptografada, o destinatário utiliza a chave correspondente descriptografar a mensagem, o que resulta na recuperação da mensagem original.

## Criptografia assimétrica para confidencialidade

A criptografia assimétrica é usada para garantir a confidencialidade dos dados durante a transmissão entre duas partes. Vamos entender como os dados são cifrados usando criptografia assimétrica para confidencialidade, utilizando um exemplo envolvendo Alice e Bob:

1. **Geração do par de chaves:** Bob gera um par de chaves composto por uma chave pública (PubBob) e uma chave privada (PrivBob). A chave pública é compartilhada publicamente, enquanto a chave privada é mantida em sigilo por Bob.
2. **Cifragem dos dados por Alice:** Alice deseja enviar dados confidenciais para Bob. Ela obtém a chave pública de Bob (PubBob) e utiliza essa chave para cifrar os dados. Alice aplica um algoritmo de criptografia assimétrica à mensagem original, produzindo uma versão cifrada da mensagem.
3. **Transmissão dos dados cifrados:** Alice envia os dados cifrados para Bob. Durante a transmissão, mesmo que um terceiro (como Eve ou Mallory) intercepte a mensagem cifrada, ele não poderá decifrá-la sem acesso à chave privada correspondente (PrivBob) de Bob.
4. **Decifragem dos dados por Bob:** Ao receber os dados cifrados de Alice, Bob utiliza sua chave privada (PrivBob) para decifrar a mensagem. Ele aplica um algoritmo de descriptografia assimétrica à mensagem cifrada utilizando sua chave privada, o que resulta na recuperação da mensagem original enviada por Alice.

Dessa forma, a confidencialidade dos dados é garantida, pois somente Bob, o destinatário legítimo com a chave privada, pode decifrar os dados criptografados. Os dados permanecem protegidos durante a transmissão, mesmo que sejam interceptados por terceiros.



Criptografia assimétrica para confidencialidade.

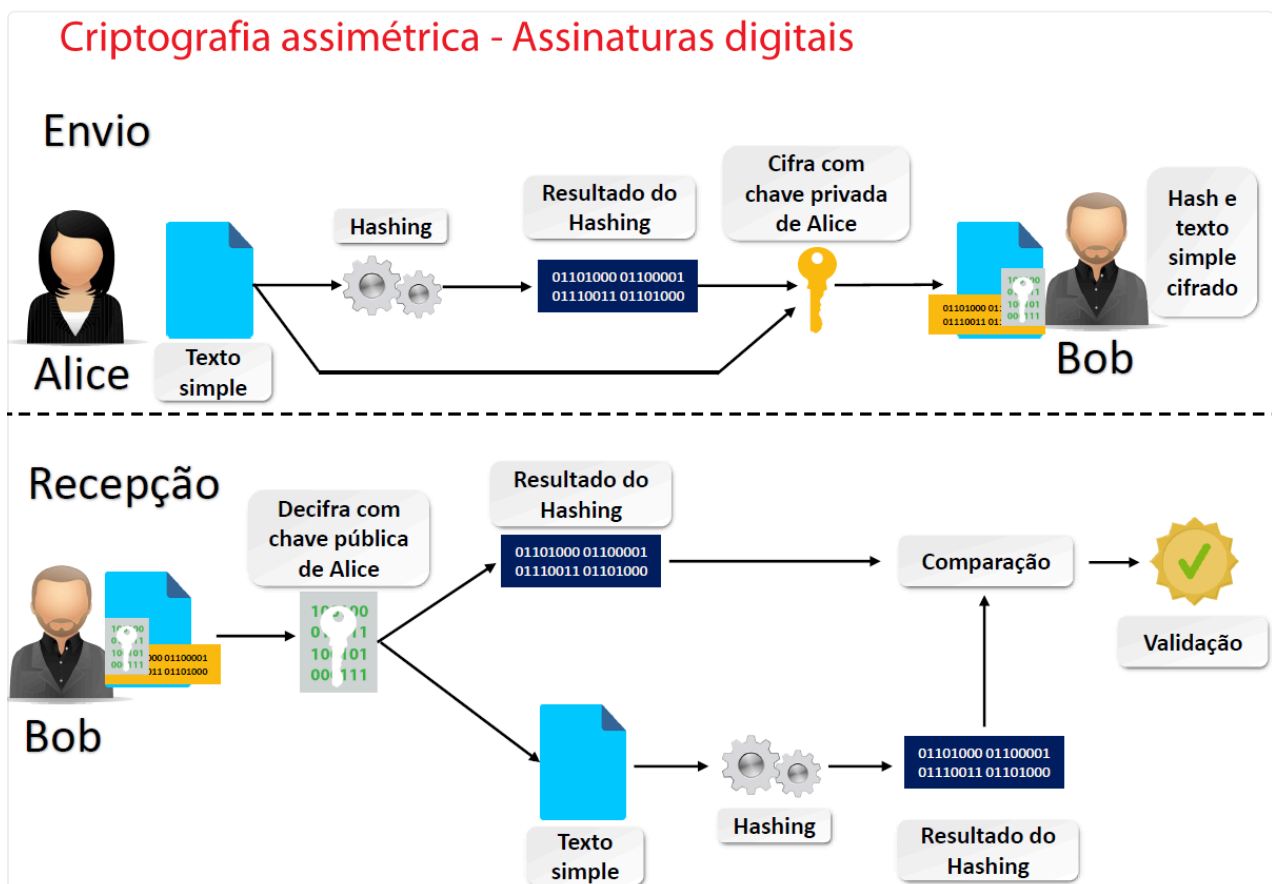
### Criptografia assimétrica para assinaturas digitais

As assinaturas digitais são mecanismos utilizados na criptografia para garantir a autenticidade e integridade de dados digitais. Por meio de técnicas criptográficas assimétricas, uma assinatura digital é criada pelo remetente, associada aos dados originais. Essa assinatura digital é única e vinculada ao remetente, sendo verificada pelo destinatário utilizando a chave pública correspondente. A verificação da assinatura permite ao destinatário confirmar a autenticidade do remetente e a integridade dos dados, certificando-se de que a mensagem não foi alterada durante a transmissão. As assinaturas digitais são amplamente utilizadas em transações eletrônicas, contratos digitais e outros cenários que requerem autenticação e confiança nos dados transmitidos.

A criptografia assimétrica utilizada para criar assinaturas digitais garante a autenticidade e integridade dos dados. Vamos entender como os dados são cifrados usando criptografia assimétrica para assinaturas digitais, utilizando um exemplo envolvendo Alice:

1. **Geração do par de chaves:** Alice gera um par de chaves composto por uma chave pública (PubAlice) e uma chave privada (PrivAlice). A chave pública é compartilhada publicamente, enquanto a chave privada é mantida em sigilo por Alice.

2. **Criação da assinatura digital por Alice:** Alice deseja disponibilizar um documento (pode ser qualquer arquivo) assinado digitalmente por ela. Ela aplica uma função hash ao documento original para gerar um resumo criptográfico único e fixo. Em seguida, Alice utiliza sua chave privada (PrivAlice) para criptografar o resumo e o documento, formando o arquivo assinado digitalmente.
3. **Transmissão dos dados assinados:** Alice disponibiliza o arquivo assinado digitalmente. Qualquer pessoa pode ter acesso ao arquivo.
4. **Verificação da assinatura:** Ao receber o documento e a assinatura digital de Alice, a parte que vai verificar o arquivo assinado digitalmente (por exemplo Bob), utiliza a chave pública de Alice (PubAlice) para descriptografar a assinatura. Isso resulta na obtenção do documento digital e o resumo criptográfico original (da função hash). Bob, então, aplica a mesma função hash ao documento digital e compara o resumo gerado por ele com o resumo descriptografado que estava no arquivo assinado digitalmente por Alice. Se forem idênticos, a integridade do documento é confirmada e a assinatura digital é considerada válida, garantindo que foi Alice quem assinou o documento.



Criptografia assimétrica para assinaturas digitais.

Dessa forma, a criptografia assimétrica é utilizada para criar uma assinatura digital única que está vinculada aos dados originais. A assinatura digital permite que Bob (ou qualquer outra pessoa) verifique a autenticidade do remetente e a integridade dos dados. Caso a assinatura seja inválida, indica que a mensagem pode ter sido modificada durante a transmissão ou que não foi enviada por Alice.

## **Desafios e considerações na implementação da criptografia assimétrica**

A implementação da criptografia assimétrica apresenta alguns desafios e considerações que devem ser levados em conta:

- **Desempenho:** A criptografia assimétrica é computacionalmente mais intensiva do que a criptografia simétrica. Os algoritmos assimétricos envolvem operações matemáticas complexas, como exponenciação modular e multiplicação de grandes números. Portanto, é necessário considerar o desempenho do sistema ao implementar algoritmos assimétricos, especialmente em cenários de grande volume de dados ou em dispositivos com recursos limitados.
- **Gerenciamento de chaves:** A criptografia assimétrica requer o gerenciamento adequado das chaves públicas e privadas. É essencial garantir a segurança das chaves privadas, pois a divulgação indevida delas pode comprometer a segurança dos dados. Além disso, é necessário estabelecer mecanismos confiáveis para distribuir e armazenar as chaves públicas de forma que os usuários possam verificá-las corretamente.
- **Algoritmos e padrões confiáveis:** A escolha do algoritmo criptográfico é crucial para garantir a segurança da implementação. É necessário selecionar algoritmos amplamente aceitos, com revisões e análises independentes, que tenham resistência comprovada a ataques criptográficos conhecidos. Além disso, seguir padrões e protocolos bem estabelecidos é importante para interoperabilidade e segurança entre diferentes sistemas.
- **Criptografia híbrida:** Para combinar eficiência e segurança, é comum utilizar a criptografia assimétrica em conjunto com a criptografia simétrica. Isso é conhecido como criptografia híbrida. Nesse caso, a criptografia assimétrica é usada para estabelecer uma chave de sessão segura, que é então usada para criptografar os dados de forma eficiente com a criptografia simétrica. A implementação adequada dessa abordagem requer a sincronização adequada dos algoritmos, o gerenciamento correto das chaves e a proteção das chaves de sessão.
-



**Atualização e segurança:** A criptografia assimétrica também está sujeita a avanços na criptoanálise e descoberta de novas vulnerabilidades. Portanto, é essencial acompanhar os desenvolvimentos e atualizações na área de criptografia para garantir a segurança contínua dos sistemas implementados.



Chaves em criptografia.

## Algoritmos de criptografia assimétrica

### RSA (Rivest, Shamir, Adleman)

O algoritmo RSA, desenvolvido por Ron Rivest, Adi Shamir e Leonard Adleman em 1977, é um dos algoritmos de criptografia assimétrica mais amplamente utilizados. Ele se baseia na dificuldade de fatorar grandes números inteiros para fornecer segurança na troca de informações. O funcionamento do algoritmo RSA é o seguinte:

1. **Geração do par de chaves:** Primeiro, é gerado um par de chaves composto por uma chave pública e uma chave privada. A chave pública é compartilhada com todos, enquanto a chave privada é mantida em sigilo pelo proprietário.
2. **Escolha de primos:** Em seguida, são escolhidos dois números primos grandes distintos,  $p$  e  $q$ . Esses números são mantidos em segredo.
3. **Cálculo do módulo  $n$ :** O módulo  $n$  é calculado como o produto dos primos  $p$  e  $q$ , ou seja,  $n = p * q$ . Esse valor é usado como o módulo nas operações criptográficas.

4. **Cálculo da função totiente de Euler:** A função totiente de Euler ( $\varphi$ ) de  $n$  é calculada como  $\varphi(n) = (p - 1) * (q - 1)$ . Essa função é importante para determinar a chave privada.
5. **Escolha da chave pública:** Um número inteiro e relativamente primo a  $\varphi(n)$  é escolhido como a chave pública, geralmente chamado de "e". A chave pública consiste no par  $(n, e)$  e é compartilhada publicamente.
6. **Cálculo da chave privada:** A chave privada, geralmente chamada de "d", é calculada como o inverso multiplicativo de "e" módulo  $\varphi(n)$ . Essa chave privada é mantida em segredo.
7. **Criptografia:** Para criptografar uma mensagem, o remetente a converte em um número inteiro representativo do espaço de mensagens. Em seguida, ele eleva esse número à potência "e" módulo  $n$  para obter o texto cifrado.
8. **Descriptografia:** O destinatário utiliza a chave privada "d" para descriptografar o texto cifrado, elevando-o à potência "d" módulo  $n$ . O resultado é a mensagem original.

O tamanho das chaves seguras no algoritmo RSA depende da aplicação e do período de segurança desejado. Normalmente, são utilizadas chaves com tamanho de 2048 bits ou mais, consideradas seguras para a maioria dos cenários. O algoritmo RSA é amplamente utilizado em diversas tecnologias, como:

- **Criptografia de dados:** O RSA é usado para proteger a confidencialidade de dados sensíveis em comunicações seguras, como trocas de chaves em TLS/SSL, criptografia de emails e assinaturas digitais.
- **Criptografia de chave pública:** O RSA é usado para proteger chaves de criptografia simétrica em sistemas de gerenciamento de chaves, como PGP (Pretty Good Privacy) e SSH (Secure Shell).
- **Autenticação:** O RSA é usado para autenticar entidades em sistemas de autenticação de dois fatores, como tokens RSA SecurID.

O algoritmo RSA fornece uma base sólida para a criptografia assimétrica e continua sendo uma ferramenta essencial na segurança da informação.

## **DSA (Digital Signature Algorithm)**

O algoritmo DSA (Digital Signature Algorithm) é um algoritmo de assinatura digital, uma forma de criptografia assimétrica, desenvolvido pelo Governo dos Estados



Unidos. Ele é amplamente utilizado para garantir a autenticidade e integridade de mensagens e documentos digitais. O funcionamento do algoritmo DSA é o seguinte:

1. **Geração do par de chaves:** Assim como em outros algoritmos de criptografia assimétrica, o DSA utiliza um par de chaves composto por uma chave pública e uma chave privada. A chave pública é divulgada publicamente, enquanto a chave privada é mantida em segredo.
2. **Escolha de parâmetros:** Antes de gerar as chaves, são escolhidos os parâmetros do algoritmo, que incluem um número primo grande ( $p$ ), um número primo menor ( $q$ ) que é um divisor de  $p-1$ , e um gerador ( $g$ ) que é uma raiz primitiva módulo  $p$ . Esses parâmetros são fixos para um conjunto de chaves e devem ser compartilhados entre o emissor e o receptor.
3. **Geração da chave privada:** A chave privada é gerada selecionando aleatoriamente um número inteiro  $x$ , que deve ser mantido em segredo pelo proprietário.
4. **Cálculo da chave pública:** A chave pública é calculada como  $y = g^x \bmod p$ , onde " $^$ " representa a operação de exponenciação. O valor de  $y$  é a chave pública correspondente à chave privada  $x$ .
5. **Criação da assinatura digital:** Para criar uma assinatura digital em uma mensagem, o emissor utiliza sua chave privada para calcular um valor chamado "assinatura". Isso é feito selecionando aleatoriamente um número inteiro  $k$  e calculando  $r = (g^k \bmod p) \bmod q$ , onde  $r$  é um componente da assinatura.
6. **Verificação da assinatura:** O receptor recebe a mensagem e a assinatura digital correspondente. Ele utiliza a chave pública do emissor para calcular dois valores,  $s_1$  e  $s_2$ . O valor  $s_1$  é calculado como  $s_1 = (h(m) / r) \bmod q$ , onde  $h(m)$  é o resumo criptográfico (hash) da mensagem. O valor  $s_2$  é calculado como  $s_2 = (k^{-1} * (h(m) + x * s_1)) \bmod q$ , onde  $k^{-1}$  é o inverso multiplicativo de  $k$  módulo  $q$ . Se os valores de  $s_1$  e  $s_2$  coincidirem com a assinatura original, a assinatura é considerada válida.

O tamanho das chaves seguras no algoritmo DSA é geralmente de 1024 bits ou mais, dependendo dos requisitos de segurança. O DSA é amplamente utilizado em tecnologias como:

- **Criptografia de emails:** O DSA é utilizado para assinar digitalmente mensagens de emails, garantindo a autenticidade do remetente e a integridade da

mensagem.

- **Certificados digitais:** O DSA é usado para criar assinaturas digitais em certificados digitais, que são utilizados em infraestruturas de chaves públicas (PKIs) para autenticação e segurança em comunicações online.
- **Protocolos de segurança:** O DSA é utilizado em diversos protocolos de segurança, como o protocolo de segurança de camada de transporte (TLS/SSL) para estabelecer conexões seguras na Internet.

O algoritmo DSA oferece um mecanismo eficiente e seguro para a geração de assinaturas digitais, garantindo a autenticidade e integridade de mensagens e documentos digitais.

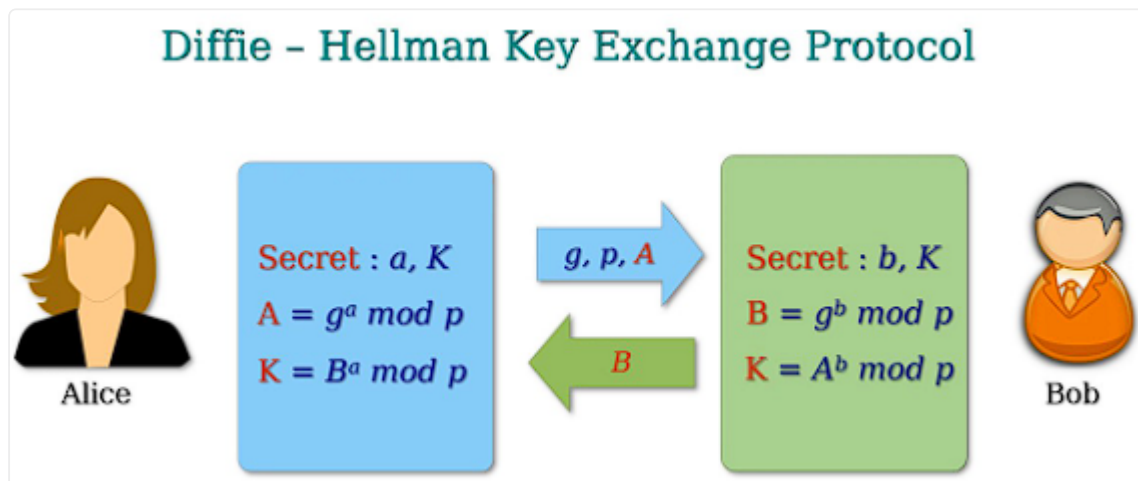
## Diffie-Hellman

O algoritmo Diffie-Hellman é um protocolo de troca de chaves que permite que duas partes estabeleçam uma chave secreta compartilhada, mesmo que estejam se comunicando por um canal inseguro. Ele foi desenvolvido por Whitfield Diffie e Martin Hellman em 1976 e é amplamente utilizado em criptografia de chave pública. O funcionamento do algoritmo Diffie-Hellman é o seguinte:

1. **Escolha dos parâmetros:** Antes de iniciar a troca de chaves, as partes devem concordar com os parâmetros do algoritmo. Isso inclui a escolha de um número primo grande ( $p$ ) e um número gerador ( $g$ ), que é um elemento do grupo multiplicativo módulo  $p$ . Esses parâmetros são fixos para um conjunto de chaves e devem ser compartilhados entre as partes.
2. **Geração das chaves privadas:** Cada parte gera sua própria chave privada, que é um número aleatório e exclusivo para cada parte. Vamos chamar essas chaves privadas de " $a$ " e " $b$ ".
3. **Cálculo das chaves públicas:** Cada parte calcula sua chave pública, que é obtida elevando o número gerador " $g$ " à potência da chave privada correspondente e realizando a operação de módulo " $p$ ". Ou seja, a chave pública de Alice é  $A = g^a \bmod p$  e a chave pública de Bob é  $B = g^b \bmod p$ .
4. **Troca das chaves públicas:** Alice e Bob trocam suas chaves públicas pela rede insegura.
5. **Cálculo da chave secreta compartilhada:** Utilizando a chave pública recebida e sua própria chave privada, cada parte realiza o cálculo da chave secreta compartilhada. Alice calcula a chave secreta compartilhada como  $S = B^a \bmod p$ .

p, enquanto Bob calcula a chave secreta compartilhada como  $S = A^b \bmod p$ . Ambos chegam ao mesmo resultado, a chave secreta compartilhada S.

A segurança do algoritmo Diffie-Hellman está baseada na dificuldade do problema do logaritmo discreto. É computacionalmente inviável para um atacante calcular a chave secreta compartilhada S apenas conhecendo as chaves públicas e os parâmetros p e g.



Chaves no Diffie-Hellman.

O tamanho das chaves seguras no algoritmo Diffie-Hellman é geralmente de 2048 bits ou mais, dependendo dos requisitos de segurança. O Diffie-Hellman é utilizado em várias tecnologias, como:

- **Protocolo TLS/SSL:** O Diffie-Hellman é utilizado no protocolo de segurança de transporte (TLS/SSL) para estabelecer chaves compartilhadas entre um servidor e um cliente, permitindo a comunicação segura pela internet.
- **Protocolo SSH:** O Diffie-Hellman é usado no protocolo de acesso remoto seguro (SSH) para estabelecer chaves compartilhadas entre um cliente e um servidor, garantindo a autenticação e confidencialidade das comunicações.
- **VPN:** Em redes privadas virtuais (VPNs), o Diffie-Hellman é empregado para estabelecer chaves compartilhadas entre os dispositivos, permitindo a comunicação segura e a criação de túneis criptografados.
- **Criptografia de mensagens:** O Diffie-Hellman é utilizado para a troca de chaves em criptografia de chave simétrica, onde as chaves de criptografia são geradas de forma segura e compartilhadas entre as partes.

O algoritmo Diffie-Hellman é uma ferramenta fundamental na área de criptografia, permitindo a troca segura de chaves em ambientes inseguros, contribuindo para a confidencialidade e privacidade das comunicações.

## ElGamal

O algoritmo ElGamal é um algoritmo de criptografia assimétrica que combina a criptografia de chave pública e a troca de chaves de Diffie-Hellman. Ele é amplamente utilizado para garantir a confidencialidade das comunicações e proteger a privacidade dos dados. O funcionamento do algoritmo ElGamal é o seguinte:

1. **Geração do par de chaves:** Assim como em outros algoritmos de criptografia assimétrica, o ElGamal utiliza um par de chaves composto por uma chave pública e uma chave privada. A chave pública é divulgada publicamente, enquanto a chave privada é mantida em segredo.
2. **Escolha de parâmetros:** Antes de gerar as chaves, são escolhidos os parâmetros do algoritmo, que incluem um número primo grande ( $p$ ) e um gerador ( $g$ ) que é um elemento do grupo multiplicativo módulo  $p$ . Esses parâmetros são fixos para um conjunto de chaves e devem ser compartilhados entre o emissor e o receptor.
3. **Geração da chave privada:** A chave privada é gerada selecionando aleatoriamente um número inteiro  $x$ , que deve ser mantido em segredo pelo proprietário.
4. **Cálculo da chave pública:** A chave pública é calculada como  $y = g^x \bmod p$ , onde  $^$  representa a operação de exponenciação. O valor de  $y$  é a chave pública correspondente à chave privada  $x$ .
5. **Criptografia:** Para criptografar uma mensagem, o emissor seleciona aleatoriamente um número inteiro  $k$  e calcula dois valores. O primeiro valor é  $r = g^k \bmod p$ , que é o componente de aleatoriedade. O segundo valor é  $c = (m * y^k) \bmod p$ , onde  $m$  é o valor numérico da mensagem original. O par  $(r, c)$  é a mensagem cifrada que será enviada ao receptor.
6. **Descritografia:** O receptor recebe a mensagem cifrada  $(r, c)$  e utiliza sua chave privada  $x$  para calcular o valor inverso de  $r$ , que é  $r^{-x} \bmod p$ . Em seguida, ele recupera a mensagem original  $m$  utilizando a fórmula  $m = (c * r^{-x}) \bmod p$ .

O tamanho das chaves seguras no algoritmo ElGamal geralmente é de 2048 bits ou mais, dependendo dos requisitos de segurança. O ElGamal é utilizado em tecnologias como:

- **Criptografia de emails:** O ElGamal é usado para criptografar emails, garantindo a confidencialidade das mensagens durante a transmissão.
- **Compartilhamento seguro de chaves:** O ElGamal é aplicado em protocolos de compartilhamento seguro de chaves, como o protocolo Diffie-Hellman, para estabelecer chaves compartilhadas entre duas partes sem que a chave seja exposta durante a troca.
- **Criptografia de arquivos:** O ElGamal é empregado em sistemas de criptografia de arquivos para proteger a privacidade e a confidencialidade dos dados armazenados.

O algoritmo ElGamal oferece uma abordagem segura e eficiente para criptografia de chave pública, fornecendo confidencialidade e privacidade na comunicação.



ElGamal.

### **ECC (Elliptic Curve Cryptography)**

O algoritmo ECC (Elliptic Curve Cryptography) é um sistema criptográfico que se baseia na teoria das curvas elípticas sobre corpos finitos. Nesse algoritmo, a chave pública e a chave privada são geradas com base em operações matemáticas na curva elíptica. A curva elíptica utilizada é definida por uma equação matemática específica.



A forma geral da equação de uma curva elíptica é:  $y^2 = x^3 + ax + b$ , onde  $a$  e  $b$  são constantes que determinam a forma e a posição da curva. A escolha adequada dos parâmetros  $a$  e  $b$  é fundamental para garantir a segurança e a eficiência do algoritmo ECC.

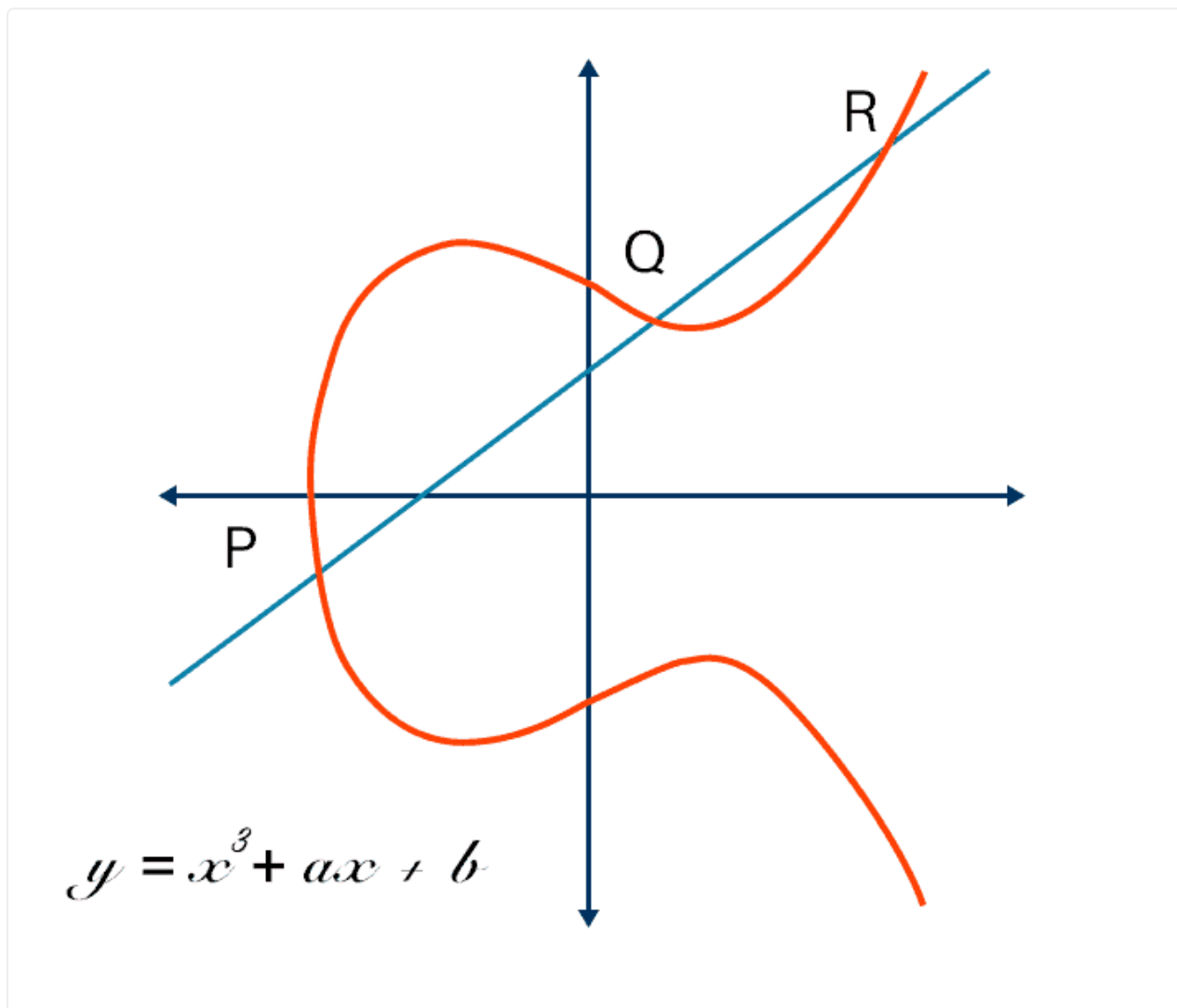


Gráfico de ECC.

Uma característica importante das curvas elípticas é que elas possuem um grupo aditivo associado a elas. Essa propriedade permite realizar operações matemáticas, como adição e multiplicação, entre pontos na curva. Essas operações são utilizadas na geração de chaves e no processo de criptografia e descryptografia.

No contexto do ECC, a segurança é baseada na complexidade do problema matemático conhecido como "problema do logaritmo discreto". Esse problema envolve encontrar o valor de um expoente desconhecido quando dado uma base e o resultado da exponenciação. A complexidade desse problema torna o ECC

resistente a ataques de força bruta e algoritmos de fatoração.. Diferente de outros algoritmos de criptografia assimétrica, como o RSA e o DSA, o ECC oferece um nível de segurança similar com chaves significativamente menores. O funcionamento do algoritmo ECC é o seguinte:

1. **Escolha da curva elíptica:** Antes de iniciar a geração de chaves, é necessário escolher uma curva elíptica adequada. Existem várias curvas elípticas disponíveis, cada uma com diferentes níveis de segurança. As curvas mais comumente utilizadas são definidas sobre corpos finitos primos.
2. **Geração das chaves privada e pública:** Assim como em outros algoritmos de criptografia assimétrica, cada parte gera sua própria chave privada, que é um número aleatório e exclusivo para cada parte. A chave privada é geralmente um número inteiro gerado dentro de um intervalo seguro. A chave pública é obtida multiplicando a chave privada pelo ponto gerador da curva elíptica.
3. **Troca das chaves públicas:** As partes enviam suas chaves públicas para a outra parte de forma segura.
4. **Cálculo da chave secreta compartilhada:** Utilizando a chave pública recebida e sua própria chave privada, cada parte realiza o cálculo da chave secreta compartilhada. Isso envolve a multiplicação da chave pública pela chave privada da outra parte.

O ECC oferece uma segurança equivalente a outros algoritmos de criptografia assimétrica, mas com chaves muito menores. Em geral, uma chave ECC de 256 bits é considerada segura o suficiente para a maioria das aplicações, enquanto um nível de segurança semelhante com outros algoritmos requer chaves de 2048 bits ou mais.

O algoritmo ECC é utilizado em várias tecnologias, como:

- **Criptografia de curva elíptica em TLS/SSL:** O ECC é suportado como um método de criptografia em protocolos de segurança de transporte (TLS/SSL), oferecendo uma alternativa mais eficiente e segura em comparação com algoritmos tradicionais.
- **Smart cards e dispositivos de segurança:** Devido ao seu baixo consumo de energia e requisitos de armazenamento, o ECC é amplamente utilizado em smart cards e outros dispositivos de segurança para autenticação e assinatura digital.
-



**Internet das Coisas (IoT):** O ECC é uma escolha popular para criptografia em dispositivos IoT devido aos seus requisitos de recursos reduzidos e capacidade de oferecer segurança adequada mesmo em dispositivos de baixo poder de processamento.

O ECC se tornou uma alternativa viável e eficiente para algoritmos de criptografia assimétrica, oferecendo segurança sólida com tamanhos de chaves menores, o que é especialmente importante em ambientes com restrições de recursos.

A seguinte imagem mostra um comparativo entre criptografia simétrica e assimétrica.

Criptografia Simétrica	Criptografia Assimétrica
Desempenho superior (mais rápida)	Desempenho inferior (mais lenta)
Sem autenticação → sem mecanismo	Com autenticação → assinaturas digitais
Não é irrefutável (autoria pode ser negada)	Assinatura digital é irrefutável
Distribuição de muitas chaves é trabalhosa	Distribuição de chaves simplificada
Qualquer um com chave compartilhada tem acesso	Acesso à informação é restrito ao proprietário da chave privada

Criptografia simétrica x assimétrica.

**Comparativo entre algoritmos de assimétrica**

Ao comparar os algoritmos de criptografia assimétrica, é importante considerar diversos aspectos, como a segurança das chaves, o tamanho das chaves necessárias e o desempenho dos algoritmos. Vamos analisar alguns dos principais algoritmos e compará-los nessas categorias:

**RSA:**

- Tamanho das chaves seguras: O tamanho mínimo recomendado para chaves RSA é de 2048 bits para garantir um nível de segurança adequado. No entanto, para proteção a longo prazo, chaves de 3072 bits ou até maiores podem ser recomendadas.
- Desempenho: O RSA é conhecido por ser um algoritmo computacionalmente intensivo, principalmente quando se trabalha com chaves maiores. O processo

de criptografia e descryptografia pode ser lento em comparação com outros algoritmos.

### **DSA (Digital Signature Algorithm):**

- Tamanho das chaves seguras: O DSA requer chaves com tamanhos de 1024 a 3072 bits para garantir segurança adequada. Recomenda-se o uso de tamanhos de chave maiores para proteção a longo prazo.
- Desempenho: O DSA é geralmente mais rápido que o RSA em termos de velocidade de assinatura e verificação de assinatura. No entanto, seu desempenho pode ser mais lento quando se trata de geração de chaves.

### **ECC (Elliptic Curve Cryptography):**

- Tamanho das chaves seguras: O ECC oferece um nível de segurança equivalente a outros algoritmos com tamanhos de chaves muito menores. Por exemplo, uma chave ECC de 256 bits é considerada segura, enquanto um nível de segurança semelhante com o RSA requer chaves de 2048 bits ou mais.
- Desempenho: O ECC é conhecido por seu desempenho eficiente, especialmente em dispositivos com recursos limitados. Ele oferece uma execução mais rápida em comparação com o RSA e o DSA, especialmente em operações como geração de chaves, criptografia e descryptografia.

É importante ressaltar que a segurança não se resume apenas ao tamanho da chave. A robustez e a resistência dos algoritmos a ataques criptográficos também são fatores cruciais a serem considerados. Além disso, a implementação correta e segura dos algoritmos é fundamental para garantir a proteção adequada dos dados.

## **Conclusão**

Na conclusão da aula de criptografia assimétrica, podemos afirmar que essa técnica desempenha um papel crucial na segurança da comunicação e na proteção dos dados. Ao utilizar chaves públicas e privadas, a criptografia assimétrica oferece recursos como confidencialidade, autenticidade e integridade dos dados. Os algoritmos, como RSA, DSA, ElGamal, Diffie-Hellman e ECC, fornecem soluções flexíveis e eficientes para diferentes necessidades de segurança. É importante

compreender as características, os desafios e as aplicações desses algoritmos para garantir uma implementação adequada. Com a criptografia assimétrica, é possível estabelecer comunicações seguras, realizar assinaturas digitais confiáveis e proteger informações sensíveis, contribuindo para a segurança na era digital.

Parabéns, aluno, por concluir com sucesso a aula de criptografia assimétrica! Você demonstrou um excelente entendimento dos conceitos e dos principais algoritmos utilizados nessa área. A criptografia assimétrica desempenha um papel fundamental na segurança da informação e na proteção dos dados em diversos cenários. Seu empenho e dedicação em aprender sobre esse tema tão importante certamente serão valiosos para sua jornada na área de segurança da informação. Continue a explorar e aprofundar seus conhecimentos nesse campo fascinante.