1.Introdução ao hashing

1.1 Característica de 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.

1.2 Resistência a tentativa de invasão

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.

2.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.

2.1 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.

2.2 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.

3.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.

4.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.

5.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.

6.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.

7.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.

8.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.

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.

9.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.

10.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.

10.1 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.

10.2 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.

10.3 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.

10.4 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 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.

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.

10.5 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.

10.6 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.

A tabela abaixo lista alguns hashes criptográficos comuns, protocolos e algoritmos.

Integridade	Autenticidade	Confidencialidade
MD5 (legado)	HMAC-MD5 (legado)	3DES (legado)
SHA	HMAC-SHA-256	AES
	RSA e DSA	