Hash Functions

Teórica #5 de Criptografia Aplicada

O que é uma função hash?

Função que dado um input, retorna uma string codificada que não pode ser revertida.

Quais as suas aplicações?

- Derivação de chaves;
- Authentication digest;
- Extração de aleatoriedade;
- Proteção de passwords;
- Provas de trabalho (proofs-of-work);
- Indexação de ficheiros em controlos de versões (e.g., Git);
- Verificação de integridade dos ficheiros relativo a deteção de intrusos (e.g., HMAC).

Tipicamente tem como tamanho de output 256 e 512 bits.

Também conhecido como **hash**, **fingerprint** e **digest**.

Funções de hash não verificam a propriedade da injetividade porque como comprimem o valor do output num tamanho diferente, significa que pode haver **n** inputs que **geram o mesmo output**.

O que faz uma função hash segura?

A melhor função hash possível satisfaz o modelo de ***Random Oracle***: dado um input, a função hash retorna a hash do input que é única no espaço das hashes, significado que não existem inputs diferentes que level ao mesmo hash (propriedade da injetividade).

Para tal, começa-esse com uma tabela vazia que vai armazenando os valores de input e seus outputs. Se um input ainda não existir na tabela, calcula-se o hash e retorna-se o mesmo, caso contrário retorna-se o hash existente.

Random Oracle não é possivel na prática porque para tal era necessário armazenar uma quantidade enorme de registos.

Na prática as melhores funções de hash são aquelas que respondem aos requisitos que definimos, segundo propriedades de segurança interessantes:

- Outputs que são **imprevisíveis**;
- Difícil de encontrar pré-imagens (diferentes inputs = mesmo output);
- Difícil de encontrar colisões.

QUAL A DIFERENÇA ENTRE ENCONTRAR COLISÕES E ENCONTRAR PRÉ-IMAGENS???

Para tal, funções de hash são descritas e validadas heuristicamente (similar aos testes do AES):

- Competições internacionais para selecionar as funções com melhor design;
- Competidores (cryptanalysis) são criticados e refutam as funções dos outros, em termos de **segurança** e **performance**;
- Iterativamente (através de rondas), vai-se refinando e escolhendo a função de hash ideal perante a opinião da comunidade (cryptanalysts).

Função de hash standard mais recente: **SHA-3**.

Propriedade: Resistência a pre-imagens (Difícil de Inverter)

Considera-se *S* o conjunto de pre-imagens (domínio) e *R* o conjunto das imagens (contradomínio): Dado um valor *Y* (hash) que pertence a R, o atacante tenta adivinhar X que pertence a S, onde H(X) = Y.

Quando o conjunto de pre-imagens S é pequeno, existe uma maior probabilidade de se encontrar uma pre-imagem X para o valor Y.

Funções hash candidatas a **One-Wayness** requerem que o conjunto S **largo**.

One-Wayness prova que `P != NP` !

Propriedade: Resistência a colisões (Dificil de encontrar hashes iguais para inputs diferentes)

Deve ser difícil `X != X' onde H(X') = H(X)`

Em teoria, a probabilidade de ser produzida uma hash sobre dois inputs diferentes, diante aleatoriedade uniforme é `1/2^n`, sendo *n* o tamanho do output (numero de bits).

Garantidamente é encontrada uma colisão após se verificar cerca de `2^n` pares.

Qualquer função resistente a ataques de colisões **é também** resistente a ataques de segunda pre-imagem, e qualquer função que seja resistente a ataques de segunda pre-imagem é também resistente a ataques de pre-imagem, e como tal são funções **One-Way**.

`CR => 2-PR => PR => One Way`

Se ninguém consegue provar que P != NP, então ninguém consegue provar que exista uma função One Way, que por sua vez é PR, 2-PR e Collision Resistant...

Como quebrar as funções de hash?

"Genéricamente", através de um ataque brute-force... Percorrer os 2^n outputs => Encontra pre-imagem.

Ataque de Aniversário (Birthday attack)

Para encontrar colisões na função, podemos recorrer ao Birthday attack. Este ataque consiste em calcular todos os valores como num ataque de brute-force, com a exceção que armazenamos como entradas de uma tabela a pre-imagem e imagem, e após calcularmos uma nova imagem, percorremos a tabela para ver se encontramos a colisão. Tem de nome birthday attack porque explora o *Birthday Paradox*.

Complexidade de `2^n/2` !

Segurança em relação a ataques de colisões

Quando procuramos satisfazer a propriedade de resistência a colisões, definimos que os outputs devem ser igual a 2x os parâmetros de segurança. Isto é, para:

128-bit security => Output 256-bit hash 256-bit security => Output 512-bit hash

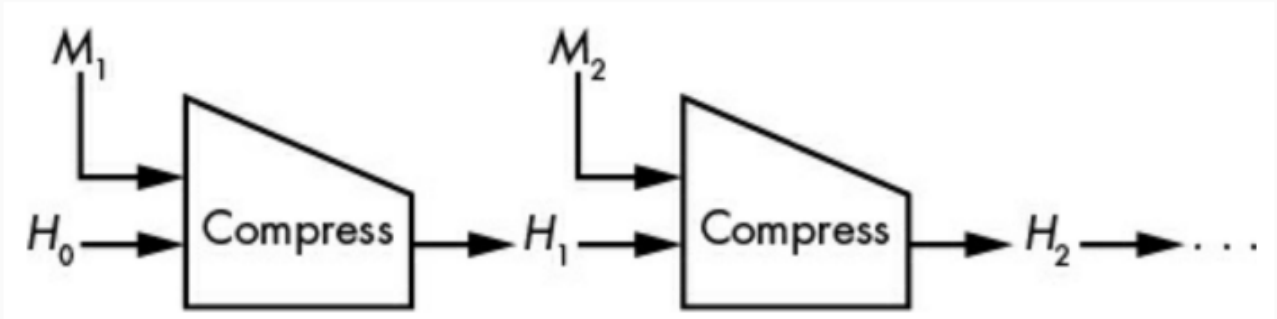
Construção de Funções Hash

Dois paradigmas que utilizam processos iterativos para construir funções hash:

- Construções **Merkle-Damgard**: Usadas para construir o MD5, MD5, SHA-1, SHA-256 e SHA-512. Utilizam como função de hash uma função que comprime *kn-to-n* bits, send *n* o tamanho do output e para um valor arbitrário de tamanho de input;
- Construições **Sponge**: Usadas para construir o SHA-3. Utilizam como função de hash um função que utiliza l-bit permutations, para qualquer tamanho de input e output.

Como é feita a construção de Merkle-Damgard?

Existe um valor inicial ou IV, *H0*, uma mensagem *M* que é partida em blocos de tamanho *kn* (M1, M2, ... Mn). Este valor inicial serve como input para a função de compressão, incluindo o M1. Depois, o output da compressão é o **H1**. Este valor H1 vai servir como input da função de compressão da mensagem M2, e assim sucessivamente.



O tamanho dos blocos depende do tamanho do output, sendo o dobro do mesmo, ou seja:

- SHA-256 tem como tamanho de bloco 512 bits e output 256;
- SHA-512 tem como tamanho de bloco 1024 bits e output 512.

O **padding** dos blocos é ***crucial***, sendo necessário incluir um bit com valor **1** à frente da mensagem e preencher com zeros até 64/128 bits (SHA-256/512) do fim do próximo bloco. Os últimos 64/128 bits codificam o tamanho da mensagem em bits.

Segurança da construção Merkle-Damgard:

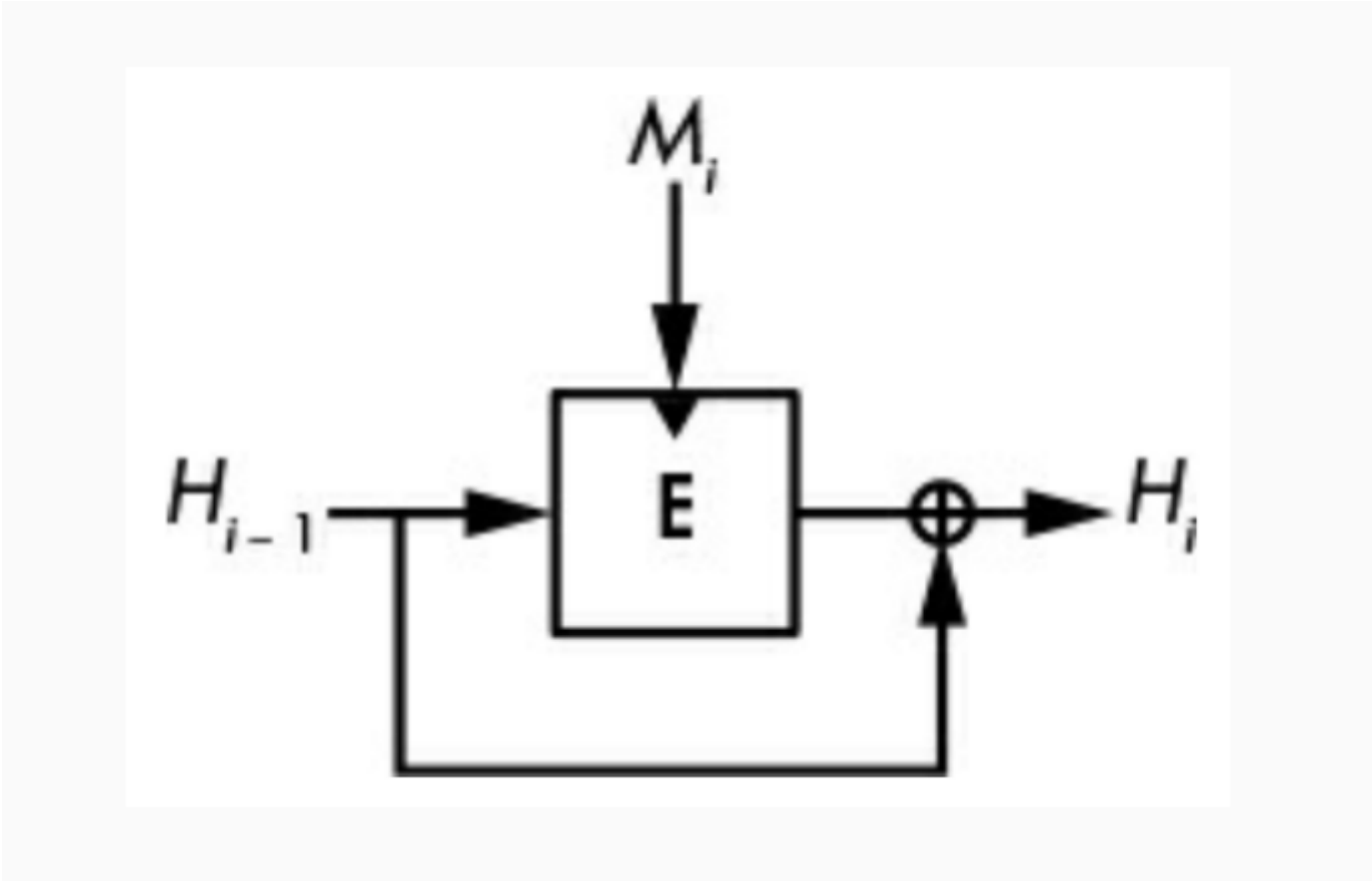
A função de compressão é resistente a colisões para **inputs pequenos** e implica que toda a construção é resistente a colisões para qualquer que seja o input.

Existe um problema nas construções de Merkle-Damgard, que incidem no uso do padding indevido. Dado uma mensagem *M*, seu hash *h* e chave *K*, é possível calcular
????

Como funcionam as funções de compressão usada no Merkle-Damgard?

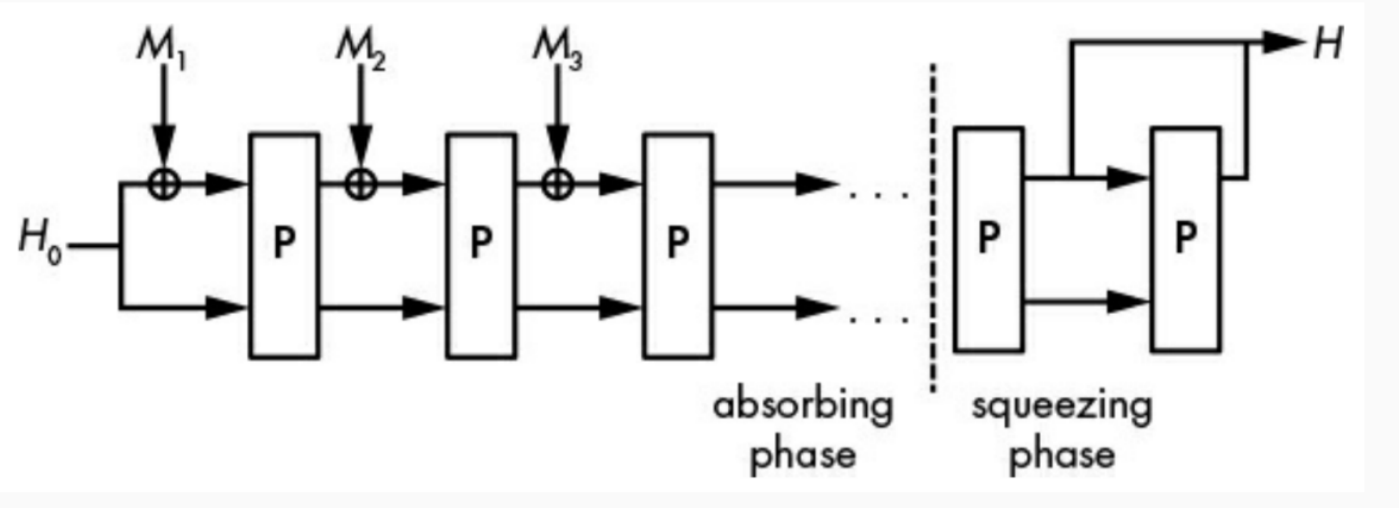
As funções mais usadas em Merkle-Damgard utilizam a construção de *Davis-Meyer*:

Dado o hash i-1 e mensagem i, é aplicado uma função de encriptação à mensagem e no final aplicado uma operação XOR com o hash.



Como é feita a construção Sponge ?

Paradigma recente e alternativo ao Merkle-Damgard. Recorre a uma permutação de tamanho fixo e que não usava chaves. É versátil no sentido em que o tamanho do input e output é arbitrário, utiliza PRGs, PRFs , stream ciphers e keyed hashes.



Na construção Sponge, existem duas fases: **absorb** e **squeeze**.

A fase absorb é semelhante ao Merkle-Damgard, onde a partir de uma hash inicial e mensagem M é realizado um XOR na mesma (H XOR M) e depois passa-se o output + o hash inicial para a função de permutação. O output da função de permutação é usado como hash na fase seguinte (mensagem e função).

A fase squeeze pega em todas as permutações criadas e a transforma-as num hash final.

O tamanho dos estados é do mesmo tamanho do input da permutação.

Funções de Hash Concretas

MD5

Quebrado (2005), tinha como output apenas **128-bits**. A função de hash mais popular até ser quebrada. Hoje em dia em poucos segundos encontra-se colisões. A família SHA utilizam um design muito parecido.

Secure Hash Algorithm (SHA) Family

Estandardizada pela NIST nos USA, mas vista como um de-facto standard no resto do mundo.

Inicialmente foi publicado o **SHA-0**, em 1993. Mais tarde, foi substituído pelo **SHA-1** em 1995. Ambos partilham propriedades similares:

- Outputs de **160-bits**;
- Ataque de Colisão na ordem das **2^60** - **2^80** operações;
- Mais tarde ataques de colisão na ordem das **2^33** operações.

Mais tarde introduzido o SHA-2, com outputs de 256 e 512 bits. Utiliza os mesmos princípios de design, mudando apenas o tamanho dos parâmetros que utiliza.

No futuro, utilizar-se-á o SHA-3, que utiliza como paradigma a construção Sponge e não Merkle-Damgard.

Como está construído o SHA-1?

Utiliza a construção Merkle-Damgard com a função de compressão Davis-Mayer. A cifra de bloco usada na função de compressão tem de nome **SHACAL**.

Blocos de mensagens com **512-bits**, outputs de hashes com **160-bits**.

Como está construido o SHA-2?

Familia de 4 funções hash: **SHA-224**, **SHA-256**, **SHA-384** e **SHA-512**, sendo que os dígitos identificam o tamanho do output.

Do SHA-1 melhorou-se as cifras de bloco utilizadas e o tamanho dos parâmetros.

SHA-224 e SHA-256 utilizam blocos de tamanho **512-bits**, sobre **64** rondas. Estes diferem no IV utilizado e o output truncado.

SHA-384 e SHA-512 relacionam-se na sua construção, sendo que a função de compressão do SHA-512 opera sobre **80** rondas.

Como está construido o SHA-3?

Após a fase de segurança heurística, o design da equipa **Keccak** foi selecionado em 2009, após uma competição de 3 anos. Esta competição realizou-se para apurar novos designs no caso do SHA-2 ser quebrado.

O Keccack é muito diferente do SHA-2 e também muito flexível. Utiliza a construção de esponja baseada em permutações de **1600 bits**. Os blocos de entrada nas funções de permutação podem ter **1152**, **1088**, **832** e **76** bits, que correspondem a outputs de **224**, **256**, **384** e **512** bits.

Keccak introduz as funções SHAKE, de 128 e 256 bits, XOFs (eXtendable Output Functions), que permitem especificar o tamanho do output.

Keyed Hashing

Keyed Hashing = Hashing utilizando chaves explícitas (que não são necessárias de especificar/introduzir)

O que são MACs?

Message Authentication Codes (MACs) são checksums construídos sobre uma sessão de comunicação, onde é aplicado a chave da sessão sobre as mensagens da comunicação, para garantir a integridade e autenticação das mensagens transmitidas.

Tipicamente são usados para garantir a integridade e autenticação nos protocolos de comunicação SSH, IPSec e TLS.

Flow MAC entre X e Y:

- X e Y concordam no uso de uma chave K;
- X calcula T = MAC(M, K) e envia (M, T);
- Y recebe (M, T) e recalcula T′ = MAC(M, K)
- Y procede com a comunicação se T = T′.

Nesta troca de informação, as mensagens são publicas pois o Mac não garante a confidêncialidade. Para tal recorre-se a esquemas de encriptação.

Segurança em MAC

Em MACs existe a noção de UF-CMA (Unforgeability - Chosen Message Attacks). A experiencia de segurança decorre da seguinte forma:

NÃO PERCEBI UF-CMA

MAC por si não protege compra ataques de replay, pois o atacante pode estar a escutar a rede e enviar (M, T) múltiplas vezes, verificando-se sempre o MAC. Para tal é necessário garantir que as mensagens nunca-se repetem na rede através da adição de nonces e timestamps.

Construção de MACs

MACs são construídos através de funções hash e cifras de blocos. A construção mais simples involve a utilização de uma **chave prefixo**.

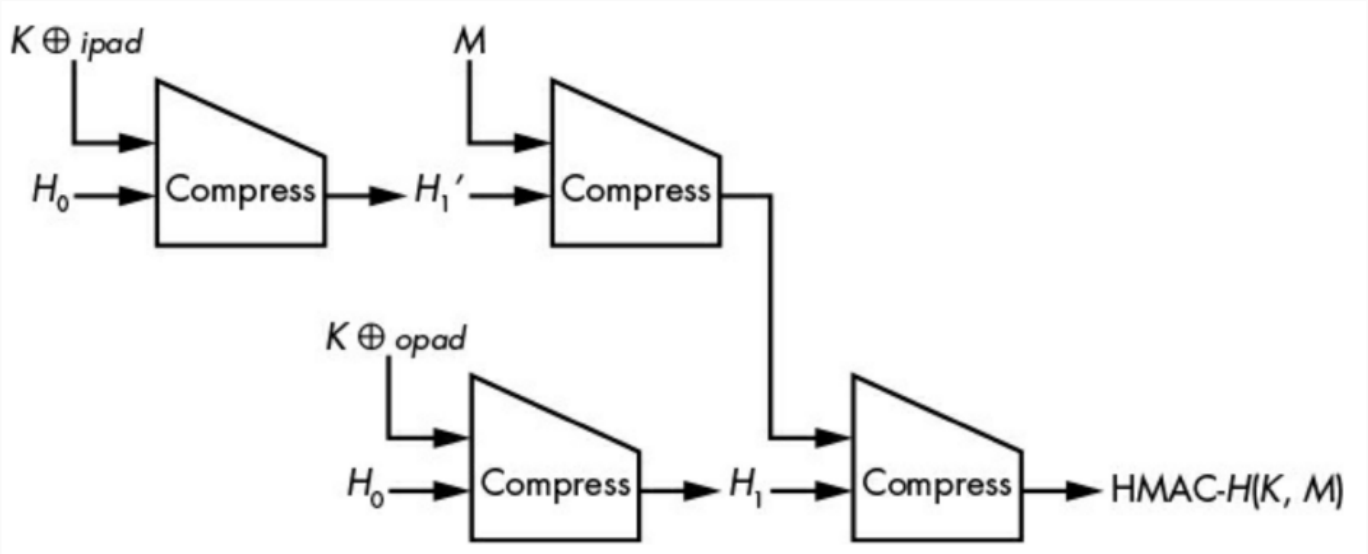
```
MAC(K, M) = H(K || M) ou PRF (K, M) = H(K || M)
```

Por causa do *extension length attack*, o MAC e PRF baseados na construção Merkle-Damgard são inseguros.

O que é o HMAC e como está construido?

HMAC (Keyed Hash Message Authentication Code) é uma construção de MAC que utiliza **key hashing**. Possivel de ser conjugado com a construção Merkle-Damgard, sendo que a função de compressão é um PRF, produzindo um MAC seguro.

No função que utiliza valor inicial (H0) é conjugado a chave com o XOR de duas constantes: ipad e opad. Calculado o hash de estas duas constantes em conjunto com o valor inicial, conjuga-se por final as duas e recebe-se o hash HMAC (M, K).



Colisões podem ocorrem o que levam a MAC forgeries. Como a chave não é conhecida no inicio, ataques de colisões são mais difíceis. Contudo, colisões acontecem com grande probabilidade a partir de **2^n/2** MACs calculados.

O que é o CMAC?

Cipher-based Message Authentication Code (CMAC) que opera de forma similar ao CBC, processando de forma sequencial e por blocos o valor do hash.

O que é o Poly1305?

Construção de MAC eficiente based em funções de hash **universais** e a construção **Wegman-Carter**.

Funções de hash universais são uma forma fraca de hashing, sendo que estas não tem que ser resistentes a colisões. São parametrizadas por uma chave *K* e garantem que para qualquer duas mensagens diferentes (M0 != M1)

`Pr[UH(K, M0) = UH(K, M1)] <= e`

Considerando que K é aleatório e `e` bastante pequeno.

Como não existem requisitos de segurança, a sua construção é fácil de especificar. No entanto estas **apenas conseguem autenticar uma mensagem**. (O QUE SIGNIFICA ISTO???)

O que é a construção Wegman-Carter?

A construção Wegman-Carter converte uma função universal de hash em um MAC completamente seguro, através de um PRF ou block cipher. Para tal é encriptado o output da função hash universal

Exemplo: UH(K1, M1) XOR PRF (K2, N), sendo *N* um valor público que nunca se deve repetir (contador ou gerado aleatoriamente). A chave completa do MAC é composta pelas duas chaves **K1**

e **K2**.