Week 6

Andre Rodrigues - up201505639
Pedro Antunes - up201507254

Consolidation questions

1 - Explain why a compressing hash function cannot be collision-resistant against computationally unbounded attackers.

Ser “collision resistant” significa que não é possível encontrar uma mesma hash, para mensagens distintas.

Uma função de compressão significa que recebe um input de tamanho n e o transforma num output de tamanho m, onde n > m.

Devido a este factor, funções de Hash que sejam funções de compressão, inevitavelmente vão ter colisões. Para explicar este princípio podemos olhar para o "pigeonhole principle” que diz: se houver m ninhos e n pombos para colocar nesses ninhos, e se n > m, então pelo menos um ninho tem de conter mais do que um pombo.

Isto significa que duas mensagens diferentes terão um mesmo valor de hash, quebrando assim a propriedade “Collision Resistance”.

2 - What is the best possible hash function? Explain why it is collision-resistant, second-preimage resistant and preimage-resistant. Why don’t we use the best possible hash function in practice?

Uma tabela que tem como característica originar valores de input com base em valores de output pode ser descrita como uma tabela com uma célula para cada valor do domínio e onde a cada célula é atribuído um valor do contradomínio. A função de hash deve criar uma tabela destas completamente aleatória, ou seja, o output gerado deve ser completamente aleatório. A função deve ser pública e acessível a todos e onde a sua computação é determinística, obtendo sempre o mesmo output para um mesmo input.

Como a função origina valores indistinguíveis de valores completamente aleatórios nós temos a maior entropia, a maior probabilidade de um atacante ter ½ probabilidade de acertar em cada bit, o que faz com que o atacante não consiga

inverter a função e através do seu output obter o input respectivo (preimage-resistant) e também com que ele tenha de tentar todas as possíveis mensagens até encontrar um valor de hash que coincida com o valor do output (second-preimage). A função de hash perfeita também é collision resistant porque mantemos todas os valores de hash gerados portanto:

Se o valor H(x) já foi calculado então retornamos o valor previamente computado e caso contrário geramos um novo valor de forma aleatória e verificamos se já existe ou não novamente, repetindo o ciclo até gerar um valor que não colide.

Não é usada na prática devido a ser estática, ou seja, necessitar de uma quantidade de memória exorbitante para manter as tabelas de todas as possíveis hashes, para além do custo de procura e construção das tabelas. Complexidade em gerir o acesso de todo o mundo a estes valores.

Uma função de hash perfeita é basicamente uma função que gera valores aleatórios e nenhuma colisão.

3 - Give one example that shows that the Merkle-Damgard construction does not give you the best possible hash function.

A construção de Merkle-Damgard é suscetível a um tipo de ataque chamado de “Length-Extension Attack”, onde se nós conhecemos um valor de Hash para uma qualquer mensagem M (Hash(M)), é possível computar Hash(M||M’) para um M’ a nossa escolha. Isto acontece porque o valor Hash(M) será o valor do estado posterior a calcularmos o valor de Hash(M’) retornando o valor de Hash(M||M’). Não deveria ser possível criar uma hash válida sem conhecermos o resto da mensagem, isto é, sem conhecer M.

4 - How does the David-Meyer construction relate to the Merkle-Damgard construction?

A construção de Merkle-Damgard utiliza uma função de compressão, isto é, dado um input grande ela comprime-o em um output mais pequeno, mais especificamente em 256 ou 512-bits. A construção de david-meyer refere se a uma forma de implementar essas tais funções de compressão, nomeadamente, através de uma cifra por blocos onde o lugar da chave dá origem a um bloco da mensagem e recebendo um valor IV (valor de estado). Portanto a construção de Merkle-Damgard utiliza a construção de Davies-Meyer para implementar a sua função de compressão.

5 - Why does the SHA-3 standard use a completely new approach to hash function design?

SHA-3 não segue a construção de Merkle-Damgard conhecida na família de hashes SHA-1 e SHA-2. SHA-3 utiliza uma construção chamada “Sponge” e pertence à família de primitivas “Keccak”.

A construção “Sponge” recorre ao uso de permutações ao invés de cifras por bloco (usadas na construção MD para implementar as funções de compressão). É dividida em duas fases: uma de absorção onde processamos cada bloco da mensagem em conjunto com um valor de estado e onde o valor de cada bloco da mensagem é menor do que o output da permutação aumentando assim a entropia do output. Depois de processada a mensagem, entramos na fase de aperto onde é gerado o valor de hash a partir de uma permutação em conjunto com o valor da permutação anterior.

Guided practical assignment

1 - Use OpenSSL to compute the SHA-256 hash value of the slides.pdf file published last week and check that it equals the below value. What does this tell you?

```
763e4d8de5915cecda6e4de8fa455bdbdd2f487a8df9420511cdb27098347
d74
```

```
openssl dgst -sha256 'slides.pdf'
SHA256(slides.pdf)=763e4d8de5915cecda6e4de8fa455bdbdd2f487a8d
f9420511cdb27098347d74

Isto diz-nos que o ficheiro slides.pdf não foi alterado, a
sua integridade mantém-se.
```

2 - Use Python to crack the security of predictable passwords in `crack_hash.py`

- The file has the twenty most common passwords of 2019.
- The code produces hash values of passwords (salted and non-salted), then they are shuffled.

- From the shuffled hashes and the list of most common passwords, retrieve the original passwords!
- Is it faster to attack salted or unsalted hashes?
- Include a succinct analysis of how long it takes to do these attacks.

Ver ficheiro crack_hash.py

3 - Use the tool available here (http://alf.nu/SHA1) (or any other tool that works) to construct two PDFs with the same SHA-1 value. One of the PDFs should explain how a single SHA-1 collision allows finding infinite pairs of colliding PDFs.