Week 3

Andre Rodrigues - up201505639
Pedro Antunes - up201507254

Consolidation questions

1 - Consider the set of numbers in the range 0..250 and note p= 251 is a prime number.

• Compute the probability of each value in this range occurring if we sample a byte b uniformly at random and then reduce b(mod p), i.e. compute the remainder of b divided by p.

Para input 0 : 0 % 251 → 0

Para input 1 : 1 % 251 → 1

…

Como p é sempre maior do que b: b mod p = b (resto) porque o quociente será sempre 0.

P(b) = quantidade de valores = b / número de possibilidades = 1 / 251

• Compute the probability of each value in this range occurring if we sample a 64-bit word w uniformly at random and then reduce w (mod p).

P(b) = (2^64 / 251) / 2^64

• Are these distributions uniform? If not, can you think of a way to quantify how distant they are from uniform?

No 1º caso sim porque todos têm a mesma probabilidade de saírem, ou seja, se tomarmos todas as probabilidades de b obtemos o valor 1.

No 2º caso não podemos quantificar a distância do valor da probabilidade para uma probabilidade uniforme, podemos usar a entropia.

2 - Repeat the exercise for p= 2^8, i.e., a power of 2.

P = 2^8 = 256

Input 251 -> 251 % 256

No 1º caso mantém-se tudo igual, apenas agora temos mais valores.
No 2º caso agora obtemos uma distribuição uniforme.

3 - What is a PRG? Suppose you are given a PRG where:
- for a seed s,
- the output is r=f(s)
- where f is a linear function.

Can you immediately conclude this is insecure? Why?

PRG significa gerador pseudo aleatório, e recebe uma um pequeno input aleatório (seed) de forma a poder gerar um output longo que seja indistinguível de um output completamente aleatório. E não pode ser completamente aleatória porque esta função é determinística onde o input é mapeado no output, como dito de forma determinística, em conjunto com outros valores aleatórios de forma a garantir que o output pareça aleatório. A PRG recebe estes valores aleatórios de fontes totalmente aleatórios (RNG) e portanto a sua aleatoriedade depende da entropia destes valores.

Sim por dois motivos. O primeiro tem haver com o facto de f ser linear, o que torna o output previsível porque teremos menos equações do que a quantidade de variáveis e que são de fácil resolução, enquanto que uma função não linear eleva o número de equações de forma expoente. E o segundo fator é o facto de utilizarmos uma mesma seed s o que faz que para um mesmo input irá produzir um output similar.

4 - Explain what is the difference between heuristic and provable security. Give one example for each type of security justification in cryptography.

Quando queremos estabelecer confiança em algoritmos de criptografia, podemos nos basear em provas matemáticas, no qual apelidamos de segurança provável ou

então através da evidência de tentativas falhadas em quebrar um algoritmo, que é conhecido por segurança heurística.

Segurança provável consiste em fazer uma redução de um algoritmo de criptografia a um problema matemático, onde qualquer método que consiga resolver o problema matemático também consegue resolver o algoritmo de criptografia. Um exemplo é de um esquema de criptografia que é reduzido a um problema matemático, é o RSA que é reduzido ao problema de factorização.
Ao invés de compararmos com problemas matemáticos podemos também comparar a outros problemas criptográficos.

Alguns esquemas criptográficos não podem ser reduzidos a nenhum outro problema, como o caso do AES. Neste caso é utilizada a segurança heurística, onde a única forma de confiarmos neste esquema é que muitas pessoas tentam fazê- lo quebrar mas sem sucesso.

Guided practical assignment

1

2

3 -hexdumpcan be used to extract randomness from/dev/urandom. Explain what the following command is doing.

```
$ hexdump -n 32 -e'1/4 "%0X" 1 "\n"'/dev/urandom
```

Implement an alternative command that uses /dev/urandom to create a file with random bytes.

- HINT: use the shell dd command.

Use openSSL to do exactly the same.

- HINT: look at command rand.

Obtém uma string de 32 bytes de /dev/urandom em hexadecimal, onde cada linha tem 4 bytes de hexadecimais.

$ dd if=/dev/urandom of=file.txt bs=32 count=1

$ openssl rand -hex 32

5 - Use openSSL to generate a key pair where the private key is protected with a password.

```
openssl genrsa -aes128 4096
```

Quando aumentamos o tamanho da chave o tempo de a gerar também aumenta, e vice-versa.

6 - Use openSSL to generate random Diffie-Hellman parameters.

```
openssl dhparam 2048
```

See what happens when you increase/decrease the key size. Compare to the previous case.

O tempo de gerar uma chave com metade dos bits da do RSA para o Diffie-Hellman tem um custo de tempo e processamento muito superior.