Tutorial #7

O presente relatório tem como objetivo descrever o processo seguido para a resolução da ficha `Tutorial #7`, disponibilizada no âmbito da disciplina de Criptografia Aplicada. As seções numeradas em baixo representam cada um dos exercícios resolvidos.

Diffie-Hellman com OpenSSL e SageMath

1) Produza parâmetros Diffie-Hellman com 128-bit (4096-bit modulus) security, sem recorrendo à opção *dsaparam* do OpenSSL

É possivel recorrer ao OpenSSL para produzir os parâmetros Diffie-Hellman através do seguinte comando:

`openssl dhparam <bit_modulus_size> -out <generated_parameters_filepath> (35 min)`

Onde `<bit_modulus_size>` corresponde ao número de bits do qual o operador modulus deve processar, e `generated_parameters_filepath` o caminho e nome do ficheiro a ser gerado com a informação dos parâmetros. Para o exercicio em concreto foi então utilizado o valor **4096** para o número de bits, como demonstrando no seguinte comando:

`openssl dhparam 4096 -out 4096-dh-parameters-no-dsaparam`

Esta operação demorou cerca de **35** min a terminar.

2) Produza parâmetros Diffie-Hellman com 128-bit (4096-bit modulus) security, recorrendo à opção *dsaparam* do OpenSSL

Da mesma forma que no exercício 1), é possível produzir os parâmetros Diffie-Hellman recorrendo à opção `-dsaparam`, de uma forma mais eficiente. Esta opção indica ao OpenSSL para usar o algoritmo *ECDSA* (Eliptic Curve Digital Signature Algorithm) em vez do Diffie-Hellman na produzação dos parâmetros, convertendo estes para o formato Diffie-Hellman no final.

O seguinte comando foi utilizado para produzir os parâmetros, demorando cerca de **10** segundos para concluir a operação:

`openssl dhparam 4096 -dsaparam -out 4096-dh-parameters-dsaparam`

3) Comente porque a primeira abordagem é mais demorosa, usando o Sage para confirmar a mesma

A primeira abordagem produz os parâmetros Diffie-Hellman da forma mais "legítma", sendo um processo demorado pois é necessário gerar valores primos de 4096 bits até que a condição modulus seja satisfeita. Como referido, na segunda abordagem é utilizado o algoritmo ECDSA para gerar os parâmetros, convertendo os mesmos no formato Diffie-Hellman antes de produzir o output final. Enquanto este último produz números primos com um tamanho suficientemente grande para produzir a chave, no caso do Diffie-Hellman os números primos são produzidos onde a condição `is_prime((p - 1) / 2)` se satisfaz, sendo **p** o número primo a ser produzido.

É possível intepretar o valor primo usado para os parâmetros produzidos através da ferramenta openSSL. Usando o seguinte comando, podemos guiar o openSSL a imprimir o valor primo em hexadecimal, como também pedir a sua avaliação relativamente à segurança do número primo:

```
openssl dhparam -in <generated_parameters_filepath> -check -text
```

Executando o comando para os parâmetros produzidos com a opção `-dsaparam`, obtemos o seguinte aviso:

```
p value is not a safe prime
```

Já para os parâmetros produzidos sem a opção, é nos informado que o número primo aparenta estar *ok*.

```
DH parameters appear to be ok.
```

Importando o valor hexadecimal no Sage e convertendo o mesmo num valor inteiro, podemos comprovar que valor com a opção `-dsaparam`, não é de facto um valor primo seguro.

```
hex_4096_dh_parameters_no_dsaparam = '00:c0:7e:12:09:54:73:99:14:32:5c:d9:82:44:4b:84:bf:41:92:bf:8c:bd:c4:8a:77:7
hex_4096_dh_parameters_with_dsaparam = '00:a5:80:69:6e:a5:8b:f3:cb:04:e4:26:fb:0a:88:93:49:d5:20:d2:f4:f2:ee:a4:a0

def hex_to_bigint(hex_string):
    hex_split = hex_string.split(":")[::-1]
    bigint = 0
    byte_value = 1
    bit8 = 256
    for i in hex_split:
        hex_int = int(i, 16)
        bigint += hex_int*byte_value
        byte_value *= bit8
    return bigint

def is_safe_prime(prime):
    return ((prime - 1) / 2).is_prime()

bigint_4096_dh_parameters_no_dsaparam = hex_to_bigint(hex_4096_dh_parameters_no_dsaparam)
bigint_4096_dh_parameters_with_dsaparam = hex_to_bigint(hex_4096_dh_parameters_with_dsaparam)

assert(is_safe_prime(bigint_4096_dh_parameters_no_dsaparam))
assert(is_safe_prime(bigint_4096_dh_parameters_with_dsaparam) == False)
```

4) Utilize o Sage para verificar que a utilização de Diffie-Hellman funciona, utilizando os parâmetros produzidos para

1. Produzir exponentes *x*, *y* sobre a distribuição `[0, q[`, onde *q* é a ordem do grupo produtor;
2. Calcular `X = g^x mod(p)` e `Y = g^y mod(p)`;
3. Verificar que `X^y mod(p) = Y^x mod(p)`.

Traduzindo estas expressões no Sage, obtemos os seguinte código:

```
from random import seed, randint
p1 = bigint_4096_dh_parameters_no_dsaparam
p2 = bigint_4096_dh_parameters_with_dsaparam
q = randint(0, bigint_4096_dh_parameters_no_dsaparam)
x = randint(0, q-1)
y = randint(0, q-1)
g = (GF(p1))(p2)
X = (g^x) % p
Y = (g^y) % p
Xy = (X^y) % p
Yx = (Y^x) % p
assert(Xy == Yx)
```

5) Seja p=1373, g=2, X=974, y=871, qual o valor Y e o valor secreto da Alice (x)?

Tendo com base as expressões préviamente definidas, podemos concluir que:

```
Y = g^y (mod(p))
= 2^871 (mod(1373))
= 805
```

Sabendo também que `X^y (mod(p)) = Y^x (mod(p))` então:

```
X^y (mod(p)) = Y^x (mod(p))
= 974^y (mod(1373)) = 805^871 (mod(1373))
= log_974(805^871 (mod(1373))) (mod(1373)) = y // 1)
= 1.0041186286870067 = y
```

Em 1) inverte-se a condição exponencial de modo a obter a versão logarítmica `2^k=c => log2(c)=k`.