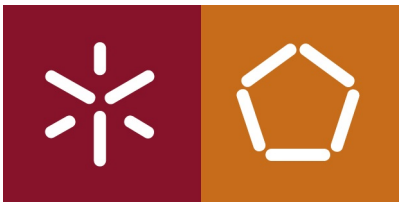


Universidade do Minho

Mestrado Integrado em Engenharia Informática

Engenharia de Segurança


# Aula2


Grupo 4

Autores

João Sousa (A77768)
Francisco Araújo (A79281)


19 de Fevereiro de 2019

# Conteúdo

# 1 Números aleatórios/pseudoaleatórios

## 1.1 Pergunta 1

As conclusões que podem ser tiradas depois de executados os quatros comandos apresentados na ficha prática, é que a geração de números pseudoaleatórios até 64 bytes, o sistema é extremamente rápido a processar esses bytes pois como o tamanho não é muito grande, o sistema possui entropia suficiente para responder de imediato, uma vez que com a execução do comando com especificação *random* tem em conta a entropia do sistema para gerar os tais números pseudoaleatórios.

Pelo contrário, o mesmo não se verificou quando tentamos gerar números pseudoaleatórios de 1024 bytes, pois neste caso o sistema não possui entropia suficiente para os gerar, demorando algum tempo a ser processado o respetivo comando, para o caso em que o comando contenha a especificação *random*.

Caso a especificação seja *urandom*, neste caso o sistema gera um número pseudoaleatório sem ter em conta a entropia do sistema, sendo o resultado imediato.

## 1.2 Pergunta 2

Neste caso após a instalação do software *haveged*, constatamos que para gerar um número pseudoaleatório de tamanho 1024 bytes foi imediato tanto para o comando com a especificação *random* como para o comando com a especificação *urandom*, o mesmo se sucedeu para os de tamanho inferior, que já acontecia anteriormente.

Isto acontece, pois o *haveged* é um software bastante usado que tem como objetivo explorar o volatilidade do estado do hardware como fonte de incerteza, o que leva a que a entropia no sistema seja bastante mais elevada do que acontecia quando este software não se encontrava ativo.

## 1.3 Pergunta 3

### 1.3.1 Pergunta 3.1

O output contêm apenas números e letras, uma vez que a função usada no respetivo código, denominada *generateSecret* importada do package *eVotUM/Cripto/shamirsecret* recebe unicamente o tamanho do número pretendido, e nessa função é apenas utilizado os caracteres que se encontram no *string.asci_letters* e no *string.digits*, sendo assim, o uso de caracteres especiais inexistentes.

### 1.3.2 Pergunta 3.2

Para usar caracteres especiais, a melhor forma seria modificar a função *generateSecret*, mais concretamente na linha 256 do ficheiro *eVotUM/Cripto/shamirsecret.py* de modo a permitir a utilização de todo o tipo de caracteres, podendo também colocar os caracteres que o utilizador entender.

```python
def generateSecret(secretLength):
    """
    This function generates a random string with secretLength characters (ascii_letters and digits).
    Args:
        secretLength (int): number of characters of the string
    Returns:
        Random string with secretLength characters (ascii_letters and digits)
    """
    l = 0
    secret = ""
    while (l < secretLength):
        s = utils.generateRandomData(secretLength - l)
        for c in s:
            if (c in (string.ascii_letters + string.digits) and l < secretLength): # printable character
                l += 1
                secret += c
    return secret
```

Figura 1: *Função que deve ser alterada*

# 2 Partilha/Divisão de segredo

## 2.1 Pergunta 2.1-A

Para dividir o segredo presente no enunciado recorremos ao comando ***python createSharedSecret-app.py number_of_shares quorum uid private-key.pem*** onde o *number_of_shares* foi oito e o *quorum* foi 5 e o *uid* foi g4 identificando assim o grupo. Antes disso foi necessário criar uma chave privada recorrendo ao seguinte comando ***openssl genrsa -aes128 -out mykey.pem 1024***.

Depois de executado tudo isto, obtivemos os 8 componentes correspondentes a cada parte.

## 2.2 Pergunta 2.1-B

O ficheiro *recoverSecretFromComponents-app.py* tem como objetivo recuperar apenas os componentes necessários para a recuperação do segredo, isto é, o número de *quorum*.

Pelo contrário o ficheiro *recoverSecretFromAllComponents-app.py* tem como objetivo recuperar todas as componentes em quais o segredo foi dividido. Este último ficheiro pode ser utilizado por exemplo, quando existe a necessidade de alterar o segredo, sendo preciso estarem os oito componentes para haver a recuperação e a respetiva distribuição das novas componentes.

# 3 Authenticated Encryption

A empresa deveria implementar o *Encrypt-then-MAC (EtM)*, isto é primeiro é realizada a cifra do texto e depois é realizado o *MAC* sobre o texto cifrado. Depois disso o texto cifrado e o MAC são enviados em conjunto. Este tipo de *Authenticated Encryption* é melhor do que o *Encrypt-and-MAC (E&M)* e *MAC-then-Encrypt (MtE)*.

Para ajudar a entender o algoritmo que deveria ser implementado foi realizado pseudo-código:

```
cifrar(plaintext):
        textocifrado = cifra(plaintext)
        mensagem_com_data = cyphertext+data
        hmac = (key, mensagem_com_data)
        return mensagem_com_data+hmac

decifrar(textocifrado):
        Hmac = retirarHmac(textocifrado)
        h=verificarMac(Hmac,key)
        if (h==True):
                data=getData(textocifrado)
                textocifrado_sem_data = retirarData(textocifrado)
                if (anuidade esta paga no dia da data):
                        return decifrar(textocifrado,key)
                    else:
                         return Null
           else:
                 return Null
```

# 4 Algoritmos e tamanhos de chaves

Nesta secção era-nos pedido para dado um país, no caso do nosso grupo, a Hungria, identificar o tamanho das chaves e os algoritmos presentes nos certificados *QCert for ESig*, emitidos pelas Entidades de Certificação (EC) que emitem certificados digitais qualificados.

Posto isto, optamos por escolher as seguintes Entidades de certificação:

- **Microsec Micro Software Engineering  Consulting Private Company Limited by Shares**

  Como esta EC já tinha emitido vários certificados, optamos pelo certificado emitido mais recentemente como nos era pedido, denominado *Qualified KET e-Szigno CA 2018*, que data do dia 31-12-2018. De seguida, realizamos os passos estipulados no enunciado, e obtivemos o seguinte:

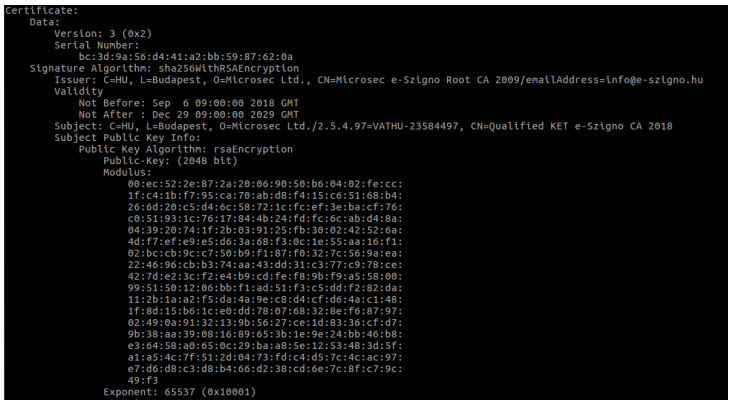


Figura 2: *Informação relativa ao respetivo certificado*

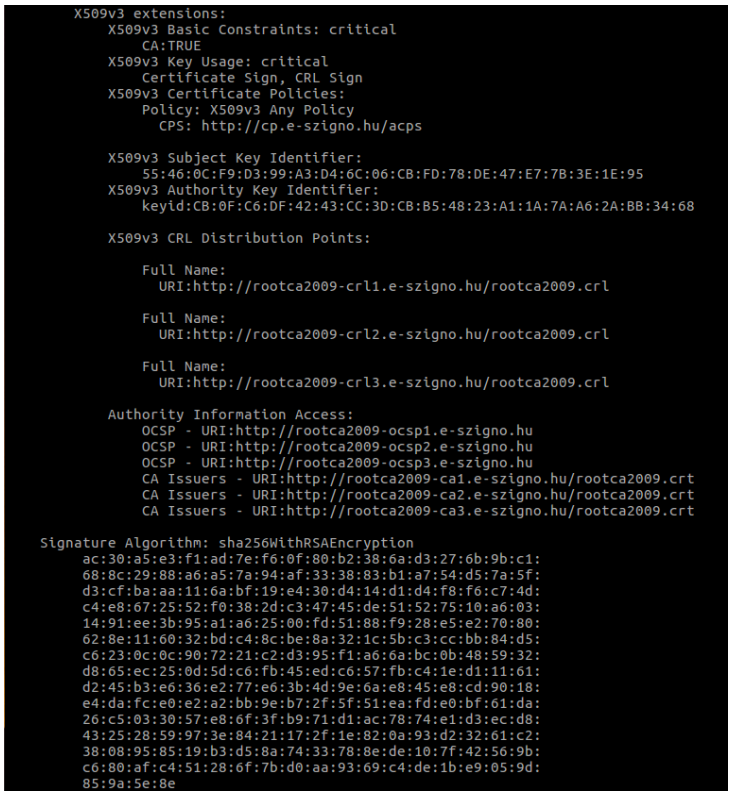


Figura 3: *Informação relativa ao respetivo certificado*

Dado o que encontramos, podemos concluir que o algoritmo utilizado é o *RSAEncryption*, e o tamanho da chave é de 2048 bit. Existem também outras informações relevantes como o algoritmo de assinatura utillizado.

- **NISZ National Infocommunications Services Company Limited by Shares**

Para esta EC tal como na primeira, escolhemos o emitido mais recentemente, denominado *Minősített Közigazgatási Tanúsítványkiadó - GOV CA*. Refizemos os mesmos passos que tínhamos feito para o anterior certificado e descobrimos que o algoritmo de assinatura e o tamanho da chave são iguais ao anterior.

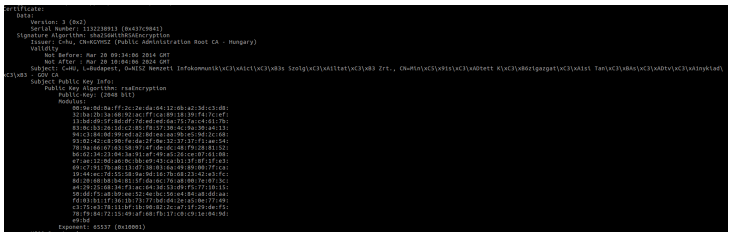

Figura 4: *Informação relativa ao respetivo certificado*

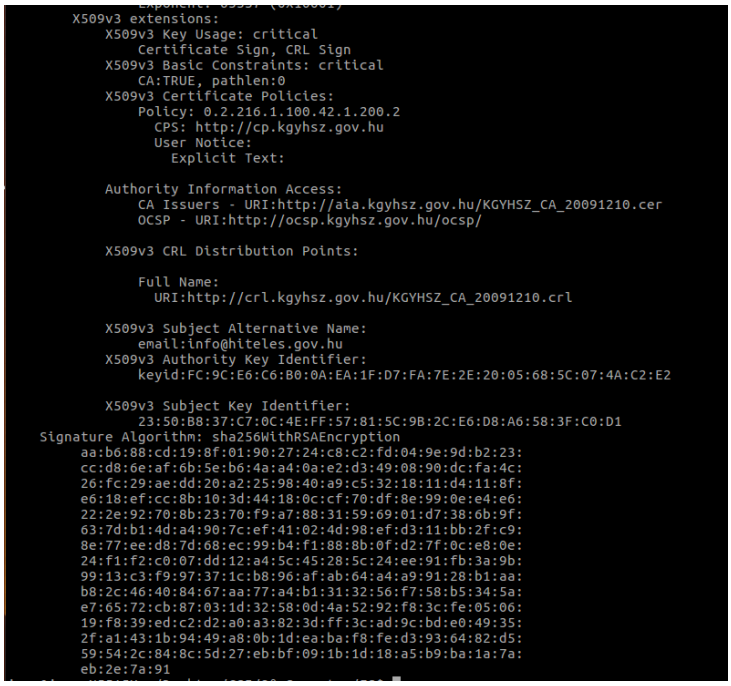


Figura 5: *Informação relativa ao respetivo certificado*

Posto isto podemos concluir que os algoritmos utilizado são os mais adequados, tanto a curto como a médio prazo uma vez que tanto a encriptação via *RSA* complementado com o *SHA-256* são dos algoritmos criptográficos mais seguros atualmente.