

UNIVERSIDADE DO MINHO
CRİPTOGRAFIA E SEGURANÇA DA INFORMAÇÃO
DEPARTAMENTO DE INFORMÁTICA

ENGENHARIA DE SEGURANÇA

TP1 - Aula 2

Grupo 7

Carlos Pinto Pedrosa A77320

José Francisco Gonçalves Petejo e Igreja Matos A77688

17 de Fevereiro de 2019

Conteúdo

1	Números Aleatórios/Pseudoaleatórios	2
1.1	/dev/random vs /dev/urandom	2
1.2	haveged - Daemon de Entropia	3
1.3	Geração de Segredos	4
2	Partilha/Divisão do segredo	5
2.1	ShamirSecret	5
2.1.1	Partilhar Segredo	5
2.1.2	Recuperar Segredo	6
3	Authenticated Encryption	7
3.1	7
4	Algoritmos e Tamanhos de Chaves	8
4.1	Holanda	8

1 Números Aleatórios/Pseudoaleatórios

1.1 /dev/random vs /dev/urandom

```
user@CSI:~$ time head -c 1024 /dev/random | openssl enc -base64
```

Figura 1: Processo /dev/random Bloqueado

```
user@CSI:~$ time head -c 1024 /dev/random | openssl enc -base64
LytX7qIe+G6Y1+h0yS9m3lkFXG0hQI7XVt1s3ggeM7zvJ559QWpmc3BgC+J4h3
KYccE01WmVujpmaA/B2p86yk30TUwEsayHxkj5Vw6D8aMrmSpZcxMLVFMq9Ekn5J
owxbt8VL89nkmB4dhZy8h1vM/ew3aMaBwPd1GEIik8B+Lkf6Y+nmF2paRwdgaFNG
2yvZM+D1/PfKsmyXY8BBVCwPF7F2925E5Tm7HNCTQ3Nl0eKR4ZgbIEjD1NoL3pS
Fm+ZyFq85x28Jf4Q0HqVgD3XaBYh5IgyXewrTrrrbcP/pKoEs+7z0hgj5ejQ1dE
9gyoUcAvC/qgBhb41/nqFrZmCNIz0g5dq/XHMW9K1T0d3JPa0tZas96x/udLU
XiNAExgW59Ng7sLE1J09bwVBFxqMj6JUb+1YjqZnLLM0mgoHd/tqGAGoPK5Po6t
j2cZaYpJdMvhYFyk2L6Uy43YptgZ80sbej+UjEfJW4N6JYXpWBC3fodg/yMtZv
JpO/vU10ymJ/bTBQIp1bvP5hv+SHB6ProUDcmTbYUqf0yZZWwcuSRBDfPmMs1IkK
YS+dLcz2GpkMGR11uT8lqvUfZbk6Q0B0Vypghcwm8djdzUK9F3+5fCLPma2CD8e
1Rpw78u849sVPY/Ro79Ns6pTj30FInJeY0jH09qQwyWRq3LxEZcx22xolJ+2xEu2
xs4tGhacVhX0laY50/ID1DAF080m6wKnNHXjU7BACd+dNE8Ax1vLPenqJ08G9s
VF7Z+tgIWWVsXAVV6lbeVaw1jeLH14dhr50qL+PbJ0C3+IodTbmF6mD5ApNl8s3G
CK5M7nd+Rqt1r0b0ekXtDny6TGCR0PFKWPk+8Fe0TEiayIS5063qVA+u4+9jV
3vuvCDR64C7m4kGyCSaMNZ1mrXZxzmKj7BPsLtrvNiUP+rC6f9TafrUqSRKLz9
MtoabKlKuCY3EVvFpdcjW3r1dhjBYvf0NiDnkj+H0LNUdR6V6uNHf87Zn6cpQH
CiI82WdPZIZJQU22/aePLPmp00XN9ZVB9K8stA8ZiDTONlEcVItI0UXCjxzHLQd2
Nidnq/9n6euBvioEwbgZhmwiTmfgrsNiHwoksBD5BfL9kziWx100jmkP0KJbz6
KC9SYXx1LF/0uX84WUihbLMY5+amZwTIIUqgkCAAQ4sJRsXKax8vui782sCnd9
00wsKfLK7pFC2kF3TLHmh09Rfnqg2T5sESWfMsBWfPrA56IZuwlMl0AQxKigCnLR
TliscFwLkzCjauSRlYpFvocMWPYTsD5Tx7CyD0SdyhhXUDLCdsc9lDmk6ZeQlB4
zWLMONxw5N3eYU6ze7LQ==

real    23m24.717s
user    0m0.008s
sys     0m0.000s
user@CSI:~$
```

Figura 2: Resultado e Tempo de Execução do /dev/random

```
user@CSI:~$ time head -c 1024 /dev/urandom | openssl enc -base64
iYAQ81lWtr6vPC5+ak0yVwM2Q1TdUDK67NC6TFj9WIJZGGhC7gZGpAE8J0r68F2N
WHzXASzZaRfDh2B6XAH5K+fzJfT/RmW7JIndIMBYhMdn4oQx55skwYk1d8k1WON
qLXI/DTXbUrsC2Mw7gESuUgTK61cLcYsN0rV9nuBPLghma5480axmT6adcmA
e/z16yHRY60/8i1wFy50wv92Cqfr0fLQapFnkx3g3+U7M1a8T/9q40jpe4DTY+
Cw0Vh0a+Fw5qz88U0TMQ/salAvx11tgo/1xGR/vjA19DMQsr2N7btDlFkdBpaHNK
b6VH2ZLUV/E3MYGK0DvWxxk6J6fbwX6a/RPR3C5PJMKnKn9wqThkVmu3Kjkom0X
Bk1Wx3NCA7zlg4woXyk4rL9h2wurLAa20B/0ps9Cc7/Hi7/Jw3fEznH4ds5Q39sz
0Afr/W/pN2IZU06b8XboiP2naJyJN2Wcd4es80Fa7Z/F/1ZVwEiwE8HPmXiHI+W1
ZujT1ldLXZkTvBzqEFP2pr5yKe17wLrQ1lUD2ankIY5ZbkVGRy6B7XHY75gTe1QU
Yc6twDKR5/+3PpzGc0lJYn/qabMGAzDqGZnQIYhe3WC0fXzkLv9KvW4NNwogZFm
7Du2ymwWmoJ9E08/NqATeAaKn9XwL1xtZak3eQHV0bzfk0EHK7oz1XXxMu+0e
Hv/6EZARwyjyDugIz6hGf0B9zVnTThoCFJq00RgtxyRz1FyDvNmrvh9fe4RmFX
q43GVR+o6n1rc1YwIYxAY2LYw0JXyIEfpU9NfdxeSraZdyVJerdTKbxxmd0edB9
NoFTGI/OfOpXWpqy/QCHPeI+8ViyQUthHZHosJ8BXbpWSTl3WTDq42pXTCRSW5f
YsWL48b1JiQvnBUTZCmnTjwmgW09F/nAivLH0nx7vpV+3e7odjpc591Rqazy0EU
0bkFWYsvC/fvhtqUBvR85GNl808f/JBTzKWFfMR/xy3ez8u+YtzW6wzKnqzYTbSn
ZciyZNUa2SLUxVyNHK8wfJH/100L2DI/7ciCbXrk+4BlinEqB7Uu0Y5kh9Y3s7PS
e/70hyRRzSPN1rkH+5m6C1rq/Yu5MYZ52xwFL0xyZs8WFtr1+yUenpgHBW/lgsQ1
XK2zYbuUpyNekETwkqJA/NNOPwK7dnuFpprRmKf9dvC6PxC3BxWsvjzmu5SUD
x5n8Tfak9W2wcgK7k0/da6G0GNvghuSG0LkosKRMLr0uJiVTkgt68dN5fega8L
50HGvBEAt5kHr5ST/T08TXFDuIXbNeHEYPmU/v1YvevX3UBkD+8RVDWw10BgJ06
cUL3/TZ5aIy709bojEP3LA==

real    0m0.004s
user    0m0.000s
sys     0m0.000s
user@CSI:~$
```

Figura 3: Resultado e Tempo de Execução do /dev/urandom

Tanto o `/dev/random` como o `/dev/urandom` são geradores de números pseudoaleatórios bastante usados no ambiente *Unix-Like*. A principal diferença entre os dois, como se pode verificar pela imagens acima, é que o primeiro bloqueia até que exista entropia suficiente para a geração do número, enquanto que no último este processo é imediato.

1.2 haveged - Daemon de Entropia

```
user@CS1:~$ systemctl start haveged.service
user@CS1:~$ time head -c 1024 /dev/random | openssl enc -base64
b4sBVo4AbR0rUHBlt0fFBvHD+7rrHKN1cv591A48Id7RD+bmiWA/081g0zj5kcc
3rmYj4m907b0UcXvKsE9cogsl5Ht1QzDvR2+K+ij14B036r0u0ejvzKXESVj8
BQ/e8q4yjdE05fXEccwRW3t+muP8XZ8uMmYl+h7+Yy4F6PT68g9p3C03Wl0PX
1x2Lh2Pt5Q/8z0zosEB+xlk44FNi4ggCUqJv6YOFYPxE68EhCM6JYv+GdP4Nu2B
2DH1zh7k3e4j1903M0HndLv2P/LA040QELtHCGDRB17cmI4W0n0Wz0kxUjHU
+Kk8zQD1zgmrfvKLBUBUcuHnSLPv/jcVUFj8jv5FgMD561d50G6v+st3m0qB
qf1aB5v2VbBGjvY/+yMnvVbv5JX+kfr1XAJyKM0fXQj/nwtVvk45wLye3FnXhLnp
0NjK8l/yj1JmdH709u4iaXFRYVn0C8INTCU0Thicq5AMUvUbuPFxVV2TRAaHbme
Jf80cuuaZ52bEtqbhyH6PgyF7Vfocy0som+ZjQK96K7glv0gRSPGAY/MvqfLXI
3PxTmuF7j4qJkyfVWm6oy+yH9S6b0S0K1ZrTcmh4lClQv5uLD63c44XmurrZC
7ZTKUVx044mc546eWdrWUrsH05+a7He1YTX4u1zuTe1Arj2c1LzrYPN0nd0HOR
ehGqAxcRRRLDLwsj5Va5JZAY0Gagj1j/Np0Ic1p0YK0YcZRLmc+dTK5L24p//My
11PqBdWRBE+dbLooCE9Nk2WJZk5FYVbL0Xh9RfLl+Cd12Gh/13vAW7tMTfcF1dP
Rkkt1BDAmu0tVjFOYAJ3Lmk1F6ZiHmpov50vzr+4V7+sm1K1Dw9VgP0u5952uJb
CPZgt099dtEa/mw1sq18ob0NKRQJ0RCySXDhtu1Kv1iFRBG0YejahSv56rgD5SX
R10BXa80wLfvHG6/CTLdis4dPFTUqa3g20Fro10fj8NEte/muQFMZjraA00ysC0qt
bsntFTG1BmqQ1YhK8gUjgpk8oFaap0Bd3F+NEthjT/aln2AKZLRc46MtC86nwBH
mgVjB/uu2w1uCOLZ5awJfae2X0ChkPte/v/T0apTxc4PZEUQc6Lp14BYMgtEq
Qx1r2E5x0jzr0myJ5aPqxV3L0Exgh2YwccUSF+PE0urs+9xIooPCCIOVLjJ0698
xb38FEUSs0F4U1a3njQ3trU0gASyWR6ysHHPc9ganoKKcMCsrx+0t1mfqnRt+Nc
y5fEGUN0827TdxzC9J0HX91Xj1TgqjDbc8X5BVd0L9qCnV/0uRCR3XgomkjpRuu
hgTlr1R1Y9s5P771Gw1HCKg==

real    0m0.005s
user    0m0.000s
sys     0m0.000s
user@CS1:~$ time head -c 1024 /dev/urandom | openssl enc -base64
g91m6FISCUgpe/2U14g8qncRyfeZgm129L7PKmjVD/78qW0KONVVk578R090tqb
X03Hg7LeAc+uNjYJ1T5HLMCMcD621eS3v1rbN99vkuMGewWzbQmQcfiyrECenU
Jti35V97RhfEunqleTo7o6JLKut4yqVMN8Ae371AHFzhfZ8TtwjTaE7bD1jCXX
1qS0b07+H3oXp5qHx4xm/Iq9R0UJgdtpw0ALuRB83Oq5PxfHMQ05B159u5S
hsauSvmj2ITBMW1JchEbZWLBQ0r5u3kfWDBNTg/u05d+WFPrunyPnrfxy2Z2tbs
bY0V3vf/Q/Egl0o6tuk6d/L01Lr9vUunn9v+IyPolaw3EIV59u4grJ7y6XDE6e3
yUmJfUgM/ftVCwJ0zyvyN/ZncuF/98rs19vfxGh3SLxwnR8secKZJVT13jUwq3
9pY5o0CFM0sc+MuyY5G/v6g1W0S1z3BELU1vnoMH3h3tK8B0v6gphh0m/0uG9
gPhtcVpMf1bdCU4LAT6ZR+nRW11X3oCG0706leNnxRRBwMmTaktICMTZ3fZH
/j38LKh2rJC2n5ngKdFPqN0otVegkf7/TLaP/mTUOFux5/yP3akZ5+Jfabwz8IY
rg9960mZtfjTep2hAyCb7CYHj6s1J05Vcdxajath+YMI+SoEwt4D0pRWAD/AJr7
D9B7T6RwEnR161bV3Rks2xoD0R0FryPjF9aLEdm0leCnwhTEAP/33tnf4z
XM3qRj6n65kV289LEUJed0G051ef6T2gor9rz/057ekrCLV5CscWAD4kfp6Tpd5s
Ks5PN3Dzd6MDXoGQRf8k0080e3HW1B3KlyeS0b0Sp4o0EC5PGAF5eLSRVmN1785
FrXZ0ox6581/DZV9orzR4VMPexEWEm/MmMURbcqQ5b+XhKnix/5AHUbtLhhjvU
xKQ20mVA2ED0Buk5b0LAUKK6qG0uqgh1AEye8j3PP6scXkM9r128G2AT05fn
DwKNpJK10g5yvg9S0u7ZB09Kc2HwSMuepX6mvaXfJfa0tq1r3Qcca+X6Jv1c1fx
90j1mj5rbF0szSLWnx/k1WBMLnoJ0f6x8/my3VhQEUfMYoyQAPV9WV050aM0FXG
idwh3b0mxtFpA9vz51Lotba0m9rK8psAbNy6BMPTEf8Xh6T3tZ0q6QvWkruiudG
GSP16jpr+P1a2L6j5VxKstfzAZAFNfV2hLwEaHBTXApf59h1TNU0N0Z7Tc2b
hCd97RSHFLq1VFPbzU1VG1CD6uVMgx92j1LXUN80L9A4QHPWzwaTcqeUv0h1f5L5
du7x1y+e6eKZE1zKy7dH0==

real    0m0.004s
user    0m0.000s
sys     0m0.000s
```

Figura 4: Tempos de Execução de `/dev/random` e `/dev/urandom` com o daemon de entropia ativo

Com a instalação e ativação de um *daemon* de entropia, a execução destes dois comandos continua a ter a diferença em cima mencionada embora não se note na sua execução. Acontece que o processo `/dev/random`, que bloqueia quando não existe entropia suficiente, nunca irá bloquear pois o *haveged* gera a entropia necessária para o processo completar sem qualquer *delay*.

1.3 Geração de Segredos

Depois de analisar o código do ficheiro *generateSecret-app.py* verifica-se que a geração do número pseudoaleatório é realizada através do método *generateSecret* da biblioteca *shamirsecret*.

Assim, o próximo passo foi analisar o método *generateSecret* da biblioteca cuja definição é a seguinte:

```
def generateSecret(secretLength):  
    ...  
    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): # Alterar esta secção  
                l += 1  
                secret += c  
    return secret
```

Podemos verificar pelo código acima que existe uma validação de cada carácter gerado (*c in (string.ascii_letters + string.digits)*) antes deste ser adicionado ao resultado e caso este não seja visível não é adicionado à string final.

Assim, para permitir que o output não sejam só letras e dígitos apenas é necessário remover a validação anterior, dando como resultado o seguinte código:

```
def generateSecret(secretLength):
    ...
    l = 0
    secret = ""
    while (l < secretLength):
        s = utils.generateRandomData(secretLength - 1)
        for c in s:
            if (l < secretLength):
                l += 1
                secret += c
    return secret
```

Figura 5: Resultados com o Código Acima

2.1 ShamirSecret

O primeiro passo para poder dividir um segredo através do ficheiro *createSharedSecret-app.py* é gerar uma chave privada e o respetivo certificado (para depois poder realizar a operação inversa) através dos comandos:

O segundo e último passo é executar o programa, como pode ser visto pela figura abaixo.

[illegible]

Figura 7: Exemplo de `recoverSecretFromComponents-app.py`

A utilização do programa *recoverSecretFromAllComponents-app.py* é muito semelhante à de *recoverSecretFromComponents-app.py*, mudando-se apenas o número de partes a utilizar.

2.1.2 Recuperar Segredo

A diferença entre os 2 ficheiros é que o *recoverSecretFromComponents-app.py* recupera o segredo com apenas um subconjunto das partes, enquanto o *recoverSecretFromAllComponents-app.py* necessita de todas as partes para

recuperar o segredo. É também importante notar que ambos os programas verificam as assinaturas dos componentes através do certificado passado com argumento.

De facto, o segundo pode ser bastante útil quando é necessário validar todos os fragmentos do segredo. Podemos tomar como exemplo uma tomada de decisão. Se for necessário consenso absoluto, isto é, todas as partes terem de concordar utiliza-se o *recoverSecretFromAllComponents-app.py*, mas se só for necessário maioria (+ que 50%) utiliza-se o que permite recuperar o segredo apenas com um subconjunto dos componentes.

3 Authenticated Encryption

3.1

Com base na biblioteca criptográfica *Cryptography* de *Python* pediria-se à equipa de desenvolvimento que utiliza-se o algoritmo *AES* em *Counter-Mode* com *Authenticated Encryption*.

```
def cifra (segredo_plaintext):
    aesgcm = AESGCM(key)
    nonce = os.urandom(12)
    ct = aesgcm.encrypt(nonce, data, nonce)
    // nonce, data, authenticated but unencrypted data
    // Garantir que o nonce não é alterado

    hmac = hmac(k, ct)

    return (ct, hmac, nonce)

def decifra (segredo_cyphertext, chave_cifra, hmac, nonce):
    hash = hmac(k, segredo_cyphertext)
    if ( hash != hmac ):
        return "Authentication Error"
    else:
        aesgcm = AESGCM(key)
        segredo_plaintext = aesgcm.decrypt(nonce, ct, nonce)
        return segredo_plaintext
```


4 Algoritmos e Tamanhos de Chaves

4.1 Holanda

Depois de devidamente analisado o conteúdo dos certificados das ECs *Ministerie van Defensie* e *QuoVadis Trustlink B.V.*, que podem ser encontrados na diretoria atual deste documento, pode-se verificar que ambos usam o *RSA*, como Algoritmo de Chave Pública com tamanhos de chave de 4096bits e o *SHA-256* como algoritmo de *Hash*. De facto, hoje em dia, tanto os algoritmos como os tamanhos de chave são apropriados, no entanto e num futuro próximo, com o avanço da computação quântica, podem vir a ser obsoletos muito rapidamente.