´y

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Engenharia de Segurança

Trabalho TP1

Grupo 2

-

Paulo Gameiro - A72067

Pedro Rodrigues - PG41092

Rafaela Soares - A79034

-

Braga, Portugal

24 de Fevereiro de 2020

Conteúdo

1 Exercício 1: Números aleatórios/pseudoaleatórios

1.1 Pergunta P1.1

Os geradores de números aleatórios podem ser obtidos através de fontes que tenham como base fenómenos físicos estocásticos e imprevisíveis, como o ruído atmosférico, ruído térmico e outros fenómenos eletromagnéticos e quânticos ou através de algoritmos computacionais, onde a fonte de entropia está na semente do algoritmo.

Os sistemas UNIX têm duas fontes de aleatoriedade: o /dev/random/ e o /dev/urandom/. A implementação de ambos usa um gerador que mantém uma estimativa dos *bits* do ruído da *pool* de entropia. A partir da *pool* de entropia são gerados números aleatórios.

No /dev/random, quando lido só irá retornar *bytes* aleatórios dentro do número estimado de *bits* de ruído na *pool* de entropia. Quando a *pool* de entropia está vazia, as leituras /dev/random irão bloquear até ser obtido mais ruído ambiente.

Por outro lado, o /dev/urandom usa uma *pool* interna para produzir mais *bits* pseudo-aleatórias. Ou seja, a leitura não bloqueará, mas o *output* poderá conter menos entropia.

Existem diferentes implementações de algoritmos para gerar sequências pseudo-aleatórias:

- Linux 4.8 e mais recentes implementações baseadas em ChaCha20
- Outros sistemas Linux: Usam o algoritmo de Yarrow, no entanto este foi substituído pelo Fortuna.

De referir que os algoritmos usam a *seed* obtida no arranque do SO e no *shutdown* do SO (/dev/urandom/).

Note-se também que as fontes de aleatoriadade do ambiente para estes dois dispositivos são os tempos entre interrupções, *keystrokes* do teclado, cliques do rato e outros processos não determinísticos e difíceis de medir.

1. head -c 32 /dev/random | openssl enc -base64
2. head -c 64 /dev/random | openssl enc -base64
3. head -c 1024 /dev/random | openssl enc -base64
4. head -c 1024 /dev/urandom | openssl enc -base64

Os primeiros três pontos (1,2 e 3) contêm 32, 64 e 1024 *bytes* do /dev/random, respetivamente. Este usa o processamento referido anteriormente e dado que dependem de eventos que causem entropia, o tempo é demorado. Para obter os *bytes* pedidos, o tempo aumenta, ou seja, é necessário esperar que a *pool* de entropia encha, isto é, bloqueie.

No ponto 4, como fonte de aleatoriedade o /dev/urandom, é mais rápida, dado que não bloqueia pelos motivos mencionados acima (a *seed* é alimentada por eventos do sistema). Para além disso, este usa um algoritmo para gerar a sequência.

1.2 Pergunta P1.2

No caso da pergunta anterior, a entropia é obtida de processos do sistema, muitas vezes causados pelo utilizador e em sistemas em que não existe interação com o utilizador, como servidores, o que pode ser um problema. No entanto, a entropia pode ser obtida através do *hardware*, em geradores de números aleatórios em *hardware*.

- ruído térmico
- efeito fotoelétrico
- entre outros

O *hardware* normalmente inclui o uso de um transdutor para manter algum aspecto de fenómenos físicos num sinal elétrico, um amplificador para aumentar a amplitude das flutuações de ruído para medição e um conservador de analógico para digital para converter o *output* num número digital. Para isto, o Linux providencia o *daemon rndg*, de forma a alimentar a entropia nas *pools* aleatórias.

No entanto, nem sempre é possível ter este tipo de *hardware*, como também o *rndg* não possui uma eficiência considerada ótima, dado que é baseado em temporizadores. De igual modo, pode haver problemas com o *hardware*, dado que o *rndg* não verifica o seu *input*.

O *haveged daemon* obtém sequências imprevisíveis de números aleatórios, com base na entropia de eventos externos baseados no algoritmo HAVEGE. Este usa elementos e processos dos processadores para obter entropia. Os processadores multiescalares modernos possuem mecanismos para otimizar a performance: vários níveis de cache, uso de *pipelines*, paralelismo ao nível da instrução, entre outros não menos relevantes.
Dado que estes componentes não fazem parte da arquitetura do sistema, ou seja, os resultados de uma aplicação normal não dependem desses componentes, como também são voláteis e não podem ser diretamente monitorizados pelo utilizador. Assim, cada invocação ao sistema operativo modifica milhares destes estados binários voláteis.

O *haveged* "planta"a fonte de aleatoriedade do sistema (/dev/random) com estes estados binários. No entanto, não o faz para o /dev/urandom.

Executaram-se os seguintes comandos:

- head -c 1024 /dev/random | openssl enc -base64
- head -c 1024 /dev/urandom | openssl enc -base64

Conforme explicado acima, o comando /dev/random foi executado com maior velocidade. Todavia, o comando /dev/urandom não sofreu alterações praticamente com o mesmo tempo que no exercício anterior.

1.2.1 Experiência 1.3

Analisando o código do ficheiro generateSecret-app.py, evidenciou-se que este utiliza shamirsecret.generateSecret(length) para gerar o segredo a ser utilizado, sendo que essa função se encontra no ficheiro shamirsecret do módulo evotum.Cripto.
Se se analisar o código do ficheiro referido, pode-se inferir que, devido ao uso do generateRandomData, irá-se obter, inicialmente, um s com tamanho específico gerado, utilizando uma função aleatória urandom.

Posteriormente, irão ser eliminados todos os caracteres de s que não contenham letras ascii ou dígitos.

De forma a s não ser limitado somente a letras e dígitos, pode-se alterar o código de modo a, em vez de retirar todos os elementos que não são "reconhecíveis", converter para Base64. Desta forma, é reconhecido pelo terminal.

2 Exercício 2: Partilha/Divisão de segredo (*Secret Sharing/Splitting*)

2.1 Pergunta P2.1

2.1.1 A)

Os comandos representados abaixo permitem gerar a chave privada, utilizada para assinar o ficheiro JWT que é retornado:

```
$ openssl genrsa -aes128 -out private-key.pem 1024
```

Posteriormente, executar o ficheiro *python*, dividindo em oito partes, com um *quorom* de 5, com *uid* 0 e a *private key* criada anteriormente.

```
$ python createSharedSecret-app.py 8 5 0 private-key.pem
```

Inicialmente, é pedido ao utilizador uma *passphrase* e um segredo.
Posteriormente, é utilizada a função createSharedSecretComponents que recebe como parâmetros os valores recebidos anteriormente (o segredo, o número de vezes em que irá ser dividido, o *quorum*, o *id* associado a este segredo, a *private key* e a *passphrase*).

Esta função recorre ao *package* eVotUM para dividir o segredo em "n"partes, onde qualquer "*quorum*"partes permite reconstruir o segredo original, e começa por dividir o segredo em "n"partes, fazendo o *encode* em hexadecimal, obtendo assim um *array* com todas as partes do segredo. De seguida, é gerado um *hash* com SHA256 para cada uma das partes.

Para finalizar, é criado um ficheiro jwt com um *array* com todas as partes, assinado com a *private-key* recebida na fase inicial.

No caso de existir algum erro neste passo, é retornado o erro, caso contrário a função termina com sucesso e são devolvidos para o *output* todas as partes, bem como o número do componente associado.

2.1.2 B)

```
$ openssl req -key private-key.pem -new -x509 -days 365 -out key.crt
```

O ficheiro recoverSecretFromComponents-app.py difere de recoverSecretFromAllComponents-app.py no número de componentes que pode receber.

Ou seja, no caso do ficheiro recoverSecretFromAllComponents-app.py é necessário introduzir todos os componentes que foram anteriormente gerados, incluindo o certificado gerado através do comando acima descrito.

Por outro lado, o recoverSecretFromComponents-app.py não necessita de todos os componentes, apesar de que continua a ser mandatória a introdução de um número de componentes superior ao *quorum* introduzido no ficheiro de geração.

O excerto apresentado abaixo está presente no *package* importado (eVotUM) e é a principal diferença entre as funções presentes nos dois ficheiros.

```
if (allComponents):
    if (len(shamirSecretComponents) != nShares):
        # Invalid number of components
        return 14, None
```

O ficheiro recoverSecretFromAllComponents-app.py poderá ser necessário utilizar em situações onde é necessário que todas as partes envolvidas confirmem a ação como, por exemplo, numa transferência bancária em que é obrigatória a confirmação de todos os membros da conta.

Já o ficheiro recoverSecretFromComponents-app.py poderá ser útil num caso em que não sejam precisos todos os envolvidos, mas apenas em número superior ao representado pelo *quorum*.

3 Exercício 3: *Authenticated Encryption*

3.1 Pergunta P3.1

Em primeiro lugar, será necessário escolher a abordagem a adoptar para se implementar *Authenticated Encryption*.

Das três abordagens existentes, *Encrypt-then-MAC*, *MAC-then-Encrypt* e *Encrypt-and-MAC*, sugere-se a implementação *Encrypt-then-MAC*, uma vez que esta não associa o texto limpo ao MAC.

De seguida, após cogitação das possíveis funções a serem utilizadas no algoritmo, para além da API fornecida, obteve-se as seguintes:

- **obterIDUtilizador()** - função que retorna o identificador do utilizador.
- **verificarPagamentoAnuidade(IDutilizador)** - função que verifica se o utilizador em questão pagou a anuidade do serviço. Esta recebe como *input* o identificador do utilizador em questão e retorna 0, caso este a tenha pago, e -1 caso contrário.
- **dataAtualFormatada()** - função que retorna a data do dia atual no formato "anos.mes.dia".
- **obterChave(dataAtualFormatada)** - função que obtém a chave de cifra a ser utilizado no dia atual. Esta recebe como *input* a data do dia atual no formato "ano.mes.dia"e retorna a chave de cifra a ser utilizada.

De referir que em ambos os contextos de cifragem e decifragem, considerou-se fulcral verificar se o utilizador pagou a anuidade.

No que concerne ao algoritmo de cifragem, após ser verificado que o utilizador em questão tem o seu pagamento regularizado, através da função verificarPagamentoAnuidade, sugere-se a cifragem do segredo e da etiqueta, individualmente, através da função cifra já existente na API.

De seguida, recomenda-se a efetuação da função hmac já existente na API com uma chave à escolha e com o segredo e a etiqueta cifrados, individualmente. Nesse sentido, propõem-se o seguinte:

```
def cifragem(segredo, etiqueta) :
    utilizadorID = obterIDUtilizador()
    if (verificarPagamentoAnuidade(utilizadorID) == -1):
        return -1  # utilizador não pagou a anuidade

    ct = cifra(segredo)
    label = cifra(etiqueta)
    hmacCt = hmac(k, ct)
    hmacLabel = hmac(k, label)

    return (dataAtualFormatada(), ct, label, hmacCt, hmacLabel)
```

Figura 1: Algoritmo cifragem

Já no que diz respeito ao algoritmo de decifragem, após ser verificado que o utilizador em questão tem o seu pagamento regularizado, através da função verificarPagamentoAnuidade, sugere-se a obtenção da chave utilizada no dia em que o secreto e a etiqueta foram cifrados, através da função obterChave.

De seguida, recomenda-se que seja recalculado o hmac do segredo cifrado e da etiqueta cifrado e que seja feita a verificação entre estes e os que já tinham sido anteriormente feitos.

Caso fossem iguais, tanto no caso do segredo como da etiqueta, seriam decifrados.

```
def decifragem(ct, labelCt, hmacCt, hmacLabel, data) :
      utilizadorID = obterIDUtilizador()
      if (verificarPagamentoAnuidade(utilizadorID) == -1):
            return -1  # utilizador não pagou a anuidade

      key = obterChave(data)
      hmac1 = (k, ct)
      hmac2 = (k, labelCt)

      if((hmac1 == hmacCt) and (hmac2 == hmacLabel)):
            segredo = decifra(ct, key)
            etiqueta = decifra(labelCt, key)

            return (segredo, etiqueta)


      return -1
```

Figura 2: Algoritmo decifragem

4 Exercício 4: Algoritmos e tamanhos de chaves

4.1 Pergunta P4.1

Seguindo o *site* disponibilizado, encontrou-se diversos provedores de certificados confiáveis, sendo o específico para o nosso tema o da Croácia, para a EC "Financijska agencija".

- `https://webgate.ec.europa.eu/tl-browser/#/tl/HR/0`

Seguindo o *site*, encontrou-se diversos serviços de confiança. Ao abrir o separador "*Qualified certificate for eletronic signature*", encontrou-se vários certificados, tendo sido um deles revogado, outro que utiliza o algoritmo SHA1 com RSA e outros dois utilizam SHA256 com a cifra RSA também, sendo a cifra utilizada para a chave pública, tendo esta um tamanho de 2048 *bits*.

O certificado utilizado tem a validade de 21 de Julho de 2023 11:27:43 GMT.

- Certificado criado:


```
-----BEGIN CERTIFICATE-----
MIIETTCCAzWgAwIBAgIEPxvOITANBgkqhkiG9w0BAQUFADAqMQswCQYDVQQGE
wJIUjENMAsGA1UEChMERklOQTEMMAoGA1UECxMDUkRDMB4XDTAzMDcyMTEwNT
c0M1oXDTIzMDcyMTExMjc0M1owKjELMAkGA1UEBhMCSFIxDTALBgNVBAoTBEZ
JTkExDDAKBgNVBAsTA1JEQzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoC
ggEBAOhno+gYjlsPDZk6gIOtK4YkRKXTEibZ1L3aBOqiFSfno+aL10UKm9R8y
sPJM94uafwlF/tgIG9NP1nqGgwFqn6uH/DCPB2d7mU0ZghqDywCWEdDAfRpJ
g3JjxdPK/piADvMf3oSCPKzy0Da9XKCa69mB6vplikhyxzBeYtzfhC7Y/Bquj
5q6EzK5krYvvHukdhfVDiz/PHp8n3GIJ01zRvm7YbHMRR0rW87lT8/wXe76HS
zvV75fI9ksBWCddi+T3D5MRE+irkL1OIzfIDna09CYyeBRE7LDCo5ZNPuuxHs
```

```
hc4/FaAOhMpKz/rEdfXFitBS+bDD8LIDhZP5NQWoBOCAwEAAaOCAXkwggF1MA
8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgEGMBEGCWCGSAGG+EIBAQQ
EAwIABzCBygYDVR0fBIHCMIG/MEGgP6A9pDswOTELMAkGA1UEBhMCSFIxDTAL
BgNVBAoTBEZJTkExDDAKBgNVBAsTA1JEQzENMAsGA1UEAxMEQ1JMMTBToFGgT
4ZNbGRhcDovL3JkYy1sZGFwLmZpbmEuaHIvb3U9UkRDLG89RklOQSxjPUhSP2
NlcnRpZmljYXRlUmV2b2NhdGlvbkxpc3QlM0JiaW5hcnkwJaAjoCGGH2h0dHA
6Ly9yZGMuZmluYS5oci9jcmxzL3JkYy5jcmwwKwYDVR0QBCQwIoAPMjAwMzA3
MjExMDU3NDNagQ8yMDIzMDcyMTExMjc0M1owEwYDVR0jBAwwCoAIR0UAbvBXp
sAwEQYDVR0OBAoECEdFAG7wV6bAMB0GCSqGSIb2fQdBAAQQMA4bCFY2LjA6NC
4wAwIEkDANBgkqhkiG9w0BAQUFAAOCAQEAVj5rqGDWAT5d84BZhC80ljoO8KH
4pX/MtB5ugJ1vsJb+iIZYBMY2EORHMfLdqkMSi7TlRicvK4AxZstMtHqv6JjT
kmeWiLBt2a6oHBf8EA1AkjPgIVlXGARu0/uvfkNBm4DUvRGpiJKsxG00jkAKK
klTx2S2iAdcacj89IEv8ZEWi+alrdnW6ceuwd2K1ctmzovtXZl1BKckHdJ45T
Gk3mr9bJUvcIxPoWvPdquN7FAzoV3GFDttWGF0QMi7zymCdJW+8TbmH4xiriM
xZn4NiWEUI92xX1jTs+7rf2JA2Z53JRGD4rfTaW+wTeYnIQ/LJ32rJoNd/Bv7
FXOyLD6arA==
-----END CERTIFICATE-----
```

- Após execução do comando:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1058786849 (0x3f1bce21)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C = HR, O = FINA, OU = RDC
        Validity
            Not Before: Jul 21 10:57:43 2003 GMT
            Not After : Jul 21 11:27:43 2023 GMT
        Subject: C = HR, O = FINA, OU = RDC
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:e8:67:a3:e8:18:8e:5b:0f:0d:99:3a:80:83:ad:
                    2b:86:24:44:a5:d3:12:26:d9:d4:bd:da:04:ea:a2:
                    15:27:e7:a3:e6:8b:d7:45:0a:9b:d4:7c:ca:c3:c9:
                    33:de:2e:69:fc:25:17:fb:60:20:6f:4d:3f:59:ea:
                    1a:0c:05:aa:7e:ae:1f:f0:c2:3c:1d:9d:ee:65:34:
                    66:08:6a:e0:3c:b0:09:61:1d:0c:07:d1:a4:98:37:
                    26:3c:5d:3c:af:e9:88:00:ef:31:fd:e8:48:23:ca:
                    cf:2d:03:6b:d5:ca:09:ae:bd:98:1e:af:a6:58:a4:
                    87:2c:73:05:e6:2d:cd:f8:42:ed:8f:c1:aa:e8:f9:
                    ab:a1:33:2b:99:2b:62:fb:c7:ba:47:61:7d:50:e2:
                    cf:f3:c7:a7:c9:f7:18:82:74:d7:34:6f:9b:b6:1b:
                    1c:c4:51:d2:b5:bc:ee:54:fc:ff:05:de:ef:a1:d2:
                    ce:f5:7b:e5:f2:3d:92:c0:56:09:d7:62:f9:3d:c3:
                    e4:c4:44:fa:2a:e4:2f:53:88:cd:f2:03:9d:ad:3d:
                    09:8c:9e:05:11:3b:2c:30:a8:e5:93:4f:ba:ec:47:
                    b2:17:38:fc:56:80:3a:13:29:2b:3f:eb:11:d7:d7:
                    16:2b:41:4b:e6:c3:0f:c2:c8:0e:16:4f:e4:d4:16:
                    a0:1d
```

```
            Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            Netscape Cert Type:
                SSL CA, S/MIME CA, Object Signing CA
            X509v3 CRL Distribution Points:

                Full Name:
                  DirName:C = HR, O = FINA, OU = RDC, CN = CRL1

                Full Name:
                  URI:ldap://rdc-ldap.fina.hr/ou=RDC,o=FINA,
                  c=HR?certificateRevocationList%3Bbinary

                Full Name:
                  URI:http://rdc.fina.hr/crls/rdc.crl

            X509v3 Private Key Usage Period:
                Not Before: Jul 21 10:57:43 2003 GMT,
                Not After: Jul 21 11:27:43 2023 GMT
            X509v3 Authority Key Identifier:
                keyid:47:45:00:6E:F0:57:A6:C0

            X509v3 Subject Key Identifier:
                47:45:00:6E:F0:57:A6:C0
            1.2.840.113533.7.65.0:
                0...V6.0:4.0....
    Signature Algorithm: sha1WithRSAEncryption
         56:3e:6b:a8:60:d6:01:3e:5d:f3:80:59:84:2f:34:96:3a:0e:
         f0:a1:f8:a5:7f:cc:b4:1e:6e:80:9d:6f:b0:96:fe:88:86:58:
         04:c6:36:10:e4:47:31:f2:dd:aa:43:12:8b:b4:e5:46:27:2f:
         2b:80:31:66:cb:4c:b4:7a:af:e8:98:d3:92:67:96:88:b0:6d:
         d9:ae:a8:1c:17:fc:10:0d:40:92:33:e0:21:59:57:18:04:6e:
         d3:fb:af:7e:43:41:9b:80:d4:bd:11:a9:88:92:ac:c4:6d:34:
         8e:40:0a:2a:49:53:c7:64:b6:88:07:5c:69:c8:fc:f4:81:2f:
         f1:91:16:8b:e6:a5:ad:d9:d6:e9:c7:ae:c1:dd:8a:d5:cb:66:
         ce:8b:ed:5d:99:75:04:a7:24:1d:d2:78:e5:31:a4:de:6a:fd:
         6c:95:2f:70:8c:4f:a1:6b:cf:76:ab:8d:ec:50:33:a1:5d:c6:
         14:3b:6d:58:61:74:40:c8:bb:cf:29:82:74:95:be:f1:36:e6:
         1f:8c:62:ae:23:31:66:7e:0d:89:61:14:23:dd:b1:5f:58:d3:
         b3:ee:eb:7f:62:40:d9:9e:77:25:11:83:e2:b7:d3:69:6f:b0:
         4d:e6:27:21:0f:cb:27:7d:ab:26:83:5d:fc:1b:fb:15:73:b2:
         2c:3e:9a:ac
```