Engenharia de Segurança

Trabalho Prático 1

Afonso Fontes (pg35389)

Bruno Carvalho (a67847)

Mariana Carvalho (a67635)

16 de Fevereiro de 2018

Universidade do Minho

Números aleatórios/pseudoaleatórios

Pergunta 1.1

Na primeira tentativa de execução dos respetivos comandos, obter 1024 bytes pseudo-aleatórios revelou ser considerávelmente mais fácil através do uso da primitiva /dev/urandom, em oposição ao que aconteceu quando tentamos gerar os bytes através da primitiva **/dev/random**. Este fenómeno é explicado devido à diferente fonte de entropia da qual se alimentam as duas primitivas de forma a serem capazes de gerar a sequência aleatória requerida. A primeira primitiva utilizada faz uso de fontes de "verdadeira"entropia, sendo que é bloqueada enquanto que esta não seja suficiente para satisfazer o pedido de números aleatórios. Já a primitiva **urandom** é um gerador não bloqueado de números pseudo-aleatórios, sendo que este, caso não seja capaz de obter *input* aleatório suficiente, é munido de mecanismos que lhe permitem satisfazer o pedido.

Pergunta 1.2

Após instalarmos o **daemon** e executarmos novamente as duas primitivas, verificamos que agora a resposta obtida pelo uso do **/dev/random** foi praticamente imediata em oposição ao que aconteceu anteriormente. Isto acontece devido ao que foi dito na resposta à **Pergunta 1.1** relativamente à fonte de entropia que alimenta os respetivos geradores, anteriormente a demora aquando da geração da sequência era causada pela condição de baixa entropia em que o sistema se encontrava. O daemon utiliza entropia proveniente do *hardware* de forma a fornecer uma maior taxa de *input* para os geradores utilizados.

Pergunta 1.3

O **output** não contém caracteres de pontuação e outros que não pertencem ao alfabeto de 64 caracteres, base para qual foram convertidos os bytes pseudo-aleatórios gerados de forma a serem imprimivéis. Cada 6 bits de **output** são então convertidos para um de 64 caracteres ([A-Z]—[a-z]—[0-9]—"/"+"="), dos quais não fazem parte qualquer caracter de pontuação.

Partilha/Divisão de segredo

Pergunta 2.1 - A

Após análise do programa *createSharedSecret-app.py*, verificou-se que este requer como argumentos o número de partes para dividir o segredo, o quorum, um id e uma chave privada. Assim, foi necessário correr inicialmente o comando **openssl genrsa -aes128 -out mykey.pem 1024**, que gerou um a chave privada rsa e guardou no ficheiro mykey.pem. De seguida foi executado o comando **python createSharedSecret-app.py 7 3 1 mykey.pem**, que "dividiu"o segredo em 7 componentes, sendo que para o segredo ser reconstruído são nescessitadas apenas 3 quaisquer partes. Para recuperar o segredo, correu-se o comando **openssl req -key mykey.pem -new -x509 -days 365 -out mykey.crt** para gerar o certificado de onde poderá ser extraída a chave pública. Por fim, foi executado o comando **python recoverSecretFromComponents-app.py 3 1 mykey.crt** que permitiu recuperar o segredo sendo apenas fornecer 3 dos 7 componentes gerados anteriormente.

Pergunta 2.1 - B

Depois de testados os dois programas, concluiu-se que a diferença entre ambos reside no facto de o programa *recoverSecretFromAllComponents-app.py* necessitar de todas as partes em que o segredo foi "dividido"para conseguir recontruir o segredo, enquanto que com o programa *recoverSecretFromComponents-app.py* é possível a reconstrução utilizando apenas um número de partes igual ao quorum. Em termos práticos a reconstituição do segredo segundo o que foi exposto nas aulas é feita por interpolação, assim só faria sentido utilizar o *recoverSecretFromAllComponents* se este, fosse parte de um sistema em que, para realização de uma tarefa em concreto fosse nescessário a comparticipação de todos os componentes de forma a garantir certos níveis de confiabilidade.

Authenticated Encryption

Pergunta 3.1

A forma mais apropriada de garantir confidencialidade, integridade a autenticidade do segredo é utilizar o HMAC sob a forma *Encrypt then MAC*, ou seja, cifrar todos os dados e calcular o HMAC do cyphertext. Desta forma é garantida a integridade do cyphertext (podemos fazer a verificação do HMAC antes de qualquer operação de decifra) e consequentemente, do plaintext.

As alternativas (menos apropriadas) seriam utilizar o HMAC sob a forma *MAC then Encrypt*, o que não garante a integridade do cyphertext (é necessário decifrar o cyphertext antes de verificar o HMAC) ou *Encrypt and MAC*, que seria a pior solução, pois além de não garantir a integridade do cyphertext, também poderia fornecer a um potencial atacante informações sobre o plaintext (pois o HMAC é calculado sobre o plaintext e concatenado com o cyphertext sem qualquer tipo de cifragem).

Assumindo que a chave do HMAC (k) foi previamente negociada entre o cliente e a empresa, os algoritmos de cifra e decifra ficariam assim (pseudocódigo baseado em python):

```
def cifrar(plaintext, label=None):
        cyphertext = cifra(plaintext + label)
        cyphertext += hmac(k, cyphertext)
        return cyphertext

def decifrar(cyphertext, cypher_date):
        if(hmac(k, cyphertext[:-32]) == cyphertext[-32:]):
                return decifra(cyphertext[:-32], cypher_date)
        else:
                raise Exception("Código HMAC inválido")
```

Para simplificar o código, assumimos que a *cypher_date* já se encontra sob o formato *ano.mes.dia*, que o hardware é capaz de decifrar com a chave correta recebendo a identificação da chave (data de cifragem) e que o HMAC gerado com recurso ao hash SHA256 (32 bytes) se encontra concatenado no fim da mensagem.

Algoritmos e tamanhos de chaves

Pergunta 4.1

"Bundesagentur fuer Arbeit", "D-Trust GmbH", "Deutscher Sparkassen Verlag GmbH"

Pare responder a esta pergunta, foram utilizados os certificados mais recentes de cada uma das entidades da categoria "Qualified certificate for electronic signature". Os certificados foram anali-

sados nas nossas máquinas utilizando o openSSL.

O certificado da entidade "Bundesagentur fuer Arbeit"foi gerado utilizando RSA com chaves de 2048 bits e o algoritmo da assinatura é o RSASSA-PSS (sistema de assinatura probabilístico RSA) com hash SHA256. As entidades "D-Trust GmbH"e "Deutscher Sparkassen Verlag GmbH"utilizam certificados gerados com chaves RSA de 2048 bits e assinados utilizando SHA256RSA.

Os algoritmos e tamanhos de chave utilizados são à partida apropriados. Visto que se tratam de certificados de entidades certificadoras, com validades relativamente alargadas (5 anos), poderia ser utilizado RSA com chaves de 4096 bits (em vez de 2048 bits), aumentando assim a segurança, no entanto, isto teria custos a nivel da performance na verificação das assinaturas.