Escola de Engenharia
Universidade do Minho

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Mestrado em Engenharia Informática
Engenharia de Segurança

Aula 2

17 de Fevereiro de 2020

Grupo 1

Ricardo Pereira a73577

Tiago Ramires pg41101

Braga, 25 de Fevereiro de 2020

1. Números aleatórios/pseudoaleatórios

1.1 Pergunta P1.1

- *head -c 32 /dev/random | openssl enc -base64* - demorou 0.014 segundos;
- *head -c 64 /dev/random | openssl enc -base64* - demorou 0.007 segundos;
- *head -c 1024 /dev/random | openssl enc -base64* - demorou 3 minutos e 11.087 segundos;
- *head -c 1024 /dev/urandom | openssl enc -base64* - demorou 0.011 segundos.

Em sistemas do tipo *Unix* existem funcionalidades que geram números pseudoaleatórios com base no "ruído"recolhido pelos vários controladores do dispositivo. Os arquivos ***/dev/-random*** e ***/dev/urandom*** têm essa finalidade havendo uma ligeira diferença entre os dois. O primeiro (***/dev/random***), bloqueia a obtenção dos *bytes* pseudoaleatórios enquanto a quantidade de entropia requerida não tiver sido atingida e o segundo (***/dev/urandom***), normalmente retorna um resultado imediato facilitando o processo e reutilizando a entropia do sistema.

Assim ficamos a perceber o porquê de os dois primeiros comandos serem rápidos - a quantidade de dados pedidos é relativamente baixa, o que não requer grande entropia. Já o terceiro comando é bem mais demorado, uma vez que a quantidade de dados aleatórios pedidos é elevada e, por esse motivo o sistema bloqueia até que exista entropia suficiente para a geração desses *bytes* pseudoaleatórios. Por fim, o último comando pede o mesmo número de dados mas obriga a que estes sejam gerados a partir do arquivo ***/dev/urandom***, diminuindo substancialmente a qualidade de pseudoaleatoriedade mas garantindo sempre bastante rapidez.

1.2 Pergunta P1.2

- *head -c 1024 /dev/random | openssl enc -base64* - demorou 0.007 segundos;
- *head -c 1024 /dev/urandom | openssl enc -base64* - demorou 0.007 segundos.

O software *HAVEGE* explora as alternâncias dos estados do *hardware* volátil interno e usa-os como fontes de incerteza, isto é, aumenta a quantidade de entropia que já se tinha, e assim, é previsível que o primeiro comando que dantes demorava minutos, seja obtido quase instantaneamente, tal como o segundo. Executando os comandos, as previsões confirmam-se, tendo até

os tempos medidos coincidido.

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

2.1 Pergunta P2.1

O programa em questão pede ao utilizador um determinado segredo que o mesmo queira partilhar e dividir e uma palavra chave para tornar esse segredo acessível apenas por quem a conhecer. Para o reconstituir, o utilizador pode posteriormente desvendar esse segredo utilizando um de outros dois executáveis que podem requer todas ou apenas algumas partes nas quais foi dividido.

Parte A

Para podermos responder à questão, tivemos inicialmente que gerar o par de chaves e construir e gerar o respetivo certificado:

- *openssl genrsa -aes128 -out private-key.pem 1024*
- *openssl req -key private-key.pem -new -x509 -days 365 -out mykey.crt*

Posteriormente, executou-se o comando *python createSharedSecret-app.py 8 5 4110173577 private-key.pem*, onde *8* é o número de partes em que o segredo é dividido, *5* o número de partes necessárias para a reconstrução do segredo, *4110173577* o identificador do segredo e *private-key.pem* o par de chaves necessário para a construção do segredo.

O resultado foram 8 objetos *JWT* que foram escritos no *STDOUT*.

Parte B

Para reconstruirmos o segredo inicial podemos utilizar o executável que necessita apenas de algumas partes em que o segredo foi dividido - *recoverSecretFromComponents-app.py* - ou podemos utilizar o executável que requer todas as partes em que o segredo foi dividido - *recoverSecretFromAllComponents-app.py*.

A principal diferença útil entre um e outro acontece quando existe a necessidade de alteração de segredo que requer a "autorização"de todas as partes que nele intervêm. Por outro lado, se se quiser "facilitar"a alteração do segredo reduz-se o número de partes necessárias para a sua

reconstituição.

4. Algoritmos e tamanhos de chaves

4.1 Pergunta P4.1

- **Algoritmo de assinatura utilizado:** *RSA* com *SHA256*.
- **Algoritmo da chave pública utilizado:** *RSA*.
- **Tamanho da chave pública:** 4096 *bits*

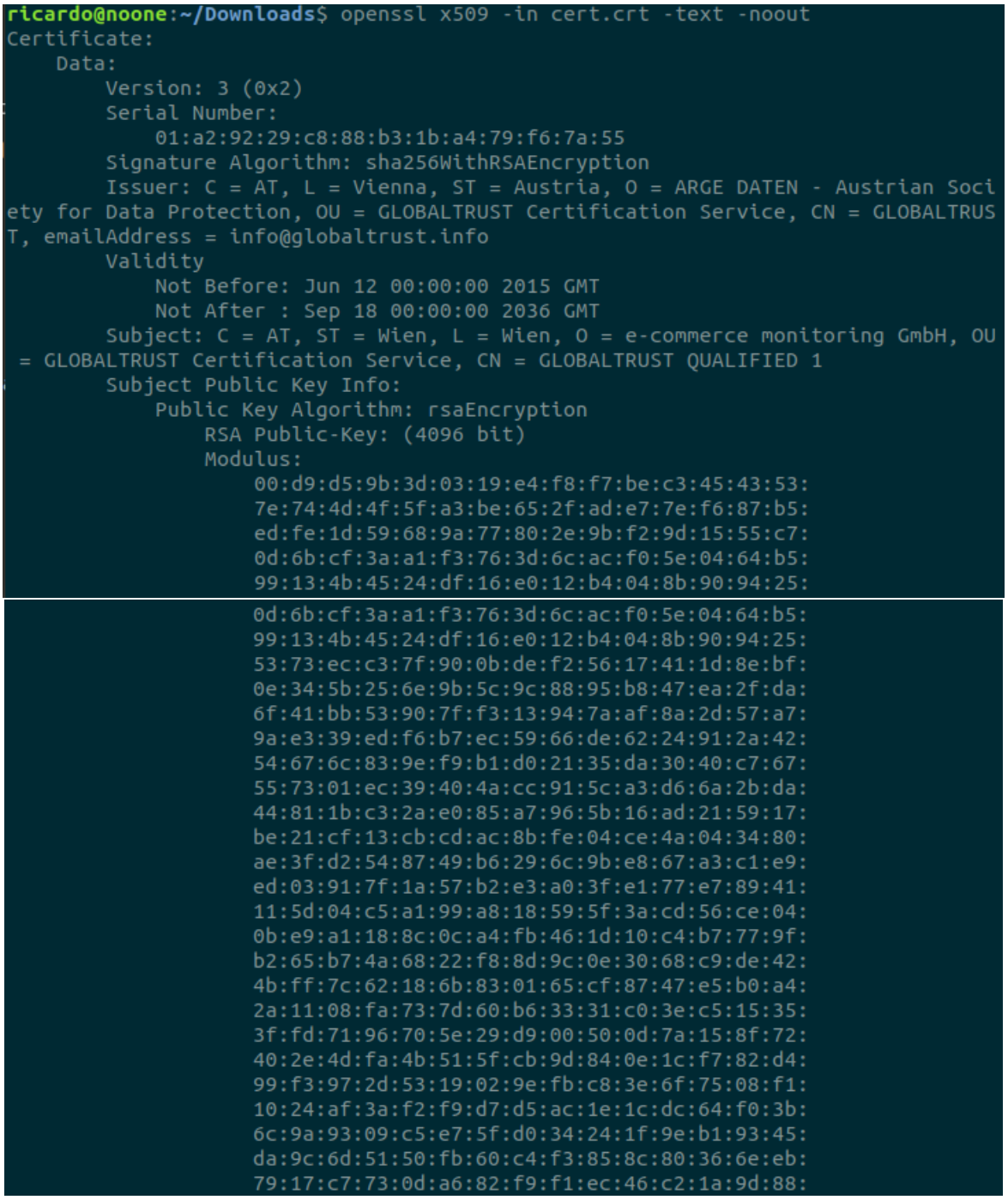

Figura 4.1: 1ª parte do resultado

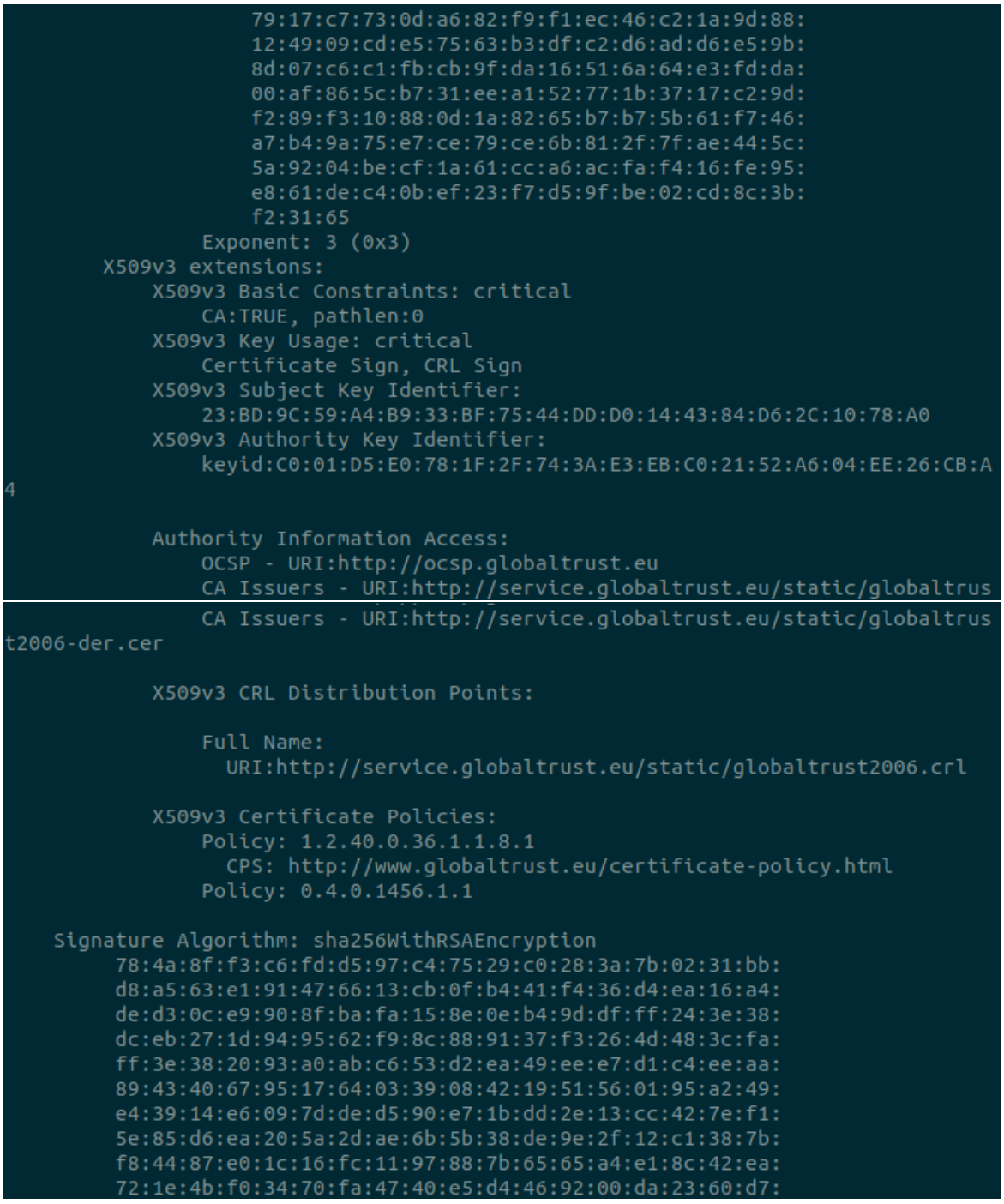

Figura 4.2: 2ª parte do resultado

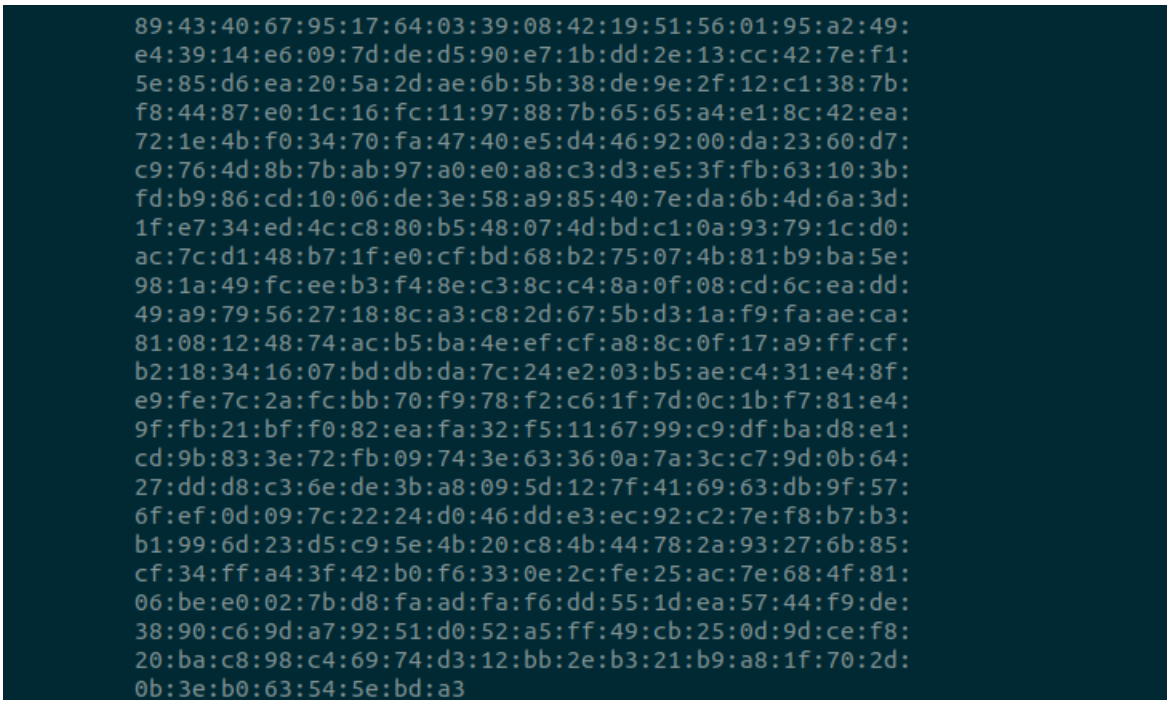

Figura 4.3: 3ª parte do resultado