Aplicação cliente-servidor baseada em KVS com TLS

Tópicos em Redes de Computadores (INFO-7065)

Josiney de Souza (josiney.souza@ifc.edu.br)

UFPR / DInf

30 de Maio de 2023

Sumário

Sumário

A especificação do trabalho

Esta é a segunda avaliação de 2023/1 de Tópicos em Redes de Computadores:

- Fazer uma aplicação cliente/servidor
- Baseada em KVS (Key-Value Store)
- Usar TLS (Transport Layer Security)
- Demonstrar SIGILO
- Demonstrar AUTENTICIDADE
- Demonstrar INTEGRIDADE

Página do componente curricular (disciplina/matéria):
`https://www.inf.ufpr.br/elias/topredes/`

Entrega

O formato de entrega do trabalho:

Relatório: pode ser vídeo, apresentação ou relatório em texto;

- Optei por este vídeo

Entrega

O formato de entrega do trabalho:

Relatório: pode ser vídeo, apresentação ou relatório em texto;

- Optei por este vídeo

Logs de execução: via demonstrações no vídeo ou *logs* em uma página *web*;

- Optei por ambos
- Página: `https://www.inf.ufpr.br/jsouza/`

Entrega

O formato de entrega do trabalho:

Relatório: pode ser vídeo, apresentação ou relatório em texto;

- Optei por este vídeo

Logs de execução: via demonstrações no vídeo ou *logs* em uma página *web*;

- Optei por ambos
- Página: `https://www.inf.ufpr.br/jsouza/`

Código comentado: entrega do código comentado

- Página: `https://www.inf.ufpr.br/jsouza/`
- GitHub: `https://github.com/josiney-souza/ufpr-topicos-redes`

Entrega por e-mail

A organização do sistema/aplicação

O sistema está assim organizado - **códigos**:

Linguagem: Python

02-cliente.py: aplicação cliente - entra em contato com o servidor para solicitar serviços ou recursos;

02-servidor.py: aplicação servidor - recebe as demandas dos clientes e retorna alguma ação;

02-invasor.py: aplicação cliente não autorizada a se comunicar com o servidor;

confs_comuns.py: biblioteca pessoal de funções e configurações comuns aos clientes e ao servidor.

A organização do sistema/aplicação

O sistema está assim organizado - **outros arquivos**:

id_dsa.pub: chave pública criada com `ssh-keygen(1)`

id_dsa: chave privada criada com `ssh-keygen(1)`

- `$ ssh-keygen`

A organização do sistema/aplicação

O sistema está assim organizado - **outros arquivos**:

id_dsa.pub: chave pública criada com `ssh-keygen(1)`

id_dsa: chave privada criada com `ssh-keygen(1)`

- `$ ssh-keygen`

cert-rsa.pem: certificado auto-assinado para uso no servidor

- `$ openssl req -newkey rsa:2048 -nodes -keyout id_dsa -x509 -days 365 -out cert-rsa.pem`

Sumário

Uso normal

O funcionamento normal envolve um menu:

1: Adicionar uma chave e um valor;

2: Consultar uma chave;

3: Consultar todo o banco;

4: Atualizar um valor a partir de uma chave;

5: Apagar uma chave e um valor;

6: Apagar toda a base;

[1] na verdade, qualquer opção não reconhecida. Caso padrão

Uso normal

O funcionamento normal envolve um menu:

1: Adicionar uma chave e um valor;

2: Consultar uma chave;

3: Consultar todo o banco;

4: Atualizar um valor a partir de uma chave;

5: Apagar uma chave e um valor;

6: Apagar toda a base;

7: Demonstrar INTEGRIDADE;

[1] na verdade, qualquer opção não reconhecida. Caso padrão

Uso normal

O funcionamento normal envolve um menu:

- 1: Adicionar uma chave e um valor;
- 2: Consultar uma chave;
- 3: Consultar todo o banco;
- 4: Atualizar um valor a partir de uma chave;
- 5: Apagar uma chave e um valor;
- 6: Apagar toda a base;
- 7: Demonstrar INTEGRIDADE;
- ?: Mostrar o menu de ajuda[1];
- ad: **A**tivar a **D**epuração/*Debug*;
- dd: **D**ESativar a **D**epuração/*Debug*;
- 0: Encerrar a conexão e sair.

[1] na verdade, qualquer opção não reconhecida. Caso padrão

Sumário

Demonstrando SIGILO

Para demonstrar que o sistema cliente/servidor possui a característica de SIGILO do TLS:

1. Iniciar o Wireshark;
   - `$ sudo wireshark`
2. Capturar os pacotes TCP relacionados à porta 8003
   - regra `tcp.port > 8000`[2]
3. Iniciar o servidor;
   - `$ python3 02-servidor.py`
4. Iniciar o cliente.
   - `$ python3 02-cliente.py`

[2] É de mais de 8.000!!! `https://www.youtube.com/watch?v=S8IL1JJuJiE`

Sumário

Demonstrando AUTENTICIDADE

Para demonstrar a AUTENTICIDADE com o uso do TLS no sistema cliente/servidor:

1. Iniciar o servidor;
   - `$ python3 02-servidor.py`
2. Iniciar o **invasor**.
   - Sem carregar certificados
   - `$ python3 02-invasor.py`

Demonstrando AUTENTICIDADE

Para demonstrar a AUTENTICIDADE com o uso do TLS no sistema cliente/servidor:

1. Iniciar o servidor;
   - `$ python3 02-servidor.py`
2. Iniciar o **invasor**.
   - Sem carregar certificados
   - `$ python3 02-invasor.py`
   - Carregando certificado errado
   - `$ python3 02-invasor.py`

Sumário

Demonstrando INTEGRIDADE

1. Iniciar o servidor;
   - `$ python3 02-servidor.py`
2. Iniciar o cliente;
   - `$ python3 02-cliente.py`
3. Escolher a opção 7 do menu de interação.

Demonstrando INTEGRIDADE

0.1 Cliente solicita serviço (aqui: todo o banco)

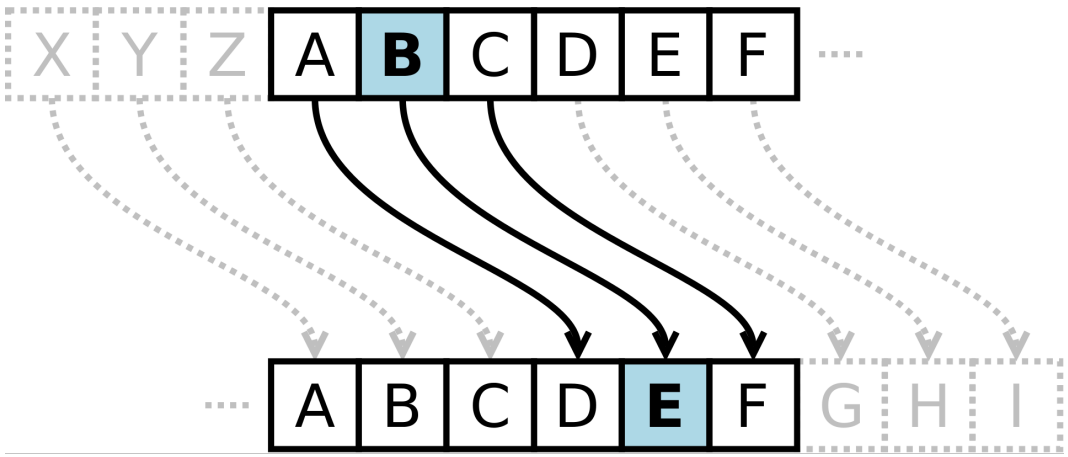
0.2 Simula captura de mensagem criptografada no cliente

- Cifra de Cesar
- Chave assimétrica (chave pública cliente - 401)
- Cifra de Cesar

Demonstrando INTEGRIDADE: Cifra de Cesar

Cifra de substituição:

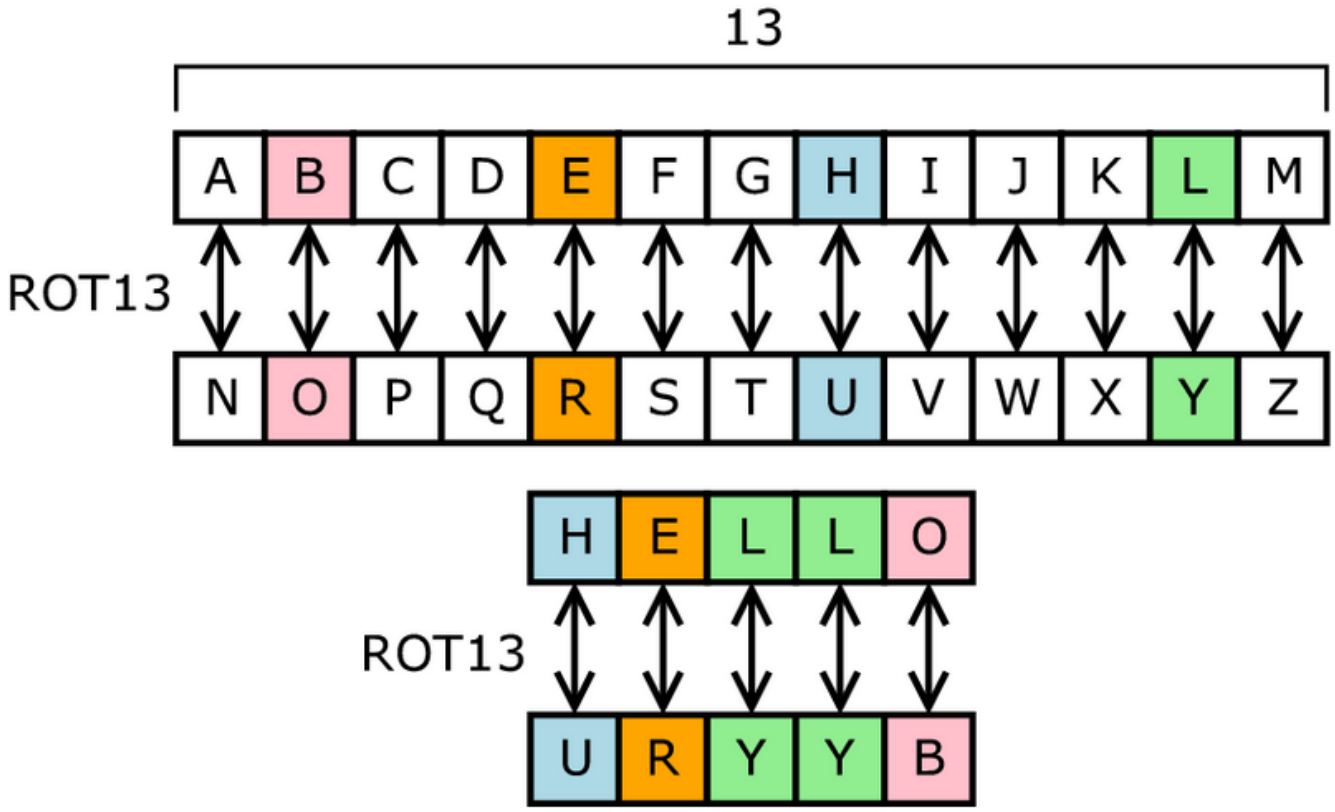
- Texto normal: ABCDEFGHIJKLMNOPQRSTUVWXYZ
- Texto cifrado: DEFGHIJKLMNOPQRSTUVWXYZABC
- Chave: 3



Fonte: `https://pt.wikipedia.org/wiki/Cifra_de_C%C3%A9sar#/media/Ficheiro:Caesar3.svg`

Demonstrando INTEGRIDADE: Cifra de Cesar

Cifra de Cesar com rotação 13 (ROT13):



Fonte:
https://pt.wikipedia.org/wiki/ROT13#/media/Ficheiro:ROT13.png

Demonstrando INTEGRIDADE: Cifra de Cesar

Cifra de Cesar em Python - codificando:

```
import codecs
print(codecs.encode("Olá, você!", "rot_13"))
```

- "Olá, você!" $\rightarrow$ "Byá, ibpê!"
- "Ççsenha: 123mudar!" $\rightarrow$ "Ççfraun: 123zhqne!"

Demonstrando INTEGRIDADE: Cifra de Cesar

Cifra de Cesar em Python - codificando:

```
import codecs
print(codecs.encode("Olá, você!", "rot_13"))
```

- "Olá, você!" $\rightarrow$ "Byá, ibpê!"
- "Ççsenha: 123mudar!" $\rightarrow$ "Ççfraun: 123zhqne!"

Cifra de Cesar em Python - DEcodificando

```
import codecs
print(codecs.decode("Wbfvarl qr Fbhmn", "rot_13"))
```

- "Wbfvarl qr Fbhmn" $\rightarrow$ "Josiney de Souza"

Demonstrando INTEGRIDADE: Aritmética modular

- Aritmética para números inteiros
- Números retrocedem quando chega a um máximo
- $37 \bmod 4 = 1$

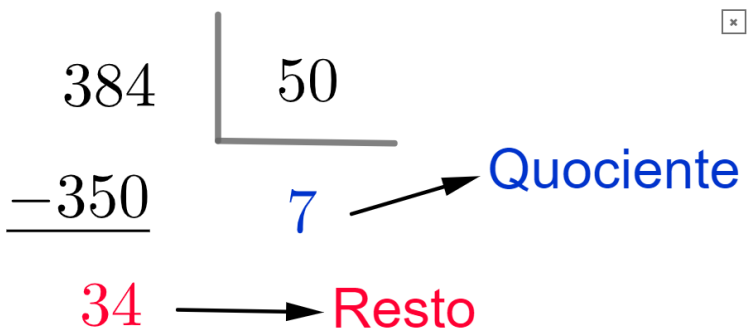
37 dividendo
4 divisor
9 quociente
1 resto

Demonstrando INTEGRIDADE: Aritmética modular

- Aritmética para números inteiros
- Números retrocedem quando chega a um máximo
- $37 \bmod 4 = 1$

37	dividendo
4	divisor
9	quociente
1	resto

Outro exemplo



Fonte:
https://escolaeducacao.com.br/para-que-serve-o-resto-da-divisao/

Demonstrando INTEGRIDADE: Inverso multiplicativo

Descobrir um x e um y tal que x*y mod 1000 = 1

- x * y mod n = 1
- 401 * 601 mod 1000 = 1
- 191 * 911 mod 1000 = 1
- 167 * 503 mod 1000 = 1

Demonstrando INTEGRIDADE: Inverso multiplicativo

Descobrir um x e um y tal que x*y mod 1000 = 1

- x * y mod n = 1
- 401 * 601 mod 1000 = 1
- 191 * 911 mod 1000 = 1
- 167 * 503 mod 1000 = 1

Programa para descobrir o inverso multiplicativo de um número mod 1000:

```
valor = int(input("Que numero quer descobrir o inverso modular
multiplicativo? "))
for i in range(1000):
    if ((valor*i)%1000 == 1):
        print(valor, "x", i, "mod 1000 =", (valor*i)%1000)
```

Demonstrando INTEGRIDADE: Inverso multiplicativo

Criptografando com uma chave ...

- Caractere 'A' $\rightarrow$ 65 na tabela ASCII
- 65 * 191 = 12.415
- 12.415 mod 1000 = 415

Demonstrando INTEGRIDADE: Inverso multiplicativo

Criptografando com uma chave ...

- Caractere 'A' $\rightarrow$ 65 na tabela ASCII
- 65 * 191 = 12.415
- 12.415 mod 1000 = 415

Descriptografando com outra chave ...

- Número 415
- 415 * 911 = 378.065
- 378.065 mod 1000 = 65
- Caractere 'A' é recuperado

Fonte: `https://www.inf.ufpr.br/elias/topredes/Sec4TopRedes23.pdf`

Demonstrando INTEGRIDADE

0.1 Cliente solicita serviço (aqui: todo o banco)

0.2 Simula captura de mensagem criptografada no cliente

- Cifra de Cesar
- Chave assimétrica (chave pública cliente - 401)
- Cifra de Cesar

Demonstrando INTEGRIDADE

0.1 Cliente solicita serviço (aqui: todo o banco)

0.2 Simula captura de mensagem criptografada no cliente

- Cifra de Cesar
- Chave assimétrica (chave pública cliente - 401)
- Cifra de Cesar

1. simula uma descriptografia com a cifra de Cesar

Demonstrando INTEGRIDADE

0.1 Cliente solicita serviço (aqui: todo o banco)

0.2 Simula captura de mensagem criptografada no cliente

- Cifra de Cesar
- Chave assimétrica (chave pública cliente - 401)
- Cifra de Cesar

1. simula uma descriptografia com a cifra de Cesar
2. simula descriptografar usando o inverso multiplicativo da operação de criptografia
   - Assume invasor com chave pública 191 $\rightarrow$ ERRO!

Demonstrando INTEGRIDADE

0.1 Cliente solicita serviço (aqui: todo o banco)

0.2 Simula captura de mensagem criptografada no cliente

- Cifra de Cesar
- Chave assimétrica (chave pública cliente - 401)
- Cifra de Cesar

1. simula uma descriptografia com a cifra de Cesar
2. simula descriptografar usando o inverso multiplicativo da operação de criptografia
   - Assume invasor com chave pública 191 $\rightarrow$ ERRO!
   - Assume cliente com chave pública 401 $\rightarrow$ ERRO!

Demonstrando INTEGRIDADE

0.1 Cliente solicita serviço (aqui: todo o banco)

0.2 Simula captura de mensagem criptografada no cliente

- Cifra de Cesar
- Chave assimétrica (chave pública cliente - 401)
- Cifra de Cesar

1. simula uma descriptografia com a cifra de Cesar
2. simula descriptografar usando o inverso multiplicativo da operação de criptografia
   - Assume invasor com chave pública 191 $\rightarrow$ ERRO!
   - Assume cliente com chave pública 401 $\rightarrow$ ERRO!
   - Assume invasor com chave privada 911 $\rightarrow$ ERRO!

Demonstrando INTEGRIDADE

0.1 Cliente solicita serviço (aqui: todo o banco)

0.2 Simula captura de mensagem criptografada no cliente

- Cifra de Cesar
- Chave assimétrica (chave pública cliente - 401)
- Cifra de Cesar

1. simula uma descriptografia com a cifra de Cesar
2. simula descriptografar usando o inverso multiplicativo da operação de criptografia
   - Assume invasor com chave pública 191 $\rightarrow$ ERRO!
   - Assume cliente com chave pública 401 $\rightarrow$ ERRO!
   - Assume invasor com chave privada 911 $\rightarrow$ ERRO!
   - Assume cliente com chave privada 601 $\rightarrow$ OK
3. simula mais uma descriptografia com a cifra de Cesar

Demonstrando INTEGRIDADE: Simulando alteração da mensagem

0.1 Cliente solicita serviço (aqui: todo o banco)

0.2 Simula captura de mensagem criptografada no cliente

- Cifra de Cesar
- Chave assimétrica (chave pública cliente - 401)
- Cifra de Cesar

1. simula alteração de mensagem nos índices 1, 3, 5, 7, 9

Demonstrando INTEGRIDADE: Simulando alteração da mensagem

0.1 Cliente solicita serviço (aqui: todo o banco)

0.2 Simula captura de mensagem criptografada no cliente

- Cifra de Cesar
- Chave assimétrica (chave pública cliente - 401)
- Cifra de Cesar

1. simula alteração de mensagem nos índices 1, 3, 5, 7, 9
2. simula uma descriptografia com a cifra de Cesar
   - ERRO!

Demonstrando INTEGRIDADE: Simulando alteração da mensagem

0.1 Cliente solicita serviço (aqui: todo o banco)

0.2 Simula captura de mensagem criptografada no cliente

- Cifra de Cesar
- Chave assimétrica (chave pública cliente - 401)
- Cifra de Cesar

1. simula alteração de mensagem nos índices 1, 3, 5, 7, 9
2. simula uma descriptografia com a cifra de Cesar
   - ERRO!
3. simula descriptografar usando o inverso multiplicativo da operação de criptografia
   - Assume cliente com chave privada 601 $\rightarrow$ "OK"
4. simula mais uma descriptografia com a cifra de Cesar

Aplicação cliente-servidor baseada em KVS com TLS

Tópicos em Redes de Computadores (INFO-7065)

Josiney de Souza (josiney.souza@ifc.edu.br)

UFPR / DInf

30 de Maio de 2023