UNIFESSPA
UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ

Universidade Federal do Sul e Sudeste do Pará

Sistemas Distribuídos

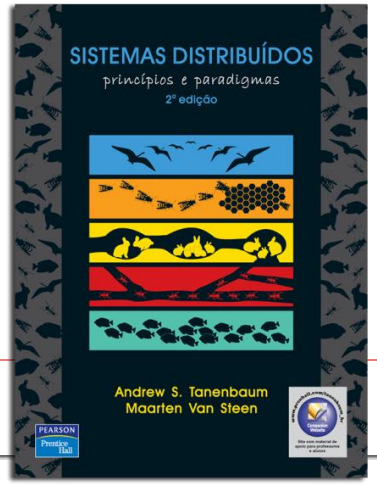
Prof.: Warley Junior
wmvj@unifesspa.edu.br

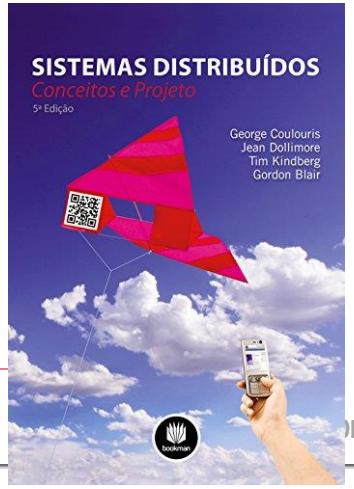
Agenda

- AULA 5:
- Segurança
  - Modelo de Segurança
  - Ameaças e formas de ataques
  - Criptografia
  - Criptografia assimétrica com java

Leitura Prévia

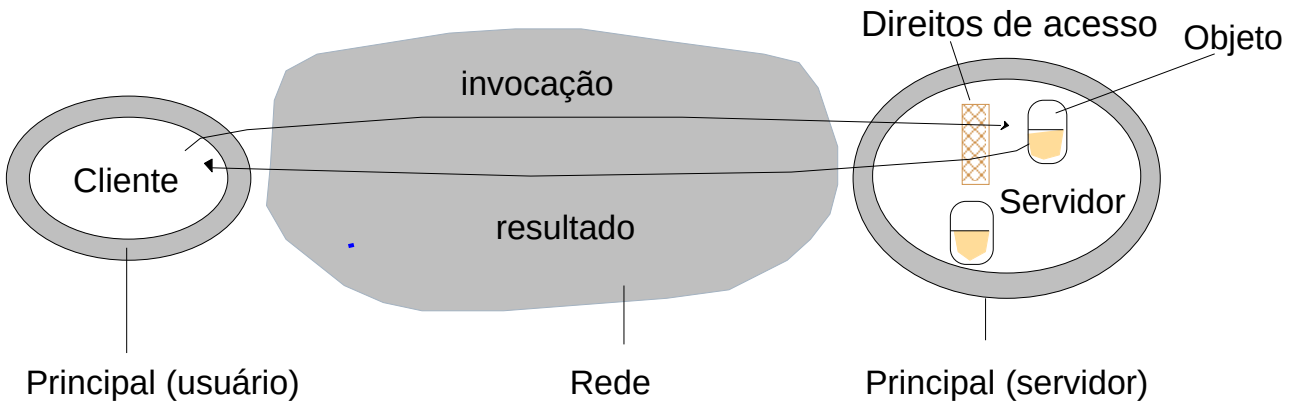
- COULOURIS, George. Sistemas distribuídos: conceitos e projetos. 5ª ed. Porto Alegre: Bookman, 2013.
  - Capítulo 2 e 11.
- TANENBAUM, Andrew S. Sistemas distribuídos: princípios e paradigmas. 2ª ed. São Paulo: Pearson Prentice Hall, 2007.
  - Capítulo 9.





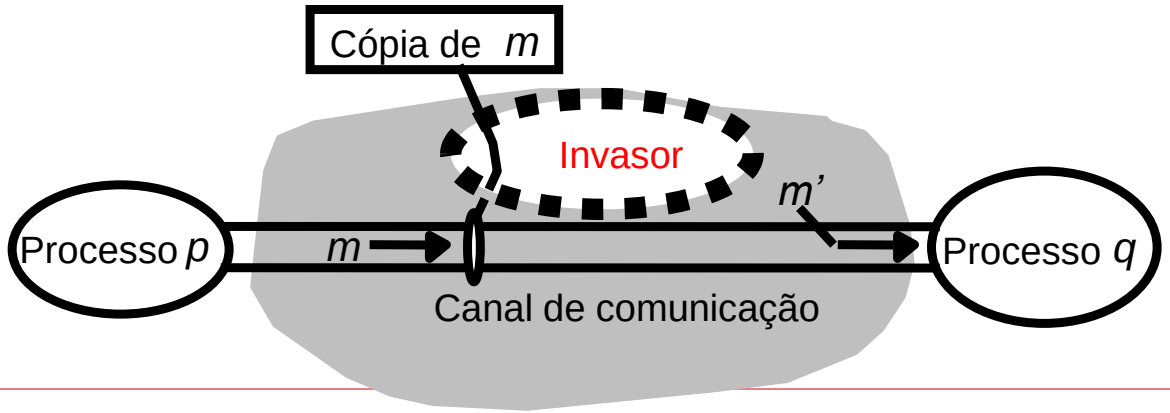
Modelo de Segurança

- A segurança de um sistema distribuído pode ser obtida tornando seguros os processos e os canais usados por suas interações e protegendo contra acesso não autorizado os objetos que encapsulam.



Modelo de Segurança

- Ameaças aos processos
  - Servidores
  - Clientes
- Ameaças aos canais de comunicação



Ameaças e formas de ataques

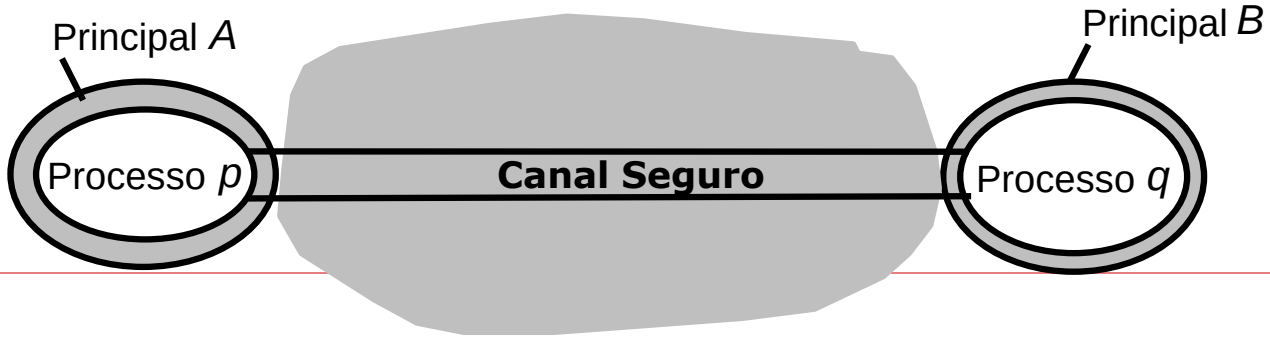
- Ameaças de segurança recaem em três classes:
  - *Leakage* (Vazamento)
    - Aquisição de informações por recipientes não autorizados;
  - *Tampering* (Falsificação)
    - Alteração não autorizada da informação;
  - *Vandalism* (Vandalismo)
    - Interferência na operação do sistema sem ganho para o infrator.

Ameaças e formas de ataques

- Classificação de ataques de acordo com o canal:
  - Intromissão
  - Mascaramento
  - Falsificação de mensagem
  - Reprodução
  - Negação de serviço

Modelo de Segurança

- Solução
  - Criptografia e segredos compartilhados
  - Autenticação
  - Canais seguros



Criptografia de Dados

- Processo para "embaralhar" os dados antes de colocá-los na rede de modo a protegê-los contra a leitura de qualquer pessoa que não seja o receptor pretendido.
- Consiste de duas partes:
  - Algoritmo de criptografia e chave de criptografia.
- As chaves consistem de dois tipos bem conhecidos:
  - Chave simétrica – chave secreta compartilhada
    - Ex.: DES (*Data Encryption Standard*).
  - Chave assimétrica
    - Ex.: Chave pública/privada.

Criptografia de Dados

- Na chave simétrica, o remetente e o destinatário possuem a mesma chave secreta.
- Na chave pública/privada: tudo que for codificado pela senha pública só é decodificado com a senha privada e vice-versa.
- Dois recursos são possíveis:
  - Preservação do caráter confidencial dos dados.
  - Mecanismo de autenticação (assinatura digital)

Chave pública-privada

Assegurar caráter confidencial dos dados (só "B" poderá decodificar):

A codifica os dados usando a senha pública de B

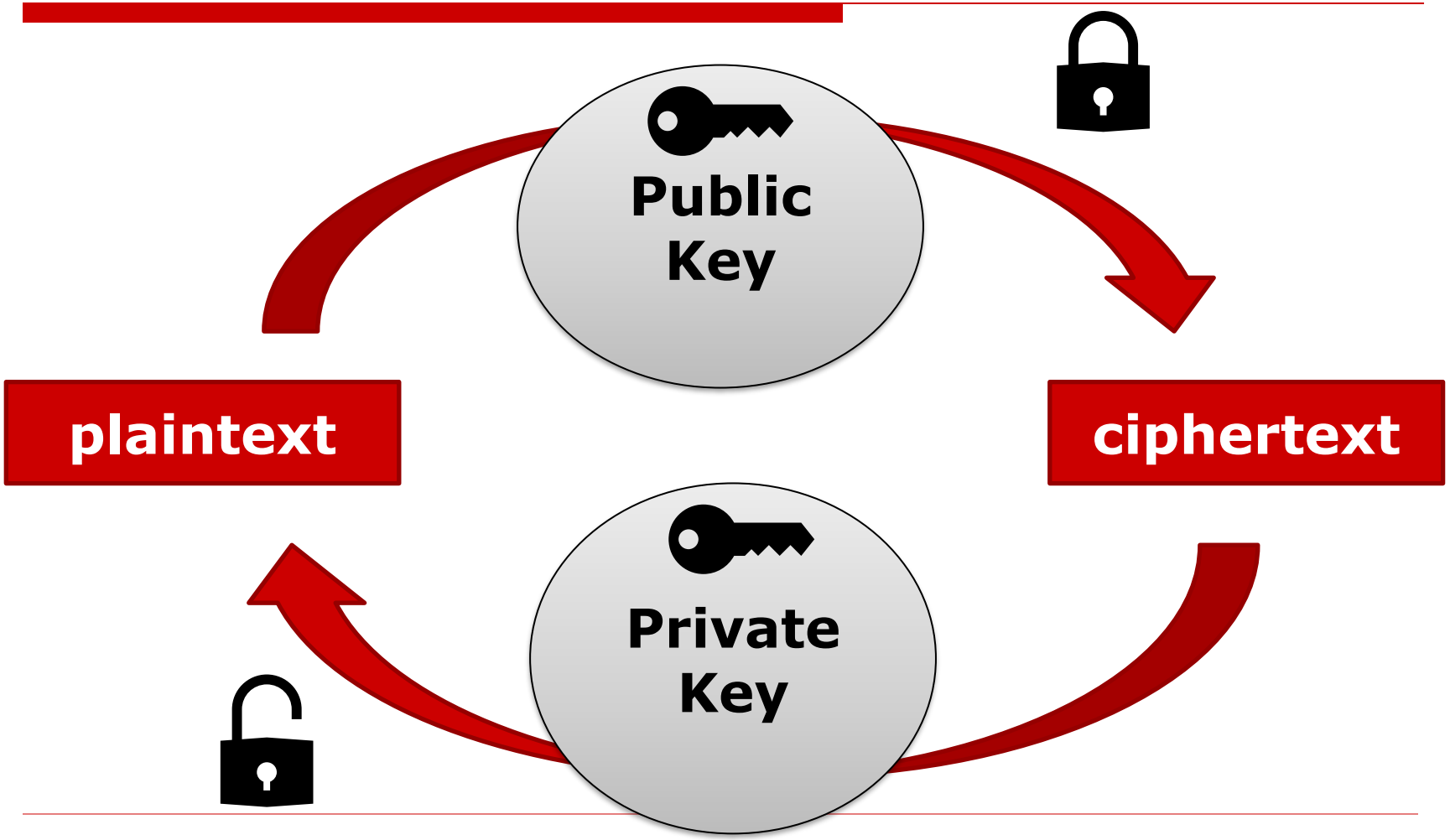
B decodifica os dados usando a sua senha privada.

Assegurar autenticação - assinatura digital (só "A" pode ter enviado os dados):

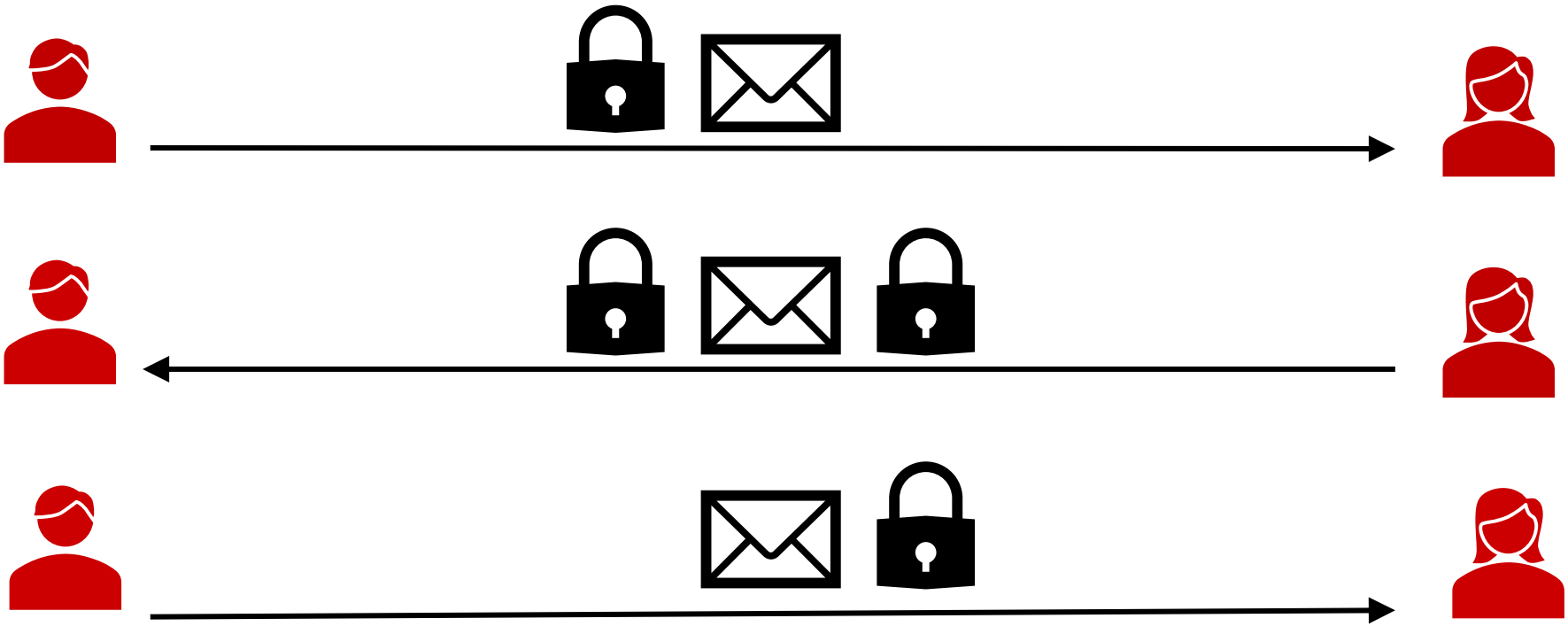
A codifica os dados usando a sua senha privada

B decodifica os dados usando a senha pública de A. (Certeza da origem)

Criptografia de chave assimétrica (RSA)



Criptografia de chave assimétrica (RSA)



Criptografia de chave assimétrica (RSA)

Chave	Uso	Conhecido por
Privado	Descriptografia	Apenas o "proprietário" do par de chaves.
Publico	Criptografia	Qualquer parte: como o nome indica, a chave pública não é um segredo e pode ser distribuída gratuitamente

Criando par de chaves RSA em Java

```java
void createRSA() throws NoSuchAlgorithmException, GeneralSecurityException, IOException {

        KeyPairGenerator kPairGen = KeyPairGenerator.getInstance("RSA");
        kPairGen.initialize(1024);
        KeyPair kPair = kPairGen.genKeyPair();
        publicKey = kPair.getPublic();
        System.out.println(publicKey);
        privateKey = kPair.getPrivate();

        KeyFactory fact = KeyFactory.getInstance("RSA");
        RSAPublicKeySpec pub = fact.getKeySpec(kPair.getPublic(), RSAPublicKeySpec.class);
        RSAPrivateKeySpec priv = fact.getKeySpec(kPair.getPrivate(), RSAPrivateKeySpec.class);
        serializeToFile("public.key", pub.getModulus(), pub.getPublicExponent());
        serializeToFile("private.key", priv.getModulus(), priv.getPrivateExponent());
}
```

RSA.java

Leitura da chave pública em Java

```java
PublicKey readPublicKeyFromFile(String fileName) throws IOException {

    FileInputStream in = new FileInputStream(fileName);
    ObjectInputStream oin = new ObjectInputStream(new BufferedInputStream(in));

    try {
        BigInteger m = (BigInteger) oin.readObject();
        BigInteger e = (BigInteger) oin.readObject();
        RSAPublicKeySpec keySpecifications = new RSAPublicKeySpec(m, e);

        KeyFactory kF = KeyFactory.getInstance("RSA");
        PublicKey pubK = kF.generatePublic(keySpecifications);
        return pubK;
    } catch (Exception e) {
        throw new RuntimeException("Some error in reading public key", e);
    } finally {
        oin.close();
    }
}
```

Client.java

Criptografa RSA em Java

```
        private byte[] encryptAESKey() {
        cipher1 = null;
        byte[] key = null;
        try {
            PublicKey pK = readPublicKeyFromFile("public.key");
            System.out.println("Encrypting the AES key using RSA Public Key" + pK);
            cipher1 = Cipher.getInstance("RSA/ECB/PKCS1Padding");

            cipher1.init(Cipher.ENCRYPT_MODE, pK);
            long time1 = System.nanoTime();
            key = cipher1.doFinal(AESkey.getEncoded());
            long time2 = System.nanoTime();
            long totalRSA = time2 - time1;
            System.out.println("Time taken by RSA Encryption (Nano Seconds) : " + totalRSA);
            i = 1;
        } catch (Exception e) {
            System.out.println("exception encoding key: " + e.getMessage());
            e.printStackTrace();
        }
        return key;
    }
```

Client.java

Descriptografa RSA em Java

```
    private void decryptAESKey(byte[] encryptedKey) {
        SecretKey key = null;
        PrivateKey privKey = null;
        keyDecipher = null;
        try {
            privKey = readPrivateKeyFromFile("private.key");
            keyDecipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
            keyDecipher.init(Cipher.DECRYPT_MODE, privKey);
            key = new SecretKeySpec(keyDecipher.doFinal(encryptedKey), "AES");
            System.out.println();
            System.out.println(" AES key after decryption : " + key);
            i = 1;
            AESKey = key;
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("exception decrypting the aes key: " + e.getMessage());
        }
    }
```

Server.java

Prática - Chat Socket com mensagens criptografadas

- 1) Execute a classe **RSA.java** para gerar os pares de chaves pública e privada:

```
Creating RSA class
Sun RSA public key, 1024 bits
  modulus: 101819040516014310406475639370634483987527172036606941878340887010672747118 9
  public exponent: 65537
Key File Created: public.key
Key File Created: private.key
```

Prática - Chat Socket com mensagens criptografadas

- 2) Execute a classe **Server.java** para abrir um socket de conexão para ouvir na porta TCP 8002:

```
run:
Receiver listening on the port 8002.Sever: Enter OUTGOING  message : >

 AES key after decryption : javax.crypto.spec.SecretKeySpec@17dee


Sever: Enter OUTGOING  message : >
```

Prática - Chat Socket com mensagens criptografadas

- 3) Execute a classe **Client.java** para enviar mensagens criptografadas para o servidor, sendo que a primeira mensagem é uma chave pública AES:

```
Genereated the AES key : javax.crypto.spec.SecretKeySpec@17dee
connection accepted localhost/127.0.0.1 :8002
Encrypting the AES key using RSA Public KeySun RSA public key, 1024 bits
  modulus: 10181904051601431040647563937063448398752717203660694187834088701067
  public exponent: 65537
Time taken by RSA Encryption (Nano Seconds) : 331600
CLIENT: Enter OUTGOING message >
CLIENT: INCOMING Message From Server   >>
CLIENT: Enter OUTGOING message >
```