CRIPTOGRAFIA DE CHAVES ASSIMÉTRICAS

Metodologias e aplicabilidade

O acordo de chaves de Diffie-Hellman-Merkle

- Um dos principais problemas no uso da criptografia simétrica para a criação de um canal de comunicação segura é a troca de chaves, ou seja, o estabelecimento de um segredo comum entre os interlocutores. Caso eles não estejam fisicamente próximos, criar uma senha secreta comum, ou substituir uma senha comprometida, pode ser um processo complicado e demorado.
- O protocolo de troca de chaves de Diffie-Hellman-Merkle (Diffie-Hellman-Merkle Key Exchange Protocol) [Schneier, 1996; Stallings, 2011] foi proposto em 1976. Ele permite estabelecer uma chave secreta comum entre duas entidades distantes, mesmo usando uma rede insegura. Um atacante que estiver observando o tráfego de rede não poderá inferir a chave secreta a partir das mensagens em trânsito capturadas. Esse protocolo é baseado em aritmética inteira modular e constitui um exemplo muito interessante e didático dos mecanismos básicos de funcionamento da criptografia assimétrica

- Considere-se um sistema com três usuários: Alice e Bob são usuários honestos que desejam se comunicar de forma confidencial; Mallory é uma usuária desonesta, que tem acesso a todas as mensagens trocadas entre Alice e Bob e tenta descobrir seus segredos (ataque de interceptação). A troca de chaves proposta por Diffie-Hellman-Merkle ocorre conforme os passos do esquema a seguir. Sejam p um número primo e g uma raiz primitiva módulo p:
- Uma raiz primitiva módulo p é um número inteiro positivo g com certas propriedades específicas em relação a p usando aritmética modular. Mais precisamente, um número g é uma raiz primitiva módulo p se todo número n coprimo de p é congruente a uma potência de g módulo p.

passo	Alice	Mallory	Bob
1	escolhe p e g	$\xrightarrow{(p,g)}$	recebe p e g
2	escolhe a secreto		escolhe b secreto
3	$A = g^a mod\ p$		$B = g^b mod\ p$
4	envia A	$\xrightarrow{A}$	recebe A
5	recebe B	$\xrightarrow{B}$	envia B
6	$k = B^a mod\ p = g^{ba} mod\ p$		$k = A^b mod\ p = g^{ab} mod\ p$
7	$m' = \{m\}_k$	$\xrightarrow{m'}$	$m = \{m'\}_k^{-1}$

- Como g^ba mod p = g^ab mod p = k, após os passos 1–6 do protocolo Alice e Bob possuem uma chave secreta comum k, que pode ser usada para cifrar e decifrar mensagens (passo 7). Durante o estabelecimento da chave secreta k, a usuária Mallory pôde observar as trocas de mensagens entre Alice e Bob e obter as seguintes informações: • O número primo p • O número gerador g • A = g ^a mod p (aqui chamado chave pública de Alice) • B = g ^b mod p (aqui chamado chave pública de Bob)

- ^ = elevado

- Para calcular a chave secreta k, Mallory precisará encontrar a na equação A = g ^a mod p ou b na equação B = g ^ b mod p. Esse cálculo é denominado problema do logaritmo discreto e não possui nenhuma solução eficiente conhecida: a solução por força bruta tem complexidade exponencial no tempo, em função do número de dígitos de p; o melhor algoritmo conhecido tem complexidade temporal subexponencial. Portanto, encontrar a ou b a partir dos dados capturados da rede por Mallory torna-se impraticável se o número primo p for muito grande. Por exemplo, caso seja usado o seguinte número primo de Mersenne3 : p = 2 127 − 1 = 170.141.183.460.469.231.731.687.303.715.884.105.727 o número de passos necessários para encontrar o logaritmo discreto seria aproximadamente de √ p = 13 × 1018, usando o melhor algoritmo conhecido. Um computador que calcule um bilhão (109) de tentativas por segundo levaria 413 anos para testar todas as possibilidades!

Considerando uma chave pública kp e sua chave privada correspondente kv, temos:

$$\begin{array}{ccc}
\{\, \{\, x \,\}_{kp} \,\}_{k}^{-1} = x & \iff & k = kv \\
\{\, \{\, x \,\}_{kv} \,\}_{k}^{-1} = x & \iff & k = kp \\
& \text{ou seja} & \\
x \xrightarrow{kp} x' \xrightarrow{kv} x & \text{e} & x \xrightarrow{kp} x' \xrightarrow{k \neq kv} y \neq x \\
x \xrightarrow{kv} x' \xrightarrow{kp} x & \text{e} & x \xrightarrow{kv} x' \xrightarrow{k \neq kp} y \neq x
\end{array}$$

Criptografia assimétrica

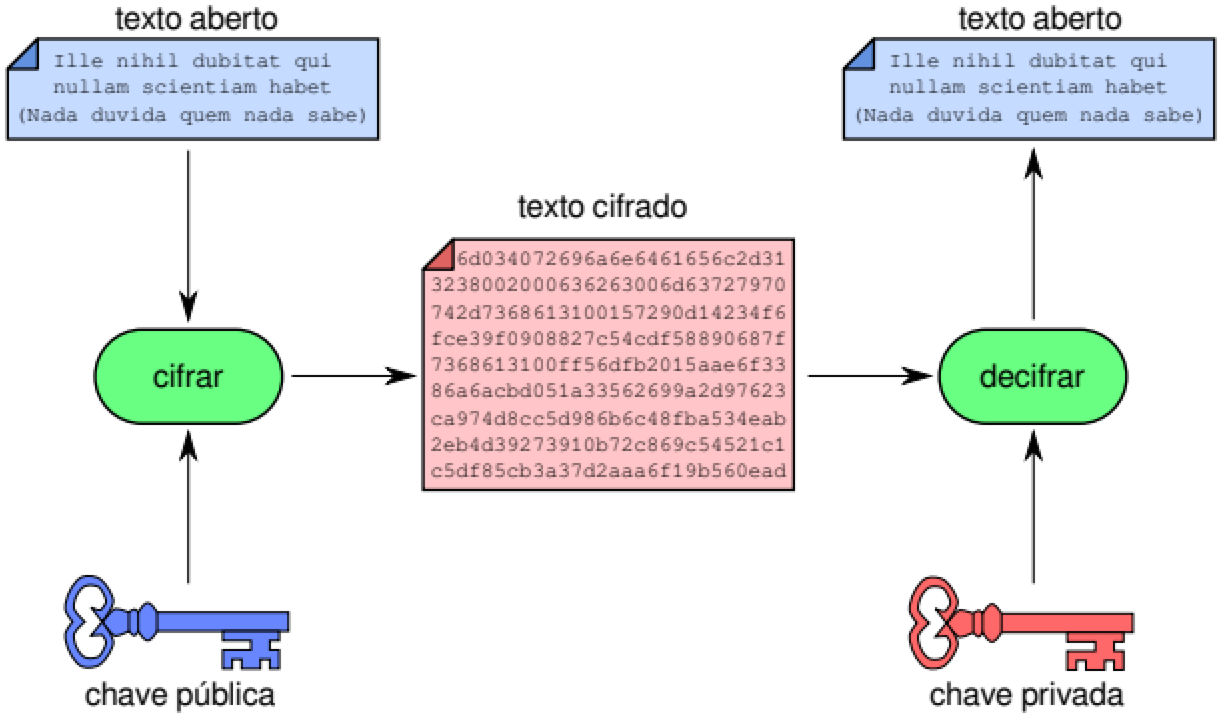

Figura 3.1: Criptografia assimétrica.

Um exemplo prático de uso da criptografia assimétrica é mostrado na Figura 3.2. Nele, a usuária Alice deseja enviar um documento cifrado ao usuário Bob. Para tal, Alice busca a chave pública de Bob previamente divulgada em um chaveiro público (que pode ser um servidor Web, por exemplo) e a usa para cifrar o documento que será enviado a Bob. Somente Bob poderá decifrar esse documento, pois só ele possui a chave privada correspondente à chave pública usada para cifrá-lo. Outros usuários poderão até ter acesso ao documento cifrado, mas não conseguirão decifrá-lo.

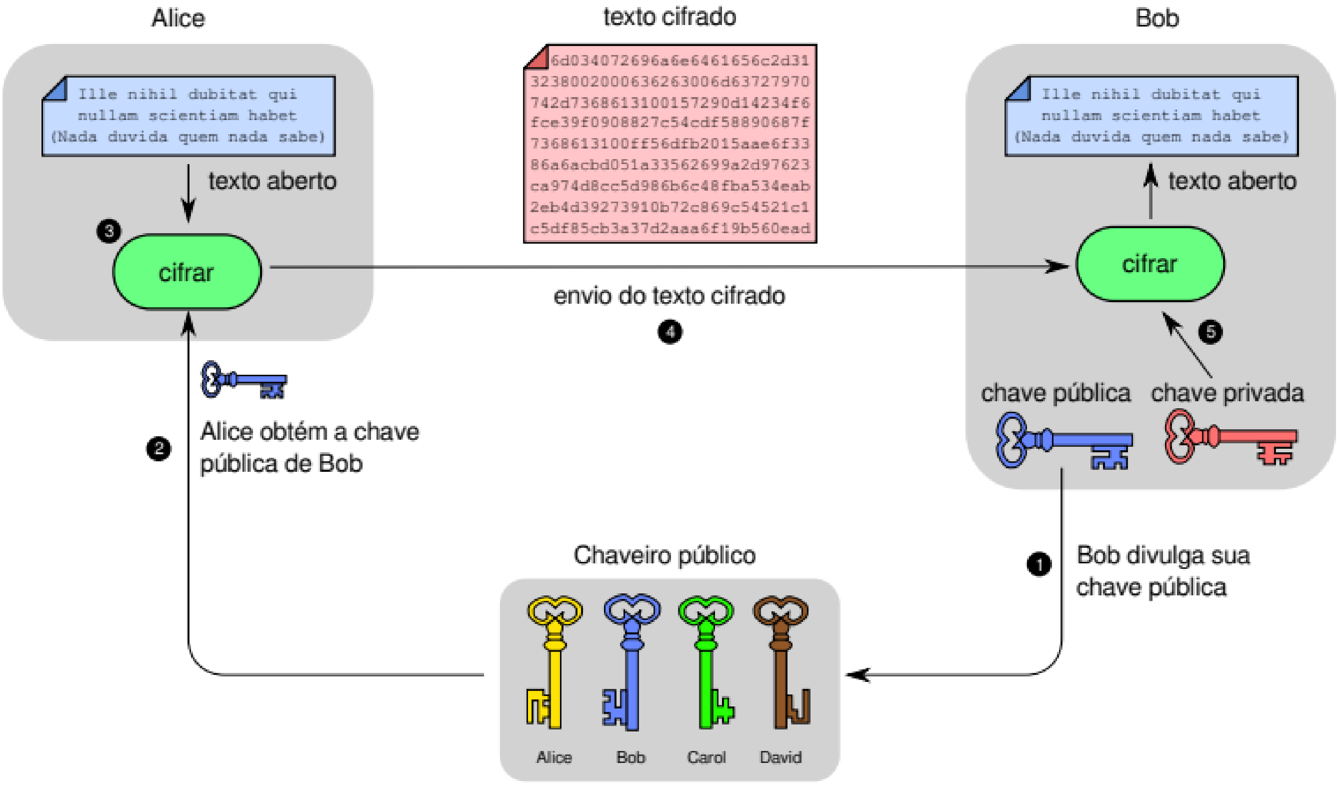

Figura 3.2: Exemplo de uso da criptografia assimétrica.

```python
import random

def gerar_chaves(p, q):
    """
    Gera um par de chaves RSA (pública e privada)

    Args:
        p (int): Número primo
        q (int): Número primo

    Returns:
        tuple: (chave_publica, chave_privada)
    """
    n = p * q
    phi_n = (p - 1) * (q - 1)
    e = random.randrange(1, phi_n)
    d = modulo_inverso(e, phi_n)
    chave_publica = (e, n)
    chave_privada = (d, n)
    return chave_publica, chave_privada
```

```python
def modulo_inverso(a, m):
    """
    Calcula o inverso modular de 'a' modulo 'm'

    Args:
        a (int): Número
        m (int): Módulo

    Returns:
        int: Inverso modular de 'a' modulo 'm'
    """
    if a < 1 or a >= m:
      raise ValueError("a deve ser maior que 0 e menor que m")
    if m <= 1:
      raise ValueError("m deve ser maior que 1")
    u, v = 1, 0
    while a > 1:
      q = a // m
      t = m
      m = a
      a = t
      u, v = v, u - q * v
    return (u + m) % m
```

```python
def criptografar(mensagem, chave_publica):
    """
    Criptografa uma mensagem usando a chave pública RSA

    Args:
        mensagem (str): Mensagem a ser criptografada
        chave_publica (tuple): Chave pública RSA (e, n)

    Returns:
        str: Mensagem criptografada
    """
    e, n = chave_publica
    texto_codificado = []
    for bloco in mensagem.split():
        bloco_numerico = int(bloco)
        criptografado = pow(bloco_numerico, e, n)
        texto_codificado.append(str(criptografado))
    return " ".join(texto_codificado)
```

```python
def descriptografar(texto_codificado, chave_privada):
  """
  Descriptografa uma mensagem usando a chave privada RSA

  Args:
      texto_codificado (str): Mensagem criptografada
      chave_privada (tuple): Chave privada RSA (d, n)

  Returns:
      str: Mensagem original
  """
  d, n = chave_privada
  texto_decodificado = []
  for bloco in texto_codificado.split():
    bloco_numerico = int(bloco)
    descriptografado = pow(bloco_numerico, d, n)
    texto_decodificado.append(str(descriptografado))
  return " ".join(texto_decodificado)

# Exemplo de uso
p = 11
q = 13
chave_publica, chave_privada = gerar_chaves(p, q)

mensagem = "Mensagem secreta para ser criptografada"
texto_codificado = criptografar(mensagem, chave_publica)
print("Mensagem criptografada:", texto_codificado)

texto_decodificado = descriptografar(texto_codificado, chave_privada)
print("Mensagem original:", texto_decodificado)
```

Referências

- M. O. Rabin. Probabilistic algorithm for testing primality. Journal of Number Theory, 12 (1):128 – 138, 1980. R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21:120–126, 1978. B. Schneier. Applied cryptography: protocols, algorithms, and source code in C, 2nd edition. Wiley, 1996. W. Stallings. Cryptography and Network Security – Principles and Practice, 4th edition. Pearson, 2011. M. Stamp. Information Security - Principles and Practice, 2nd edition. Wiley, 2011.