Segurança Informática e de Redes de Computadores

Laboratorio 2

Djocodjo Lourenço Pihale

2.a) No código **RandomKeyRSA** fornecdido o modulo é definido da seguinte forma *new BigInteger("d46f473a2d746537de2056ae3092c451", 16*). Este é um número hexadecimal, e para encontrar o tamanho da chave, precisamos converter este número em bits.

O número hexadecimal “d46f473a2d746537de2056ae3092c451” tem 32 dígitos. Cada dígito hexadecimal representa 4 bits (2^4 = 16), o tamanho da chave seria 32 \* 4 = 128 bits.

2.b) No RSA, a geração de chaves envolve os seguintes passos:

1. Escolha dois números primos grandes, p e q.
2. Calcule N = p \* q. N é usado como o módulo para ambas as chaves, pública e privada.
3. Calcule a função totiente φ(N) = (p-1)(q-1).
4. Escolha um número inteiro e tal que 1 < e < φ(N), onde e e φ(N) são coprimos. e é a chave pública.
5. Determine d como d ≡ e⁻¹ (mod φ(N)). d é a chave privada.

No código fornecido, as chaves e o valor N são definidos da seguinte maneira:

* A chave pública (e) é definida como new BigInteger("11", 16), que é 17 em decimal.
* A chave privada (d) é definida como new BigInteger("57791d5430d593164082036ad8b29fb1", 16).
* O valor N (módulo para ambas as chaves) é definido como new BigInteger("d46f473a2d746537de2056ae3092c451", 16).

Na cifra RSA:

* Para cifrar uma mensagem m, calculamos c = m^e mod N, onde c é o texto cifrado.
* Para decifrar o texto cifrado c, calculamos m = c^d mod N, onde m é a mensagem original.

No código fornecido, a cifração e decifração são realizadas usando as chaves públicas e privadas definidas acima com o valor N. A biblioteca Cipher cuida dos detalhes de implementação dessas operações.

2.c) Sim, faz sentido. No RSA, o tamanho do texto cifrado (ciphertext) é determinado pelo tamanho da chave usada para a cifração. Especificamente, o tamanho do texto cifrado será igual ao tamanho da chave pública usada para cifrar.

No código fornecido, a chave pública tem um tamanho de 128 bits (como discutido anteriormente). Portanto, o texto cifrado resultante também terá um tamanho de 128 bits.

Isso ocorre porque o RSA transforma o texto simples (plaintext) em um número inteiro e, em seguida, realiza operações matemáticas nesse número. O resultado é outro número inteiro que tem aproximadamente o mesmo número de bits que a chave. Esse número é então convertido de volta em uma sequência de bytes para criar o texto cifrado.

Portanto, independentemente do tamanho do texto simples original, o tamanho do texto cifrado será sempre aproximadamente igual ao tamanho da chave. Isso é uma característica do RSA e de muitos outros sistemas de criptografia de chave pública.

2.e) Se o bloco de texto simples (plaintext) for maior em dimensão que o valor N subjacente à operação mod e às chaves usadas, então a cifração RSA não funcionará corretamente.

Isto ocorre porque o RSA pressupõe que o texto simples é um número menor que N. Se o texto simples for maior que N, então quando elevamos esse número à potência da chave e realizamos a operação mod N, o resultado não será uma representação válida do texto simples original.

Além disso, se tentarmos cifrar um bloco de texto simples que é maior que N, a decifração resultante não será igual ao texto simples original. Isto é devido à natureza das operações matemáticas envolvidas na cifração e decifração RSA.

Portanto, antes de cifrar um bloco de dados com RSA, é importante garantir que o tamanho do bloco de dados seja menor que o tamanho da chave. Se o bloco de dados for muito grande, ele deve ser dividido em blocos menores que sejam menores que o tamanho da chave antes da cifração. Este é um dos motivos pelos quais o RSA é muitas vezes usado para cifrar chaves simétricas em vez de grandes quantidades de dados.

2.f) O trecho de código Cipher.getInstance("RSA/None/NoPadding", "BC") especifica que nenhum preenchimento é usado na criptografia RSA. Isso significa que o texto simples é criptografado diretamente, sem nenhum preenchimento adicional. **Usar "NoPadding" pode introduzir vulnerabilidades de segurança**, como a possibilidade de um ataque de texto simples escolhido ou o vazamento de informações por meio do tamanho do texto cifrado.

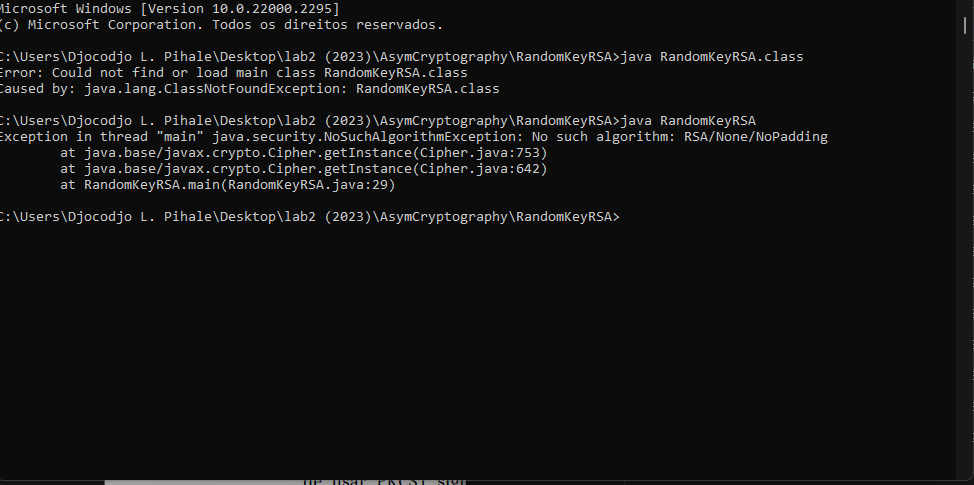
Por questões de segurança, é recomendado usar esquemas de preenchimento adequados, como o preenchimento PKCS#1 v1.5 (Cipher.getInstance("RSA/ECB/PKCS1Padding", "BC")) ou o preenchimento OAEP (Cipher.getInstance("RSA/ECB/OAEPWithSHA-256AndMGF1Padding", "BC")). Esses esquemas de preenchimento oferecem melhor segurança, adicionando aleatoriedade e garantindo que o tamanho do texto cifrado não esteja diretamente relacionado ao texto simples.

2.2

2.2.a) Neste código, ao contrário do código anterior, as chaves RSA são geradas aleatoriamente em vez de serem fornecidas manualmente. Isto é feito usando a classe KeyPairGenerator do Java, que é inicializada com um tamanho de chave de 512 bits e um objeto SecureRandom.

A principal diferença aqui é que, como as chaves são geradas aleatoriamente, este código pode ser usado para cifrar diferentes mensagens com diferentes chaves, aumentando a segurança. Como o tamanho da chave é de 512 bits, o código pode cifrar mensagens maiores.

2.2.b) Apos a execução de um dos códigos, o resultado encontrado foi



2.3.a) A implementação no trecho de código fornecido está usando o padding PKCS1 com a variante V1. Isso pode ser inferido a partir da seguinte linha de código:

Cipher cipher = Cipher.getInstance("RSA/NONE/PKCS1Padding", "BC");

O parâmetro "RSA/NONE/PKCS1Padding" passado para o método Cipher.getInstance() especifica a transformação do cifrador, que inclui o esquema de padding. Neste caso, "PKCS1Padding" indica o uso do padding PKCS1.

O comentário no código também fornece informações adicionais sobre as variantes de padding PKCS1. Ele afirma que o padding PKCS1, tipo 1 é usado, onde `Mp = 0x00 || 0x01 || F || 0x00 || M`. Aqui, `F` é uma sequência de bytes `0xFF` (mínimo de 8 bytes), e `M` é a mensagem a ser criptografada.

A razão para usar o padding PKCS1 com a variante V1 é garantir a segurança e prevenir certas vulnerabilidades. O padding é importante em algoritmos criptográficos como o RSA porque adiciona aleatoriedade e complexidade à entrada, tornando-a mais resistente a ataques.

O uso do padding PKCS1 ajuda a prevenir certos problemas de segurança, como o ataque de Bleichenbacher, que explora vulnerabilidades no processo de criptografia RSA quando o padding não é usado corretamente. O padding PKCS1 fornece uma maneira padronizada e segura de adicionar padding à entrada antes da criptografia, garantindo a segurança do algoritmo RSA.

2.4.a) O tamanho é esperado.