DCC-IME-USP

MAC 110 – Introdução à Computação
Primeiro Semestre de 2006

Ciclo Básico - profs. Flávio e Roberto

Segundo Exercício-Programa — Entrega: **16 de maio de 2006**

Algoritmo de Criptografia - MiniRSA

1 Codificação de dados

Alice e Bob são alunos do curso de introdução à computação no IME e fazem o EP2 em dupla, mas surgiu um problema: Bob precisa revisar o trabalho mas está doente em casa e sem acesso à Internet :-(. Bob então pediu para que Alice entregasse um disquete contendo o trabalho a seu colega de classe Carlos, que posteriormente levaria para Bob. Mas o que aconteceu? Carlos copiou o EP e os 3 tiraram zero...

Como fazer com que Carlos não possa ler ou copiar o trabalho contido no disquete? A solução é utilizar técnicas de codificação (ou cifra) de dados para garantir que ninguém além de Alice e Bob tenha acesso às informações contidas no arquivo. Esse processo de codificação é chamado de criptografia.

Em um esquema de criptografia, geralmente uma senha (ou chave) para encriptar/decriptar os dados é compartilhada entre ambas as partes, e se esta chave for roubada, qualquer um terá acesso à informação.

2 Criptografia de chaves assimétricas - RSA

Uma das técnicas de criptografia é o método das chaves assimétricas, o qual usa duas chaves distintas: uma *pública* (para encriptar) e a outra *privada* (para decriptar). A vantagem deste método é que a chave para **encriptar** é a pública e pode ser livremente divulgada. Somente o portador da chave privada poderá decriptar os dados encriptados com a chave pública correspondente. Mágico não?

No caso de Bob, ele pode mandar por telefone ou através de Carlos a chave pública para Alice, que irá usá-la para então encriptar os dados do projeto e gravá-los no disquete, e somente Bob (de posse da chave privada) poderá decriptar os dados e ler o trabalho. :-)

No nosso projeto, iremos usar o famoso algoritmo de criptografia RSA criado por Ron Rivest, Adi Shamir e Len Adlema (RSA) em 1978, que até hoje é muito utilizado, principalmente em conexões seguras na internet.

2.1 Gerando as chaves pública e privada

Bob precisa gerar ambas as chaves de criptografia, a *privada* e a *pública* e não podem ser escolhidas por ele, mas sim devem ser geradas através de um procedimento, que é mostrado aqui:

1. Gerar dois números primos (p e q) aleatórios e **distintos** com valor máximo de 512 cada
2. Calcular $n = p * q$
3. Calcular $m = (p - 1) * (q - 1)$
4. Calcular e, que dever ser um inteiro $1 < e < m$ tal que $mdc(e, m) = 1$
5. Calcular d, tal que $e * d \equiv 1 \ (mod\ m)$

Calculados n,d,e, as chaves são definidas da seguinte forma:

Chave pública: n, e
Chave privada: n, d

A chave pública (n,e) pode ser usada por qualquer um que queira mandar uma mensagem codificada para Bob. A chave privada (n,d) deve ser mantida em segredo, e será usada para decriptar a mensagem.

2.2 Como Encriptar

De posse da chave pública, para Alice enviar um texto para Bob ela precisa encriptar os dados caractere por caractere. Cara letra t será codificada em um inteiro c da seguinte maneira:

$c = t^e \ mod \ n$

2.3 Como Decriptar

Bob, irá decriptar os dados recebidos de Alice também caractere por caractere, e irá proceder de forma parecida com a de encriptar. Cada código recebido c será decodificado em uma letra t:

$t = c^d \ mod \ n$

2.4 Curiosidades

As chaves pública e privada são diferentes, mas estão relacionadas entre si. O interessante é que de posse da chave pública, é computacionalmente muito demorado para descobrir a privada, e isso se deve a dificuldade em fatorar o número $n = p * q$, portanto quanto maior o valor dos primos p e q, mais difícil será quebrar a criptografia. Hoje dia, usam-se valores de p e q com pelo menos 1024 bits.

2.5 Um Exemplo Simples

Exemplo simples para encriptar um número.

1. Suponha que os números primos aleatórios p e q escolhidos foram:

   ```
   p = 7
   q = 19
   ```

2. Calcular $n = 7 * 19 = 133$

3. Calcular $m = (7 - 1) * (19 - 1) = 6 * 18 = 108$

4. Escolher um número e pequeno, que seja *coprimo* de m (i.e. maior número que divida e e m):

   ```
   e = 2 => mdc(e, 108) = 2 (não)
   e = 3 => mdc(e, 108) = 3 (não)
   e = 4 => mdc(e, 108) = 4 (não)
   e = 5 => mdc(e, 108) = 1 (SIM!), então e = 5
   ```

5. Escolher um número d tal que $d * e \ mod \ m = 1$. Isso é equivalente a encontrar $d = (1 + x * m)/e$, sendo x um inteiro e a solucão também seja um valor inteiro.

   ```
   n = 1 => d = 109 / 5 = 21,8 (não)
   n = 2 => d = 217 / 5 = 43,4 (não)
   n = 3 => d = 325 / 5 = 65 (SIM!), então d = 65
   ```

(DICA: ao implementar esta parte, use o operador módulo (%) para conferir se a divisão é exata).

CHAVE PÚBLICA: n=133, e=5
CHAVE PRIVADA: n=133, d=65

De posse da chave privada $n = 133$ e $e = 5$, vamos encriptar o número 6:

```
c = t^e % n
  = 6^5 % 133
  = 7776 % 133
  = 62
```

Então, a codificacão de 6 é 62. De posse da chave privada $n = 133$ e $d = 65$, uma maneira para decriptar o 62 é a seguinte:

```
t = c^{d} % n
  = 62^{65} % 133
  = (62 % 133)*62^{64} % 133
  = (62*62 % 133)*62^{63} % 133
  = (120*62 % 133)*62^{62} % 133
  = (125*62 % 133)*62^{61} % 133
  = ...
  = 2666 % 133
  = 6
```

Esta maneira só é válida para números pequenos de d e e (que é o caso de vocês), para números maiores devemos utilizar a técnica da exponenciação (não é obrigatório implementá-la).

3 O que deverá ser feito

Implementar um sistema de criptografia RSA simplificado sem utilizar funções da biblioteca matemática do java. O sistem terá duas funcionalidades:

- Encriptar um arquivo de texto e gravar a saída codificada em inteiros.
- Decriptar o arquivo codificado e gravar a saída legível em outro arquivo.

O sistema deverá conter pelo menos 2 classes, de nomes: *CriptoRSA* e *DecriptoRSA*.

Métodos obrigatórios da classe **CriptoRSA**:

`void carregaChavePublica(long n, long e)` //n e e são as chaves geradas na classe DecriptoRSA
`void encriptaArquivo(String nomeEntrada, String nomeSaida)` //encripta um arquivo texto chamado nomeEntrada e grava a saída codificada (sequencia de inteiros separados por espaco) em um arquivo chamado nomeSaida

Métodos obrigatórios da classe **DecriptoRSA**:

`void geraChaves()` // gera as chaves pública e privada, e guarda em atributos do tipo *long* de nomes: n, e, d
`void mostraChavePublica()`
imprime chave pública `void mostraChavePrivada()`
imprime chave privada `void decriptaArquivo(String nomeEntrada, String nomeSaida)` // Decripta um arquivo nomeEntrada e grava a saida legível em nomeSaida

Sugestão: é interessante também que sejam criados os seguintes métodos para que o seu código fique bem organizado:

`long geraPrimo(long tamanhoMax);` // devolve um primo alatório menor que tamanhoMáximo
`long calculaE(long m);` // a partir de m calcula o expoente e
`long calculaD(long m, long e);` // a partir de m e e, calcula o expoente d
`long mdc(long a, long b);` // máximo divisor comum entre a e b
`long potenciaModulo(long num, long exp, long n);` // calcula num^{exp} mod n

3.1 Detalhes

Para gerar os números aleatórios use o gerador do Java:

java.util.Random gerador = new java.util.Random();

//gerador.setSeed(12345); //Semente é opcional: use para testes

int numeroAleatorio = gerador.nextInt(n) + 1; // Número aleatório entre 1 e n

Dica: Se voce chamar o método `gerador.setSeed(umNumeroQualquer)` antes das chamadas `gerador.nextInt(n)`, os número aletórios gerados seguirão sempre a mesma sequência, e isso poderá lhe ajudar na fase de testes.

3.2 Lendo e gravando arquivos

No exemplo da seção 2 vimos como encriptar um número apenas, mas no projeto vocês devem encriptar um arquivo inteiro, caractere por caractere. Para facilitar esta parte disponibilizamos um arquivo no **Paca** (*LeitorGravador.java*) que contém duas classes (Leitor e Gravador) que fazem o "trabalho sujo" de mexer com arquivos. Baixe o arquivo e compile no DrJava.

Exemplo de uso para ler um arquivo caractere a caractere, imprimir o valor lido na tela e gravá-lo em outro arquivo.

```
Leitor entrada = new Leitor();
entrada.abre("entrada.txt");

Gravador saida = new Gravador();
saida.abre("saida.txt");

while (entrada.chegouAoFim() == false) {
  char c = entrada.leChar();
  System.out.print(c+" ");
  saida.gravaChar(c);
}

entrada.fecha();
saida.fecha();
```

ATENCÃO: os caracteres codificados no método `encriptaArquivo(...)` serão números grandes, e deverão ser armazenados/lidos usando os seguintes métodos:

```
// Para gravar o long codificado no arquivo saida
saida.gravaLong(codificado);

// Para ler o proximo número inteiro (long) codificado no arquivo
long codificado = entrada.leLong();
```

Portanto, ao encriptar você deverá ler os dados do texto usando o `leChar()` e gravar a codificação usando `gravaLong(long código)`, já para decriptar é o inverso, use `leLong()` para ler o número codificado e `gravaChar()` para gravar de forma legível (modo texto).

4 Exemplo de uso do seu sistema:

```
DecriptoRSA bob = new DecriptoRSA();
bob.geraChaves(512); // gera chaves (pública e privada)
bob.mostraChavePublica(); //Imprime, por ex: Chave pública n=133 e=5

//... Bob envia Chave publica para Alice :-)

CriptoRSA alice = new CriptoRSA();
alice.carregaChavePublica(133, 5);
alice.encriptaArquivo("entrada.txt", "arquivo.rsa");

//... Alice envia arquivo "arquivo.rsa" para Bob

bob.decriptaArquivo("arquivo.rsa", "saida.txt");
```

5 Instrucões para entrega

- **O prazo de entrega é o dia 16/05/2006**
- O nome do arquivo com ambas as classes deverá ser **MiniRSA.java**
- A entrega será no **Paca** - `http://paca.ime.usp.br`, envie um arquivo zip contendo o arquivo java, as simulações feitas no Dr.Java e alguns arquivos de exemplo.
- O trabalho pode ser elaborado em dupla;
- Não poderá ser utilizado métodos da biblioteca do Math do java;
- Todas as instrucões de entrega (inclusive o cabeçalho do arquivo) são os mesmas do exercício anterior (EP1).

Boa sorte!

versão 1 - Ciclo Básico