Acordo de chaves através do método Diffie Hellman

Aluno: Lucas Pereira da Silva (10100754)

O método Diffie Hellman consiste inicialmente da geração de um número primo **p**. Também é preciso determinar uma raiz primitiva **r** do número primo **p** em questão. A determinação da raiz primitiva **r** é a parte mais custosa do método já que para realiza-lá é necessário conhecer os fatores primos de **p-1**. Devido a esse fato, o programa desenvolvido não consegue trabalhar com números com precisão de 100 dígitos decimais em tempo hábil.

Tendo o número primo **p** e uma de suas raízes primitivas **r**, cada entidade deverá gerar um número privado (**xa** e **xb**) que deve ser menor que **p**. Através de uma operação de exponenciação modular é necessário que cada entidade cacule um número compartilhado (**ya** e **yb**). A entidade **A** calcula **ya** através de $r^{xa} \bmod p$, enquanto que a entidade **B** calcula **yb** como $r^{xb} \bmod p$. As entidades devem então trocar os seus números compartilhados.

Em posse do número compartilhado da entidade alheia e em posse do seu número privado, a entidade em questão poderá calcular a chave simétrica que será usada por ambas entidades no processo de ciframento e deciframento. Para calcular a chave, basta realizar uma operação de exponenciação modular. A entidade **A** calcula a chave através de $yb^{xa} \bmod p$, enquanto que para a entidade **B** calcular a chave basta utilizar $ya^{xb} \bmod p$.

É importante mencionar que a operação de exponenciação modular, usada no cálculo do número compartilhado e no cálculo da chave, é uma operação relativamente simples de ser computada. O que garante a segurança do método é que mesmo que exista um interceptador que tenha acesso aos números compartilhados das entidades, se torna computacionalmente inviável para esse interceptador realizar a operação inversa (logaritmo discreto) e, com isso, encontrar a chave.

Código Fonte

```
package br.ufsc.inf.ine5429.diffieHellman;

import br.ufsc.inf.ine5429.millerRabin.MillerRabin;
import br.ufsc.inf.ine5429.raizPrimitiva.RaizPrimitiva;
import java.math.BigInteger;
import java.util.Random;

public class DiffieHellman {
```

```
	private BigInteger primo;
	private BigInteger raizPrimitivaDoPrimo;
	private Entidade entidadeA;
	private Entidade entidadeB;
	private Random geradorAleatorio;

	// Gera o número primo p a ser utilizado no acordo.
	public BigInteger calcularPrimo() {
		MillerRabin millerRabin = new MillerRabin(10);
		primo = millerRabin.encontrarProvavelPrimo();
		geradorAleatorio = new Random();
		return primo;
	}

	// Calcula uma raiz primitiva do número primo p gerado.
	public BigInteger calcularRaizPrimitivaDoPrimo() {
		RaizPrimitiva raizPrimitiva = new RaizPrimitiva(primo);
		raizPrimitivaDoPrimo =
raizPrimitiva.encontrarRaizPrimitivaDePrimo();
		return raizPrimitivaDoPrimo;
	}

	// Realiza o acordo de chaves.
	public void iniciarAcordoDeChaves() {
		entidadeA = new Entidade();
		entidadeB = new Entidade();
		BigInteger xa = entidadeA.gerarNumeroPrivado();
		BigInteger xb = entidadeB.gerarNumeroPrivado();
		BigInteger ya = entidadeA.calcularNumeroCompartilhado();
		BigInteger yb = entidadeB.calcularNumeroCompartilhado();
		entidadeA.receberNumeroCompartilhadoDaOutraEntidade(yb);
		entidadeB.receberNumeroCompartilhadoDaOutraEntidade(ya);
		BigInteger ka = entidadeA.calcularChave();
		BigInteger kb = entidadeB.calcularChave();
		System.out.printf("Acordo de chaves iniciado.\n");
		System.out.printf("xa gerado: %d.\n", xa);
		System.out.printf("xb gerado: %d.\n", xb);
		System.out.printf("ya calculado: %d.\n", ya);
		System.out.printf("yb calculado: %d.\n", yb);
```

```
            System.out.printf("ka calculada: %d.\n", ka);
            System.out.printf("ka calculada: %d.\n", kb);
            System.out.printf("Acordo de chaves finalizado.\n");
      }

      // Representa uma entidade participante do acordo de chaves.
      private class Entidade {
            private BigInteger numeroPrivado;
            private BigInteger numeroCompartilhado;
            private BigInteger numeroCompartilhadoDaOutraEntidade;
            private BigInteger chave;

            // Gera o seu número privado que será utilizado para calcular a
chave.
            public BigInteger gerarNumeroPrivado() {
                  do {
                        numeroPrivado = new BigInteger(primo.bitCount(),
geradorAleatorio);
                  } while (numeroPrivado.compareTo(primo) >= 0);
                  return numeroPrivado;
            }

            // Calcula o número compartilhado a ser usado pela outra entidade.
            public BigInteger calcularNumeroCompartilhado() {
                  numeroCompartilhado =
raizPrimitivaDoPrimo.modPow(numeroPrivado, primo);
                  return numeroCompartilhado;
            }

            // Recebe o número compartilhado da outra entidade.
            public void receberNumeroCompartilhadoDaOutraEntidade(BigInteger
numeroCompartilhadoDaOutraEntidade) {
                  this.numeroCompartilhadoDaOutraEntidade =
numeroCompartilhadoDaOutraEntidade;
            }

            // Calcula a chave através do número compartilhado da outra entidade
e do número privado.
            public BigInteger calcularChave() {
                  chave =
numeroCompartilhadoDaOutraEntidade.modPow(numeroPrivado, primo);
```

```
                return chave;
            }
        }

        public static void main(String[] argumentos) {
            DiffieHellman diffieHellman = new DiffieHellman();
            System.out.printf("Número primo: %d.\n",
diffieHellman.calcularPrimo());
            System.out.printf("Raiz primitiva do número primo: %d.\n",
diffieHellman.calcularRaizPrimitivaDoPrimo());
            diffieHellman.iniciarAcordoDeChaves();
        }
}
```