



## Segurança de Sistemas Informáticos

### Guia para Aula Laboratorial 6

2º Ciclo em Engenharia Informática

2º Ciclo em Eng. Eletrotécnica e de Computadores

2º Ciclo em Matemática e Aplicações

## Computer Systems Security

### Guide for Laboratory Class 6

M.Sc. in Computer Science and Engineering

M.Sc. in Electrical and Computer Engineering

M.Sc. in Mathematics and Applications

### Sumário

Implementação do terno de algoritmos que concretizam a cifra ElGamal na linguagem de programação Java, como forma de abordar e estudar a *Java Cryptography Architecture* (JCA).

### Summary

*Implementation of the three algorithms that define the ElGamal cipher in Java programming language, as an excuse to study the Java Cryptography Architecture (JCA).*

### Pré-requisitos:

Este guia laboratorial pressupõe que o(a) seu(ua) executante tem conhecimentos na linguagem de programação orientada a objetos Java e os meios necessários para compilar e executar aplicações desenvolvidas nessa linguagem (um compilador de linha de comandos é suficiente para trabalhar todos os exercícios deste guia).

## 1 Preliminares

### Preliminaries

A ideia da tarefa principal desta secção é implementar o terno de algoritmos que definem a cifra ElGamal na linguagem Java. Antes de avançar, convém verificar se consegue compilar e correr programas em Java no seu ambiente de trabalho.

### Tarefa 1 Task 1

Numa máquina com sistema operativo Linux (considera-se que estamos a usar o sistema operativo Fedora), abra um terminal e introduza os seguintes comandos:

```
> java
> javac
```

Caso o sistema avise que um ou ambos os comandos não estão disponíveis, precisa instalar o Java.

**Q1.: Para que serve o comando `javac`?**

- ☒ Serve para compilar código Java.
- ☐ Serve para correr um programa escrito em Java.
- ☐ Serve para mostrar no terminal a ilha principal da Indonésia.

Se respondeu corretamente à questão anterior, e por exclusão de partes, já deve saber para que serve o comando `java`. Caso o sistema se tenha queixado da falta do primeiro comando, convém instalar uma Máquina virtual Java

J \_\_\_\_\_

V \_\_\_\_\_

M \_\_\_\_\_

A instalação da JVM pode ser conseguida através de um comando semelhante ao seguinte, numa *shell* com privilégios de administração:

```
> yum install java-1.8.0-openjdk
```

Já a instalação do compilador Java (`javac`) pode ser conseguida através de um comando semelhante ao que se segue, também pressupondo acesso a uma *shell* com privilégios de administração:

```
> yum install java-1.8.0-openjdk-devel
```

### Tarefa 2 Task 2

A cifra ElGamal elabora no protocolo de acordo de chaves Diffie-Hellman. Por isso, fica aqui a sugestão da sua implementação à priori ou posteriori das tarefas seguintes, de modo a colmatar possíveis dú-

vidas no funcionamento do protocolo e a facilitar o entendimento do que é feito a seguir.

### Tarefa 3 Task 3

Havendo instalado o compilador e plataforma de execução de programas em Java, a tarefa passa agora pela implementação, compilação e execução de um programa simples. Para isso, sugere-se que crie uma pasta chamada Alice (`> mkdir Alice`) contendo o ficheiro AliceGen.java (`> touch AliceGen.java`) que, por sua vez, inclui o código que se mostra a seguir:

```
import java.io.*;
import java.math.BigInteger;
import java.security.*;

public class AliceGen {
    static public void main(String[] args)
        throws NoSuchAlgorithmException {

        BigInteger oBIP = new BigInteger(
            "9949409665013933710618693397761851"+
            "3974146274831566768179581759037259"+
            "7887981514998146539514927243654713"+
            "1625365146334225578531174860292245"+
            "8795201382445323499931625451272600"+
            "1731801361232454412041335158004959"+
            "1724201186355872172330366152337257"+
            "2477211620144038809673692512025566"+
            "6737469935933846006670473736922035"+
            "83");

        BigInteger oBlg = new BigInteger(
            "4415740483796032876887268067768680"+
            "2650999163226766694797650810379076"+
            "4164631472654010844911136676240545"+
            "5733539476160487688244692492984068"+
            "1990106974314935015501571333024773"+
            "1724403524753587506682134446073538"+
            "7275465080503191286669211981937704"+
            "1901642732455911509867728218394542"+
            "7453300140710403268568469901197196"+
            "75");

        System.out.println("--- Public key ---");
        System.out.println("--- P ---");
        System.out.println(oBIP.toString());
        System.out.println("--- g ---");
        System.out.println(oBlg.toString());
    }
}
```

**Nota:** por comodidade, o código e ficheiro mencionados antes também foram disponibilizados com este guia laboratorial.

**Q2.: Conseguiu correr o código Java listado anteriormente?**

☐ Foi canja de galinha...

☐ Sinto a falta de um ambiente de desenvolvimento integrado, e não sei compilar ou executar programas Java a partir da linha de comandos...

**Q3.: Quantos bits têm os números com que está a lidar?**

☒ Uii! Muitos bits!

☐ 128 bits   ☐ 256 bits   ☐ 512 bits   ☐ 1024 bits

## 2 Cifra ElGamal em Java

### ElGamal Cipher in Java

A definição da cifra ElGamal foi feita na aula 2 desta unidade curricular. Para facilitar a sua implementação e análise, foi transcrito para aqui.

Seja  $J$  um grupo ciclico finito de ordem  $n$ . Considere-se também uma cifra de chave simétrica autenticada representada pelos algoritmos  $(E, D)$  definida sobre  $(\mathcal{K}, \mathcal{M}, \mathcal{C})$  e uma função de *hash* criptográfica  $H : J \times J \rightarrow K$ . O sistema pode então ser definido da seguinte forma:

- O **gerador** de chaves públicas e privadas  $G$  especifica-se da seguinte forma:

1. Escolhe um gerador para  $J$ , um número aleatório  $x$  em  $J$  e calcula  $X = g^x \bmod p$ ;
2. Devolve  $sk = x$  e  $pk = (g, X)$ .

- O **algoritmo de cifra**  $E(pk, m)$  atua da seguinte forma:

1. Escolhe um número aleatório  $y \in J$  e calcula  $Y = g^y \bmod p$  e  $k = X^y \bmod p$ ;
2. Calcula  $k_2 = H(X, k)$  e cifra a mensagem com esta chave, i.e.,  $c \leftarrow E(k_2, m)$ ;
3. Devolve  $(Y, c)$ .

- O **algoritmo de decifra**  $D(sk, (Y, c))$  atua da seguinte forma:

1. Calcula  $k = Y^x \bmod p$ ;
2. Calcula  $k_2 = H(X, k)$  e decifra o criptograma com esta chave, i.e.,  $m \leftarrow D(k_2, c)$ ;
3. Devolve  $m$ .

### Tarefa 4 Task 4

Como se pode concluir da análise do sistema criptográfico, a ElGamal (tal como outros sistemas criptográficos de chave pública) é constituída por 3 algoritmos: um para gerar um par de chaves, outro para

cifrar e outro para decifrar. A primeira tarefa consiste na implementação no algoritmo de geração de chaves públicas e privadas da ElGamal. **Note** que, no código incluído na secção anterior, já lhe foram facultados valores para o  $P$  e para o  $g$ , pelo que não os precisa de os gerar novamente mas, caso precisasse, podia usar o seguinte comando:

```
> openssl dhparam -text -dsaparam 1024.
```

A parte `dhparam` do `openssl` permite a geração e manipulação de valores Diffie-Hellman sobre grupos  $\mathbb{Z}_P^*$ . Sem a opção `dsaparam`, apenas é gerado o primo de 1024 (os números 2 ou 5 são sempre usados como geradores, neste caso). Com esta opção, obriga-se a gerar também o gerador, conforme requisito do *Digital Signature Algorithm* (DSA).

O programa para gerar chaves ElGamal deve chamar-se `AliceGen` e escrever, no fim ou ao longo da sua execução e no ecrã, a chave privada  $sk$  e a chave pública  $pk$ . Deve escrever os respetivos valores no ecrã no formato que já foi sugerido no exemplo de código incluído em cima.

Repare que a geração de uma chave pública e privada ElGamal presume que se gere um número aleatório  $X$  entre 1 e  $P$ . A forma mais segura de fazer isso é usar um bom gerador de sequências pseudo-aleatórias (e.g., um baseado numa função de *hash* criptográfica) ou uma fonte de verdadeira aleatoriedade (e.g., o `/dev/random`). Esta última possibilidade é a preferida. Contudo, para esta aula, sugere-se o uso da classe `SecureRandom`:

```
SecureRandom prng = SecureRandom.getInstance("SHA1PRNG");
```

A geração de um `BigInteger` pode ser conseguida através de uma instrução parecida com a seguinte:

```
BigInteger oBlsk = new BigInteger(prng.generateSeed(iNumBytes));
```

em que `iNumBytes` é o número de bytes que o número que queremos gerar deve efetivamente ter. Note que deve garantir que o número gerado é positivo e menor que  $P$ .

**Q4.: Qual o papel do número gerado entre 1 e  $P$ ?**

- ☐ Este número é a chave privada.
- ☐ Este número é a chave pública.
- ☐ Este número será usado para calcular a chave pública e privada.
- ☐ Este número não tem efeito no contexto do sistema criptográfico, para pode ser usado para obter uma chave para o totoloto.

**Q5.: Como se cria a chave pública?**

- ☐ A chave pública é constituída pelo  $g$  e pelo  $P$ .
- ☐ Calcula-se  $pk = sk^g \bmod P$ .
- ☐ Calcula-se  $pk = g^{sk} \bmod P$ .
- ☐ Calcula-se  $pk = sk + g \bmod P$ .

## Tarefa 5 Task 5

O próximo programa a implementar é o de cifra, que se deve chamar `BobEncrypt`. Este programa deve aceitar, como parâmetros de entrada via linha de comandos, o nome do ficheiro a cifrar, o valor de  $P$ , o valor de  $g$  e uma das chaves. Estes parâmetros devem ser, respetivamente, o `args[0]`, `args[1]`, `args[2]` e `args[3]`.

**Q6.: O programa deve aceitar a chave pública ou privada?**

- ☐ A chave pública da Alice.
- ☐ A chave privada da Alice.
- ☐ A chave pública do Bob.
- ☐ A chave privada do Bob.
- ☐ A chave privada da T'Maria!

Note que o algoritmo de cifra requer também que se gere um número aleatório, se calcule um valor de *hash*, uma exponenciação módulo  $P$  e uma cifra com chave simétrica. Para este exercício sugere-se que use a função de *hash* SHA256 e a cifra AES128 no modo CBC. Mais concretamente, o seguinte trecho de código deve estar algures no seu programa:

```
MessageDigest oMD = MessageDigest.
    getInstance("SHA-256");
byte keyHash[] = new byte[32];
keyHash = oMD.digest(oBlkey.toByteArray());
byte bKey[] = new byte[16];
byte bIV[] = new byte[16];

for( int i = 0; i < 16 ; i++ ){
    bKey[i] = keyHash[i];
    bIV[i] = keyHash[i+16];
}

Cipher cipher = Cipher.getInstance("AES/CBC/
    PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, new
    SecretKeySpec(bKey, "AES"), new
    IvParameterSpec(bIV));
```

**Q7.: Consegue detetar alguma discrepância entre a definição do sistema criptográfico e a implementação constante no trecho anterior?**

- ☐ Olhos de águia.
- ☐ Não... não consigo.

O programa deve ler o ficheiro a cifrar e escrever o resultado da cifra no ficheiro `ciphertext.egm`. Deve simultaneamente escrever no ecrã (embora esta escolha não seja a escolha recomendada) o valor da chave efémera criada pelo Bob, denotada por  $Y$  anteriormente, no formato já discutido aquando da geração de chaves. Idealmente,  $Y$ ,  $g$  e  $P$  deveriam ser escritos no início da mensagem (ficheiro) cifrada.

**Q8.: Quando usa esta cifra, qual o tamanho do criptograma em relação à mensagem original?**

- ☐ O criptograma tem o mesmo tamanho da mensagem.
- ☐ É o triplo ou próximo do triplo.
- ☐ É o dobro ou próximo do dobro.
- ☐ O tamanho do criptograma é sempre maior que a mensagem.
- ☐ O tamanho do criptograma é sempre menor que a mensagem.

Depois de implementar o algoritmo de cifra e de compilar o resultado para um programa, experimente-o, cifrando várias vezes o mesmo ficheiro. **Q9.: O ficheiro cifrado assim obtido é sempre igual?**

- ☐ Sim, é sempre igual.
- ☐ Não, é sempre diferente.

**Q10.: Por que é que esta cifra é dita probabilística?**

☐ \_\_\_\_\_

☐ Opção inválida (assinale e preencha corretamente a opção anterior).

## Tarefa 6 *Task 6*

A última tarefa consiste na implementação do algoritmo de decifra do sistema criptográfico ElGamal. Dada a implementação do algoritmo de cifra, esta implementação deve avizinhar-se mais simples. Este algoritmo deve aceitar o nome do ficheiro a decifrar, os valores decimais de  $P$ , da chave privada  $sk$  e da chave efémera denotada por  $Y$ , por esta ordem. Na *string* de argumentos, estes devem ser, respetivamente, `args[0]`, `args[1]`, `args[2]` e `args[3]`. O ficheiro e classe Java que implementam o algoritmo de decifra devem chamar-se `AliceDecrypt.java` e `AliceDecrypt`, respetivamente.