Universidade da Beira Interior
Faculdade de Engenharia
Departamento de Informática

Segurança de Sistemas Informáticos
Guia para Aula Laboratorial 7
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 7
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 o sistema criptográfico ElGamal em curvas elípticas na linguagem de programação Python, como forma de encapsular o estudo desta abordagem para construir mecanismos criptográficos de chave pública e enfatizar o nível de abstração fornecido pela linguagem de programação utilizada.

Summary
Implementation of the three algorithms that define the Elliptic Curve ElGamal cryptosystem in the Python programming language, as an excuse to study this approach to build public key cryptosystems and emphasize the abstraction level provided by this specific programming language.

Pré-requisitos:
Este guia laboratorial pressupõe que o(a) seu(ua) executante tem alguns conhecimentos na linguagem de programação Python e os meios necessários para interpretar aplicações desenvolvidas nessa linguagem (não é necessário, para este guia, qualquer ambiente de desenvolvimento integrado).

1 Preliminares
Preliminaries

Este guia laboratorial tem dois objetivos principais. O primeiro é o de facilitar a compreensão das operações envolvidas na criptografia em curvas elípticas. O segundo é o de implementar a ElGamal em Python. Antes de avançar, convém verificar se consegue interpretar (e correr) programas escritos em Python 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:
`> python2`
e
`> python3`

Caso esteja instalado, o sistema deve apresentar-lhe uma *shell* para introdução de instruções em Python. Caso contrário, pode ser necessária a instalação do suporte à linguagem na sua máquina, que normalmente é conseguida através de uma instrução semelhante a
`> sudo dnf install python3`

Tarefa 2 *Task 2*

Construa um programa simples que imprima no ecrã a cadeia de caracteres `Olá` seguida do seu nome, que deve ser passado como argumento ao mesmo na sua invocação. Comece por criar uma diretoria onde vai guardar o que resultar deste trabalho com `> mkdir Lab7`, seguido de `> cd Lab7` e `> nano hello.py`.

Por conveniência e de modo a agilizar o desenvolvimento deste guia, é colocado um exemplo para o conteúdo deste programa em baixo. Ainda assim, antes de prosseguir, considere responder às seguintes questões. **Q1.: A linguagem Python é interpretada ou compilada?**
☒ Interpretada. ☐ Compilada.
☐ Maltratada. ☐ Incompreendida.
☐ Todas as linguagens podem ser compiladas ou interpretadas. O facto de ser usada, de forma mais ou

menos típica, de uma forma ou de outra, não deve ser considerada uma característica da própria linguagem.

Q2.: O Python usa chavetas `{}` para definição de blocos de instruções?
☐ Obviamente. E é possível fazer isso de outra forma qualquer?
☒ Não, não usa.

Q3.: De que forma é que são definidos os blocos de código em Python?
☒ Através da indentação.
☐ Através de parêntesis retos.
☐ Usando *tags* delimitadoras, e.g., `IF ... ENDIF`.

Q4.: Existem *ponto e vírgulas* (`;`) a terminar instruções em Python?
☒ Nem por isso.
☐ Sim, tal como no C e no Java.

Q5.: Como se definem funções em Python?
☒ Usando a *keyword* `def`, seguida do nome da função, possíveis parâmetros entre parêntesis, e de *dois pontos* (`:`).
☐ Simplesmente colocando o nome da função e escrevendo a sua definição a partir da linha de baixo, devidamente identada.
☐ Com uma sintaxe igual à do C.
☐ Com uma sintaxe igual à do Java.

Q6.: Quando coloca `> python3 hello.py` no terminal e carrega no `enter`, quais são as instruções que são imediatamente executadas?
☐ Todas as instruções dentro do ficheiro.
☐ Todas as instruções que definem funções.
☒ Todas as instruções que estão no nível 0.
☐ Todas as instruções que estão no nível 0 do *League of Legends* (LOL).

Experimente colocar o código seguinte no ficheiro `hello.py` e executá-lo corretamente. Não avance antes de ter a certeza de que esta parte está terminada com sucesso.

```
import sys

def main():
    print ('Hello ' + sys.argv[1] + '.\n')

if __name__ == "__main__":
    main()
```

2 Criptografia em Curvas Elípticas

Elliptic Curve Cryptography

Tarefa 3 *Task 3*

Juntamente com este guia laboratorial foi disponibilizado um ficheiro com uma implementação incompleta em python de algumas operações criptográficas em curvas elípticas, bem com a definição de uma curva com 192 bits normalizada pelo NIST na secção *2.5.2 Recommended Parameters secp192r1* do documento *SEC 2: Recommended Elliptic Curve Domain Parameters*, disponível em `http://www.secg.org/SEC2-Ver-1.0.pdf`. O programa contém ainda um conjunto de testes definidos na função `main()`, que podem ser usados para verificar se a implementação das funções foi bem feita ou não. Existem valores (de teste) para fazer estas verificações em `http://point-at-infinity.org/ecc/nisttv`.

Esta tarefa consiste em completar a implementação das funções `sum(self,p2)` e `multiplyPointByScalar(self, n)`. Para isso, pode consultar a aula 4 desta unidade curricular, e as fórmulas e algoritmo transcritos para aqui por conveniência.

Adição e Multiplicação por 2 em Curvas Elípticas em $\mathbb{Z}_p$
Considere que $A = (x_1, y_1)$ e $B = (x_2, y_2)$. As coordenadas de $A + B = (x_3, y_3)$ são dadas respetivamente por
$x_3 = s^2 - x_1 - x_2 \bmod p$ e
$y_3 = s(x_1 - x_3) - y_1 \bmod p$
em que

$$s = \begin{cases} (y_2 - y_1) \times (x_2 - x_1)^{-1} \bmod p, & \text{se } A \neq B \\ (3x_1^2 + a) \times (2y_1)^{-1} \bmod p, & \text{se } A = B \end{cases}$$

Corra várias vezes o programa após a implementação das funções, e certifique-se de que os valores coincidem com os de teste fornecidos pelo NIST em `http://point-at-infinity.org/ecc/nisttv`.

Tarefa 4 *Task 4*

A cifra ElGamal em curvas elípticas elabora, de novo, no protocolo que acordo de chaves Diffie-Hellman, mas sobre estas estruturas algébricas. A definição da cifra ElGamal foi feita na aula 4 desta

Algorithm 1: Multiplicação modular rápida.

input : Uma curva elíptica E, um ponto dessa curva A e número $d \in \mathbb{N}$, cuja representação binária é dada por $d = \sum_{i=0}^{t} d_i 2^i$, $d_i \in \{0,1\}$ e $d_t = 1$.
output: $T = d \times A$.
begin
  T $\longleftarrow A$;
  foreach *bit* d_i *desde* d_{t-1} *até* d_0 **do**
    $T = T + T \bmod p$;
    if *bit* $d_i = 1$ **then**
      $T = T + A \bmod p$;
  return *T*

unidade curricular. Para facilitar a sua implementação e análise, essa definição foi transcrita para aqui.

Seja $\mathcal{E}$ uma curva elíptica com parâmetros considerados seguros (p, a, b, G, n, h) (G é o ponto base). 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 $H : J \times J \to K$. O sistema pode então ser definido da seguinte forma:

- O **gerador** de chaves públicas e privadas F especifica-se da seguinte forma:
  1. Escolhe um número aleatório d_A em $\{1, ..., n-1\}$ e calcula $X = (x_1, y_1) = d_A * G$;
  2. Devolve $sk = d_a$ e $pk = (X, (p, a, b, G, n, h))$.
- O **algoritmo de cifra** $E(pk, m)$ atua da seguinte forma:
  1. Escolhe um número aleatório d_B em $\{1, ..., n-1\}$ e calcula $Y = (x_2, y_2) = d_B * G$;
  2. Calcula também $K = (x_3, y_3) = d_B * X$ – o segredo S entre os dois passa a ser x_3;
  3. Calcula $k_2 = H(X, x_3)$ e cifra a mensagem com esta chave, i.e., $c \leftarrow E(k_2, m)$;
  4. Devolve (Y, c).
- O **algoritmo de decifra** $D(sk, (Y, c))$ atua da seguinte forma:
  1. Deriva o ponto comum $K = (x_3, y_3) = d_B * X$ – o segredo S entre os dois passa a ser x_3;
  2. Calcula $k_2 = H(X, x_3)$ e decifra o criptograma com esta chave, i.e., $m \leftarrow D(k_2, c)$;
  3. Devolve m.

A primeira tarefa consiste, portanto, em implementar o primeiro algoritmo que compõe o conjunto de três algoritmos do sistema criptográfico: aquele que gera as chaves públicas e privadas. O programa deve gerar o número aleatório com 192 bits, calcular a chave pública e imprimir tudo em hexadecimal, conforme mostra o *output* seguinte:

```
> python2 gen-ecc.py


--- BEGIN PRIVATE KEY ---
0x10d5a87c26d4eb7e11cfd98bb7c5e481bccd1ef3f278a88dL

--- BEGIN PUBLIC KEY ---
pk.x = 0x76b228770243918b544a1d3b0acf7e8fcab5d639258fa437L
pk.y = 0xdee6b8c2e58f4e71ffaa7b9b9d59a10673a341e545da664eL

G.x  = 0x188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012L
G.y  = 0x7192b95ffc8da78631011ed6b24cdd573f977a11e794811L
```

Nota: pode utilizar como ponto base aquele definido pelas seguintes coordenadas:

```
x = 188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012
y = 7192b95ffc8da78631011ed6b24cdd573f977a11e794811
```

Q7.: Dado o número de linhas de código necessárias para implementar o procedimento pedido, acha que o Python facilita ou dificulta a tarefa do programador?
☒ Chiça! Facilita fortemente.
☐ Dificulta... :_(

Nota: para efeitos desta aula, pode usar o gerador disponibilizado pelo Python (`import random`). Contudo, convém sempre usar um gerador que se saiba ser de qualidade criptográfica, nomeadamente o que é fornecido no módulo `Crypto`.

```
from Crypto import Random
rnd = Random.new()
rnd.read(24)
```

Tarefa 5 *Task 5*

A próxima tarefa consiste em implementar o algoritmo de cifra do sistema criptográfico ElGamal em curvas elípticas. Note que serão necessários os seguintes passos:

1. Gerar um número aleatório e calcular a multiplicação de um ponto por um escalar (parecido ao que já feito anteriormente);
2. Calcular a chave partilhada através da multiplicação de um ponto por um escalar:

3. Calcular um valor de *hash*;
4. Cifrar com uma cifra de chave simétrica, e.g., a AES.

Note que, em Python, e se tiver o conjunto de módulos criptográficos `pycrypto` instalados, o cálculo de valores de *hash* pode ser simplesmente conseguido através de um *snippet* semelhante a:

```
from Crypto.Hash import SHA256
hash = SHA256.new()
hash.update('message')
hash.digest()
```

A cifra (a decifra é parecida) pode ser conseguida através de instruções parecidas com:

```
from Crypto.Cipher import AES
aescipher = AES.new('key', AES.MODE_CBC, 'IV')
msg = "mensagem a ser cifrada"
ctxt = aescipher.encrypt(msg)
```

Caso o sistema operativo não tenha o conjunto de módulos necessários disponíveis, considere instalá-los com uma sequência de comandos semelhante a:

```
> su (palavra-passe root)
> yum install python-crypto
```

Este programa deve aceitar o nome do ficheiro a cifrar e os valores hexadecimais das coordenadas do ponto G e da chave pública Y, por esta ordem. Na *string* de argumentos, estes *inputs* devem ser, respetivamente, `argv[1]`, `argv[2]` e `argv[3]`, `argv[4]` e `argv[5]`. O ficheiro com código fonte Python que implementam o algoritmo de decifra devem chamar-se `bob-encrypt.py`.

O *output* produzido deve ser algo semelhante ao que é apresentado a seguir:

```
Ciphertext written in ciphertext.aes.

--- BEGIN EPHEMERAL KEY ---
G.x  = 0x188da80eb03090f67cbf20eb43a18800f4ff0afd82ff1012L
G.y  = 0x7192b95ffc8da78631011ed6b24cdd573f977a11e794811L

Y.x = 0x969f7a8dfa269661c9b99b6d4b07b45ab67a575c99de77aeL
Y.y = 0x6f620f6c789ae17a83ff1e11a7c49506db9193c675d046e6L
```

Note que as funções fornecidas nos módulos `Crypto` não contêm mecanismos de *padding*. Embora não esteja completamente bem (porque não funciona no caso específico em que a mensagem já tem um tamanho múltiplo de 16 bytes), a instrução seguinte é capaz de alinhar o tamanho de uma `msg` até ao múltiplo de 16 bytes mais próximo de acordo com o PKCS7. Procure perceber o código antes de o usar:

```
msg = msg + (16 - len(msg) % 16) * chr(16 -
    len(msg) % 16)
```

Tarefa 6 *Task 6*

A última tarefa consiste na implementação do algoritmo de decifra do sistema criptográfico estudado. Dada a implementação do algoritmo de cifra, esta implementação deve apresentar-se mais simples. Este programa deve aceitar o nome do ficheiro a decifrar, o valor hexadecimal da chave privada sk, das coordenadas do ponto G e da chave efémera Y, por esta ordem. Na *string* de argumentos, estes *inputs* devem ser, respetivamente, `argv[1]`, `argv[2]`, `argv[3]` e `argv[4]`, e `argv[5]` e `argv[6]`. O ficheiro com código fonte Python que implementam o algoritmo de decifra devem chamar-se `alice-decrypt.py`.