# Trabalho Final de Números Inteiros e Criptografia

Prof. Luis Menasché Schechter

**LEIA TODO O CONTEÚDO DESTE ARQUIVO, ATÉ O FINAL.**

## Instruções Gerais

- Este trabalho deve ser feito *individualmente* e enviado para o professor através da plataforma **Chalkup** até às 23 horas do dia 29 de junho de 2017.
- Trabalhos entregues depois do prazo **não serão aceitos**.
- Todo o código produzido para este trabalho deve ser feito na linguagem de programação **Python 2** e deve conter comentários detalhados ao longo de toda a sua extensão.
- Será valorizado na correção o uso de código escrito pelo próprio aluno, ao invés de funções já prontas do Python, especialmente quando se tratar de algoritmos e funções que constam das atividades de laboratório desenvolvidas ao longo do semestre.

## Descrição do Trabalho

O objetivo deste trabalho é programar em **Python 2** uma versão "genérica" simplificada de softwares como o PGP (Pretty Good Privacy) e GPG (GNU Privacy Guard, software livre que implementa as mesmas funcionalidades do PGP, descritas no padrão OpenPGP). Detalhes sobre estes dois softwares podem ser obtidos nas referências apresentadas no final deste arquivo. De forma básica, estes softwares permitem **criptografar** e/ou **assinar digitalmente** e-mails e também qualquer arquivo salvo no computador, seja ele um arquivo de texto (.txt, por exemplo) ou um arquivo binário (.jpg ou .exe, por exemplo). Tanto o PGP quanto o GPG permitem a escolha entre RSA ou El Gamal para a realização da criptografia e/ou da assinatura digital. Na realidade, no caso da criptografia, estes softwares utilizam uma técnica mista que aplica um método de chave pública (RSA ou El Gamal) combinado com um método de chave privada, com o objetivo de melhorar a sua velocidade de processamento. Entretanto, no nosso caso, trabalharemos a encriptação puramente com métodos de chave pública, já que foram estes que estudamos.

O programa que deverá ser elaborado para este trabalho deverá ser capaz de encriptar e/ou assinar digitalmente qualquer arquivo salvo no computador. Em caso de realização de uma assinatura digital, o método utilizado será o El Gamal. Já no caso de realização de criptografia, o programa deverá oferecer ao usuário a escolha entre RSA ou El Gamal (ou seja, o programa precisa implementar ambos os métodos de criptografia). Deste modo, o programa deverá suportar diversos modos de operação:

1) **Modo de geração de chaves**: Neste modo de operação, o usuário deverá informar se deseja criar um par de chaves de assinatura ou um par de chaves de criptografia. Neste último caso, o usuário deverá também informar se o par de chaves de criptografia será para o método RSA ou El Gamal.

   As chaves geradas deverão ter **256 bits** de tamanho (números de aproximadamente 78 algarismos na base 10). Isto significa que, para chaves RSA, o gerador de chaves deverá encontrar **dois primos de 128 bits** de tamanho, enquanto que para chaves El Gamal, o gerador deverá encontrar **um primo de 256 bits** de tamanho.

   1.1) Chaves RSA:

      1.1.1) Pública: $(n, e)$, onde $n = pq$, tal que $p$ e $q$ são primos distintos, e $\text{mdc}(e, \phi(n)) = 1$.

      2.2.2) Privada: $(n, d)$, onde $n$ é como acima e $ed \equiv 1 \pmod{\phi(n)}$.

   2.2) Chaves El Gamal:

      1.2.1) Pública: $(p, g, c)$, onde $p$ é primo, $g$ é gerador de $U(p)$ e $c \in U(p)$.

      2.2.2) Privada: $a$, tal que $g^a \equiv c \pmod{p}$.

   As chaves deverão ser retornadas para o usuário em formato **hexadecimal**. Um número de 256 bits de tamanho possui **64 algarismos** em formato hexadecimal, logo, para haver uniformidade, todos os componentes das chaves devem ser retornados com 64 algarismos hexadecimais, mesmo que seja necessário completar o valor de um componente com zeros à esquerda.

   Deverá ser oferecida ao usuário a opção de retorno dos componentes das chaves impressos no terminal ou a geração de dois arquivos de texto, o primeiro contendo os componentes da chave pública gerada e o segundo contendo os componentes da chave privada gerada. O processamento de arquivos em Python é muito semelhante a C, de forma que o que foi aprendido em Comp 1 sobre manipulação de arquivos é de fácil adaptação. Para as principais funções de manipulação de arquivos em Python, consulte a referência apresentada no final deste arquivo.

2) **Modo de encriptação (pura)**: Neste modo de operação, o usuário deverá informar se a encriptação será realizada com RSA ou El Gamal e também o nome do arquivo que será encriptado. Por fim, o usuário deverá fornecer ao programa a chave pública de encriptação. Deverá ser oferecida ao usuário a opção da digitação da chave pública no terminal (em hexadecimal) ou a leitura desta chave pública de um arquivo que a contenha. O conteúdo encriptado do arquivo informado deverá ser escrito em um novo arquivo, cujo nome será informado pelo usuário.

3) **Modo de decriptação (pura)**: Neste modo de operação, o usuário deverá informar se a decriptação será realizada com RSA ou El Gamal e também o nome do arquivo que será decriptado. Por fim, o usuário deverá fornecer ao programa a chave privada de encriptação. Deverá ser oferecida ao usuário a opção da digitação da chave privada no terminal (em hexadecimal) ou a leitura desta chave privada de um arquivo que a contenha. O conteúdo decriptado do arquivo informado deverá ser escrito em um novo arquivo, cujo nome será informado pelo usuário.

4) **Modo de assinatura digital (pura)**: Neste modo de operação, a assinatura será necessariamente realizada com El Gamal. O usuário deverá informar o nome do arquivo que terá o seu conteúdo assinado. Além disso, ele deverá fornecer ao programa a chave privada de assinatura. Deverá ser oferecida ao usuário a opção da digitação da chave privada no terminal (em hexadecimal) ou a leitura desta chave privada de um arquivo que a contenha. A assinatura produzida para o conteúdo do arquivo informado deverá ser escrita em um novo arquivo, cujo nome será informado pelo usuário.

   Para o método de assinatura El Gamal, é necessária a utilização de uma função hash $h$. Efetivamente, como vimos em sala, não é calculada uma assinatura para a mensagem, mas sim uma assinatura para o **hash da mensagem**. Não estudamos funções hash no curso, então utilizaremos uma função hash pronta que é considerada segura para fins criptográficos. Esta função é a SHA-224, da família SHA-2 de funções hash. Ela está disponível na biblioteca de funções do Python. Ela calcula um hash de 224 bits de tamanho para qualquer string (de qualquer comprimento).

   Para utilizar a função pronta do Python, devemos colocar no topo do arquivo com o código a seguinte linha:

   ```
   from hashlib import sha224
   ```

   Tendo a string que desejamos calcular o hash armazenada na variável `texto`, por exemplo, podemos calcular o seu hash com a seguinte linha:

   ```
   h = sha224(texto).hexdigest()
   ```

O valor retornado será uma string contendo o hash em **hexadecimal**. Como o hash tem sempre 224 bits, esta string conterá 56 caracteres. Para transformar esta string em um valor inteiro (e na base 10), podemos utilizar

```
h2 = int(h,16)
```

Para mais informações sobre a função `sha224` do Python, consulte a referência apresentada no final deste arquivo.

5) **Modo de verificação de assinatura (pura)**: Neste modo de operação, o usuário deverá informar o nome de dois arquivos: o primeiro contém o conteúdo que foi assinado digitalmente e o segundo contém a assinatura propriamente dita. Além disso, ele deverá fornecer ao programa a chave pública de verificação da assinatura. Deverá ser oferecida ao usuário a opção da digitação da chave pública no terminal (em hexadecimal) ou a leitura desta chave pública de um arquivo que a contenha. O programa deverá então informar ao usuário, através do terminal, se a assinatura é ou não válida.

6) **Modo de assinatura digital e encriptação (combinados)**: Neste modo de operação, devemos primeiro assinar digitalmente o arquivo fornecido. Em seguida, devemos agrupar o conteúdo do arquivo com a assinatura produzida (de alguma forma que seja possível posteriormente separar estes dois componentes novamente) e aplicar a encriptação a este conteúdo agrupado, produzindo um novo arquivo com o conteúdo encriptado.

7) **Modo de decriptação e verificação de assinatura (combinados)**: Neste modo de operação, devemos primeiramente decriptar o conteúdo do arquivo fornecido. Em seguida, devemos separar o conteúdo do arquivo original (decriptado) da assinatura, para que então possamos proceder à verificação desta assinatura, informando o usuário, através do terminal, se ela é válida ou não.

## Entrega do Trabalho

A entrega deste trabalho consistirá de **três arquivos**:

1) Todo o código fonte produzido deverá estar reunido em **um único arquivo**, nomeado como `(DRE do aluno).py` (ex: `106423731.py`).

2) O aluno deverá enviar um manual de uso do seu programa (em formato .doc ou .pdf), explicando em detalhes qual é o passo-a-passo para que o programa realize cada uma das tarefas que ele deve ser capaz de

fazer. Este manual deve ser nomeado como `m(DRE do aluno).doc` ou `m(DRE do aluno).pdf` (ex: `m106423731.doc` ou `m106423731.pdf`).

3) O aluno deverá enviar um relatório (em formato .doc ou .pdf) explicando detalhadamente que algoritmos e funções utilizou em cada etapa, isto é, explicar cada bloco que utilizou na construção do programa como um todo. Por exemplo: por que, em dado passo necessário à execução do programa, foi utilizado o algoritmo X e não o Y? Por que o algoritmo Z era necessário na etapa W? O programa não foi psicografado dos céus, então o aluno deve saber explicar em detalhes todas as suas escolhas para a construção dele e deve saber justificar as ferramentas que foram utilizadas, dentre todas que estudamos ao longo do curso. Este relatório deve ser nomeado como `r(DRE do aluno).doc` ou `r(DRE do aluno).pdf` (ex: `r106423731.doc` ou `r106423731.pdf`).

## Referências


1) PGP na Wikipedia:
`https://en.wikipedia.org/wiki/Pretty_Good_Privacy`

2) Site oficial do PGP:
`https://www.symantec.com/products/information-protection/encryption`

3) GPG na Wikipedia:
`https://en.wikipedia.org/wiki/GNU_Privacy_Guard`

4) Site oficial do GPG:
`https://www.gnupg.org`

5) Especificação oficial do padrão openPGP:
`https://tools.ietf.org/html/rfc4880`

6) Documentação do Python sobre as funções para tratamento de arquivos:
`https://docs.python.org/2/tutorial/inputoutput.html#reading-and-writing-files`

7) Documentação do Python sobre a biblioteca *hashlib*:
`https://docs.python.org/2/library/hashlib.html`