Relatório do projeto de Cripotgrafia RSA

Pedro Henrique Rodrigues Meira Bastos

April 2019

0.1 Introdução

A criptografia RSA foi desenvolvida pelos cientistas Ron Rivest, Adi Shamir e Leonard Adleman, publicada em 1973. Seu prńcipio básico vem da ideia de que, a partir de informações públicas, é possível enviar uma mensagem codificada que apenas o receptor será capaz de decifrar e saber seu conteúdo.

Para tal feito, na criptografia RSA existem duas chaves, a chave pública, que é usada para codificar a mensagem, e a chave privada, utilizada para decodificar a mensagem. Como os nomes sugerem, apenas os interlocutores da mensagem tem acesso a chave privada, sendo que a chave pública também serve como identificador da mensagem.

Para se fazer obter as duas chaves, é necessário o seguinte processo:

- Obter dois números **p** e **q** primos (de preferência randômicos por razões de segurança)
- Calcular $n = p * q$
- Calcular o mínimo multiplo comum de **p-1** e **q-1**, que a partir de agora chamaremos de **lambda(n)**
- Achar um inteiro **e** entre 1 e **lambda(n)** tal que esses dois números sejam coprimos. Os números **n** e **e** compõem a chave pública.
- Calcular **d** que satisfaça $d * e \equiv 1(modn)$, d é a chave privada.

Munidos de chave pública e privada, e dada uma mensagem **m** (já em forma numérica, o processo de codificacação e decodificação é simples

- Para a mensagem codificada **c**, basta fazer $c^e(modn)$
- Para decodificar c e obter m de volta, basta fazer $c^d \equiv m(modn)$

Com um entedimento básico do conceito, o objetivo do exercício era criar um programa em linguagem Python (a versão utilizada foi a 3.6.7) que, a rigor, fosse capaz de gerar chaves públicas e privadas, e utilizá-las em processos de codificação e decodificação.

0.2 Metodologia

O primeiro passo importante, foi a busca pelo entedimento do processo, pois ele envolvia conceitos matemáticos pouco utilizados (ou pelo menos pouco vistos pelo autor até o momento) nas disciplinas de engenharia. Para isso, bastaram-se algumas horas fazendo pesquisas em diversas fontes, sendo as principais o Wikipedia e o Youtube.

O Youtube, em primeiro momento, foi bastante útil, pois nessa plataforma existem vídeos onde professores não apenas explicam, como também exemplificam a criptografia RSA em funcionamento, o que facilita bastante o entedimento.

Por outro lado, a página do Wikipedia em inglês tem um ótimo artigo sobre a criptografia RSA, mostrando o seu processo num forma de passo-a-passo bastante didatica, e mais importante, auxiliou na formação da perspectiva necessária para criar um algoritmo computacional.

Com o conhecimento em mãos, a primeira etapa do processo foi buscar reproduzir, gradativamente, o processo de geração de chaves. Para isso, foi necessário o uso de duas bibliotecas: a *math* que forneceu a função *gcd* que calcula o maximo divisor comum e a *random*, com sua função *uniform* que escolhe pseudoaleatoriamente um numero dentro de um dado intervalo (foram usados intervalos de números pequenos para agilizar a execeção do código, permitindo um número maior de testes).

Com essas bibliotecas, foi possível separar o processo é uma série de funções pequenas, que quando usadas na sequência certa, resultariam na formação do par de chaves pública e privada. Essas funções ficaram guardadas em um arquivo separado, o *support.py*, para manter os outros códigos mais limpos e concisos, além disso, qualquer erro é mais fácil de encontrar e corrigir no formato mais "modularizado" utilizado.

Com essa etapa pronta, partiu-se para a elaboração do módulo de criptografia, uma classe que seria capaz de gerar chaves aleatorias, criptografar e descriptografar mensagens a partir dessas chevas. A elaboração do módulo não foi complexa, bastou empregar as funções do *support.py* corretamente, que o código funcionou corretamente.

A única ressalva foi que exisitiu uma certa dificuldade em relação ao processo de conversão de mensagens, pois elas eram dadas em formato de string e precisavam ser convertidas para valores numéricos, necessitando um esforço de manipulação, mas nada muito difícil.

Por fim, com o módulo pronto, bastava fazer uma aplicação que de

fato utilizasse o módulo de criptografia recém criado. Foi pedido que essa aplicação fosse um simples gerenciador de contas, capaz de armazenar usuários e senhas. O emprego da criptografia se dá porque foi requerido que, quando tal programa fosse encerrado, todos os dados dos usuários fossem criptografados e salvos num arquivo externo, e quando o programa fosse reaberto, tais dados seriam resgatados, descriptografados e recarregados na memória.

Dito isso, o primeiro passo foi criar uma classe usuário, que armazenaria os seus respectivos nome e senha. Com essa classe pronta, o primeiro método da classe cliente foi o método de adição de usuários, que simplesmente pegava uma entrada e instanciava esse novo usuário com essas informações, sendo então essa instância jogada numa lista de usuários dentro da classe cliente.

Em seguida foram feitos os métodos de remover usuários e checar se um dado usuário existe de fato ou não dentro da lista, sendo que ambos foram bem fáceis de desenvolver.

Com esses métodos feitos, o próximo passo foi criar um método de salvar os arquivos. Para isso, o modulo de criptografia foi instanciado, e partir da interação desse módulo com a lista de usuários, os dados puderam ser apropriadamente salvos num arquivo externo, conforme pedido.

Por fim, só faltava criar o método que pudesse recarregar os dados a partir das informações criptografadas no arquivo externo. O processo foi um pouco trabalhoso, por causa da formatação na qual os dados eram apresentados, mas nada que uns minutos de experimentação no terminal do Python não ajudassem a obter o conhecimento necessário.

Apenas uma biblioteca foi usada aqui, *os.path*, com sua função *isfile* que checa se um dado arquivo existe ou não. Essa função é importante pois ela determina se vão existir dados ou não para serem recarregados toda vez que o programa for aberto.

É bom frisar que as chaves pública e privada também são salvas, então ocorreu um tratamento especial no instanciamento do módulo de criptografia, que, caso fosse carregado com um par de chaves, não precisaria gerar novas aleatórias.

Como adendo, foi elaborada uma função *main()*, na qual foi elaborado um simples menu para o terminal do python, assim um usuário mais leigo teria um controle mais direcionado do que apenas a classe cliente instanciada.

0.3 Resultados e discussões

Ao final do processo, foi obtido um programa bem elaborado, de entedimento simples e bastante didático, pois em nehnum momento foram utilizados algoritmos de mais díficil entedimento, o que por um lado é bom, já que facilita a compreensão do código por outrém, mas por outro significa que esse código pode ter um tempo de execução mais alongado, especialmente quando utilizados números muito grandes.

Outra informação importante, que ressalta o caráter didático do projeto, é que o processo de seleção de números primos é pseudoaleatório, e além disso, as chaves privadas são armazenadas em um arquivo .txt externo, o que sinaliza que essa aplicação não é indicada para usos profissionais no mercado, ela tem, de fato, um valor mais educativo.

0.4 Conclusão

Foi possível, com sucesso, criar uma ferramenta que crie chaves aleatoriamente (ou quase que aleatoriamente), codifique e decodifique mensagens com tais chaves.

0.5 Referências

- `https://en.wikipedia.org/wiki/Modular_multiplicative_inverse`
- `https://en.wikipedia.org/wiki/RSA_(cryptosystem)`