Primeiro Projeto de Matematica Discreta

Henrique da Silva
hpsilva@proton.me

14 de agosto de 2022

Sumário

1 Introdução

Neste relatório, vamos discutir e implementar o sistema RSA.

Para ter as ferramentas necessarias para construi-lo, primeiro precisamos construir algumas ferramentas, estas serao discutidas nas secoes 2, 3, 4, e 5.

A inspiracao para escolha do projeto de chaves RSA eh que algo que eu ja utilizo diariamente, a autenticacao feita pelo *git* utiliza um par privado e publico de chaves RSA**.**

Objetivo do projeto eh conseguir gerar um par seguro de chaves *RSA* e utiliza-lo para fazer minha autenticacao no *git*.

Todos arquivos utilizados para criar este relatorio, e o relatorio em si estão em: `https://github.com/Shapis/ufpe_ee/tree/main/4thsemester/`

2 O Codificador de texto

Este foi criado para transformar uma *string* de texto em um *int*, Atravez de dois metodos. TextoParaInteiro(string) e InteiroParaTexto(int).

2.1 TextoParaInteiro

Este metodo recebe um texto e o torna em um m do tipo *int* da seguinte maneira:

$$m = \sum_{i=0}^{N-1} cod(a_i) * 27^i \tag{1}$$

Com *a* ate *z* sendo definidos como 1 ate 26, "espaco"sendo definido como 27.

2.2 InteiroParaTexto

Para retornar o texto, este metodo recebe um inteiro m e faz a seguinte operacao:

$$a_i = cod\left(\frac{m}{27^i} \pmod{27}\right) \tag{2}$$

Para todo i que nao faca m ser menor que 1

2.3 Restricoes e limitacoes

A principal restricao eh que isto foi implementado usando o tipo int do $C\#$ que tem 32 bits. Porem, ja que ele contem tanto numeros positivos quanto negativos o valor maximo dele eh de:

$$\frac{2^{32}}{2} - 1 = 2147483647 \tag{3}$$

Estamos codificando o texto de maneira que cada digito ocupa ate: $2^N = 27$, $N = \frac{\log 27}{\log 2}$ bits

Entao a quantidade maxima de bits ocupados eh simplesmente $N * L$

Para o nosso caso em especifico, que o tipo int tem $2^{31} - 1$ de tamanho, ou seja, seguramente ate 31 bits. Temos que:

$$L * \frac{\log 27}{\log 2} \leq 31 \tag{4}$$

Que nos da $L = 6$, ou seja, podemos seguramente converter ate 6 caracteres para tipo int e converte-los de volta.

Vale notar, que isto eh um limite inferior de seguranca. Na verdade temos 6.3 digitos disponiveis, que nos permitiria por exemplo, guardar e recuperar, uma frase de sete digitos do tipo $zzzzzd$, Mas para ter certeza. Tem de ser 6 ou menos digitos.

3 A classe *BigNumber*

Esta sera uma classe que armazenara os numeros que utilizarei para a criacao do RSA.

Utilizarei como base para meu $BigNumber$ a classe $BigInteger$ do $C\#$, que tem limite de tamanho tao grande quanto couber na memoria do computador que o esta utilizando.

Para o nossos fins, queremos um $BigNumber$ que tenha no maximo 2048 bits. Entao para todas operacoes de $BigNumber$ incluindo a sua propria criacao, criarei um $SafetySizeCheck$ que caso o $BigNumber$ exceda 2048 bits, ele ira lancar uma excecao e parar o programa com a mensagem de erro apropriada.

Importante lembrar que inclui o zero no $BigNumber$, entao na verdade o limite superior dele fica da seguinte maneira:

$$BigNumber \leq 2^{2048} - 1 \tag{5}$$

E tambem importante lembrar que todas operacoes de checagem de seguranca ocorreram *apos* a operacao ser realizada.

Ou seja, o programa permitira operacoes inseguras, desde que o $BigNumber$ resultante desta operacao insegura nao exceda 2048 bits.

3.1 Multiplicacao de *BigNumber*

Aqui podemos observar o seguinte:

$$2^a * 2^b = 2^{a+b} \tag{6}$$

Entao a multiplicacao de dois $BigNumber$ de tamanho a e b, pode no maximo nos dar um $BigNumber$ de tamanho $a + b$

3.2 Soma de *BigNumber*

Neste caso temos o seguinte:

$$2^a + 2^a = 2 * (2^a) = 2^1 * 2^a = 2^{a+1} \tag{7}$$

Logo podemos concluir que no maximo a soma de dois numeros de tamanho N bits dara um numero de tamanho $N + 1$ bits.

3.3 Codificando numeros grandes

Inicialmente, notemos que o ***Codificador de Texto*** discutido na secao dois, tinha limitacao de utilizar o tipo int de 31 bits. Que nao sera suficiente para nossos propositos.

Entao escrevi dois novos metodos ***TextoParaBigNumber***, e ***BigNumberParaTexto***

Estes tendo as mesmas limitacoes porem alterando tamanho do nosso numero de 31 bits para 2048 bits.

Resolvendo a equacao (4) para 2048, teremos que o nosso L deve se limitar a no maximo 430 caracteres.

4 Aritmetica Modular

4.1 AddMod

As limitacoes aqui sao as mesmas da soma de dois $BigNumber$ como vimos acima em (7).

A funcao AddMod pode no maximo dar um $BigNumber$ de tamanho $N + 1$ bits, N sendo o tamanho do maior dos dois $BigNumber$.

4.2 MulMod

Vimos acima em (6) as limitacoes de multiplicacao de dois $BigNumber$.

Entao no maximo a soma dos tamanhos em bits dos nossos $BigNumber$ deve dar 2048 que eh o tamanho que escolhemos para o nosso $BigNumber$

4.3 ExpMod

No caso da exponenciacao precisamos que o produto dos tamanhos dos dois $BigNumber$ seja menor que 2048

4.4 InvMod

Para resolver a congruencia linear utilizamos o algoritmo de euclides extendido. E a operacao de maxima ordem que utilizamos em todas operacoes eh a de multiplicacao de $BigNumber$ que descrevemos em (6)

Logo, nossa limitacao para garantir que nao vamos exceder os **2048** bits do $BigNumber$ eh que a soma em pares, de a, b, e n nao exceda 2048 bits.

5 Busca por numeros primos

utilizarei o metodo de Miller Rabin para testar a primalidade dos numeros.

5.1 Testando os numeros dados:

$2^{521} - 1$	$\rightarrow$	primo
$2^{523} - 1$	$\rightarrow$	nao primo
$2^{607} - 1$	$\rightarrow$	primo

5.2 Achando novos primos:

Para achar novos numeros primos criarei um novo $BigNumber$ de tamanho n, e testarei numeros impares menores que este $BigNumber$ ate o teste de Miller Rabin me retornar que provavelmente eh um primo.

Para adicionar um elemento de aleatoriedade. Apos checar um numero, vamos subtrair a este $2 * x$ com x variando entre 0 e n aleatoriamente.

Esta maneira de gerar numeros aleatorios, me garante que todo numero gerado tera tamanho menor do que n bits. Porem, ela me retorna numeros primos repetidos com alguma frequencia.

Com alguns testes numeros, tive a chance de aproxidamente 1 em 5000 de obter dois numeros primos identicos. O que para um sistema que sofra ataques com certeza nao seria aleatoriedade suficiente. Mas para nossos propositos de testar a criacao de um sistema RSA que nao vai sofrer ataques. Sera o suficiente.

6 O Sistema RSA

Vamos utilizar de todas ferramentas que criamos acima para montar o nosso sistema.

6.1 Geracao do par de chaves RSA

Inicialmente vamos achar dois numeros primos, p, e q de 512 bits aleatorios.

Vamos definir $n = pq$ e $\phi(n) = (p - 1)(q - 1)$

Vamos tambem escolher um ϵ tal que $MDC(\epsilon, \phi(n)) = 1$. Para simplicidade, escolhi o menor ϵ para qual esse MDC seja 1.

E atravez do sistema de resolucao de congruencias lineares que criamos na secao 4.4, vamos computar um d da seguinte maneira:

$$\epsilon d \equiv 1 (mod \phi(n)) \tag{8}$$

Entao, afinal temos um ϵ e um n que serao nossa chave publica, e um d que sera nossa chave privada.

Ambos chaves privadas e publicas serao salvas em arquivo.

6.2 Codificando e decodificando

Vamos utilizar do nosso ***Codificador de Texto*** nesta etapa.

Com tudo que temos em maos, os passos para codificacao e decodificacao sao simples como veremos a seguir:

6.2.1 Codificacao

$$Y \equiv X^{\epsilon} \pmod{n} \tag{9}$$

Onde X eh a mensagem em texto limpo. Y a mensagem criptografada, e ϵ e n sao a chave publica.

Isto foi implementado na classe RSA, e para simplificar, $RSA.Codificar$ recebe textos limpos e a chave publica, e retorna textos criptografados ao invez de cifrar o $BigNumber$

6.2.2 Decodificando

$$Z \equiv Y^{d} \pmod{n} \tag{10}$$

Onde Z eh a mensagem em decodificada. Y a mensagem criptografada, e d eh a chave privada.

Importante de notar que o seguinte eh verdade:

$$Z \equiv X \pmod{n} \tag{11}$$

Isto foi implementado na classe RSA, e para simplificar, $RSA.Decodificar$ recebe textos cifrados e a chave privada, e retorna textos limpos

6.3 Testando para um texto predeterminado

Quando executado o programa ira mostrar um texto ao usuario, codifica-lo, mostrar o texto codificado, e apos isso decodificar e mostrar o texto decodificado. O texto decodificado deve ser o mesmo que o texto original.

O embasamento teorico disto esta descrito na secao 6.2.

6.4 Testado para texto arbitrario Y

O programa ira pedir do usuario um texto, vamos codificar este texto, exibi-lo codificado, e apos isso decodificar e exibi-lo novamente. O texto decodificado deve ser o mesmo que o texto original.

Lembrando que ha limitacao de tamanho do texto em 430 caracteres como visto na secao 3.3

6.5 Assinatura digital

Para assinar um documento X, basta divulgar o Y de:

$$Y \equiv X^{d} \pmod{n} \tag{12}$$

E para verificar, fazemos o oposto que eh:

$$Z \equiv Y^{d} \pmod{n} \tag{13}$$

Lembrando da equacao (10) que nos da que $Z \equiv X$ ja que:

$$Z \equiv X^{de} \equiv X \pmod{n} asdas \tag{14}$$

Nesta etapa. o programa ira assinar o texto que o usuario escreveu anteriormente, e verificar a assinatura.

Importante notar que assinatura eh um caso parecido com o de codificacao de texto. Exceto que neste caso vamos cifrar o texto limpo com nossa chave privada, e decodificar o texto cifrado com a chave publica.

6.6 Autenticacao de remetente e destinatario

Neste caso, temos dois usuarios A e B.

O usuario A fara a codificacao da mensagem com a chave publica de B, assim garantido que so B lera a mensagem, e enviara junto com a mensagem a sua assinatura digital.

O usuario B usara sua chave privada para decodificar a mensagem, e usara a chave publica de A para verificar a assinatura digital.

O programa vai guiar o usuario A em enviar uma mensagem assinada para o usuario B, e o usuario B verificara que a mensagem foi assinada por A.