Documentação Trabalho Prático – Primeira Entrega (10/06)

Aluno: Lucas Resende Pellegrinelli Machado
Matrícula: 2018048621
Turma: TM3

O Problema

O problema proposto pela primeira parte do trabalho se resume em um software escrito na linguagem C cuja função é executar a cifragem e decifragem de mensagens utilizando do método de criptografia RSA.

Esse software deve ter duas partes: o codificador (responsável por receber a mensagem que será cifrada e os parâmetros que serão utilizados no processo de cifragem, mais especificamente, os dois números primos necessários), e o decodificador (responsável por receber os blocos de mensagem já cifrados e retornar ao usuário a mensagem original mediante o conhecimento de variáveis que são obitidas pelos dois primos informados na hora da codificação).

Na próxima fase do projeto (a entrega final do trabalho), o software deverá ser atualizado a fim de possuir um sistema de esteganografia, relacionado com a "camuflagem" de mensagens dentro de uma imagem.

Descrição do algoritmo

• Codificador:

O processo de codificar consiste em:

1. Transformar a mensagem original em uma representação numérica para trabalhar com as fórmulas do RSA. O procedimento escolhido nesse ponto na maioria dos casos[1] foi a simples conversão de cada caractere da mensagem em um valor decimal por meio da tabela ASCII. Após converter cada caractere para números, uma string é criada com todos os números em sequência para armazena-los.

2. Depois de ter a string com a sequência dos caracteres já transformados em números, precisamos dividi-la em blocos tal que o valor numérico de cada um desses blocos seja menor que N. Dessa forma, a string é particionada até que cada bloco seja o maior possível (possuía a maior quantidade de números) respeitando o limite do valor N.

3. Logo depois de obter os blocos contendo a representação numérica dos caracteres, foi aplicada o RSA para os blocos utilizando a fórmula:

   $$c \equiv m^e \left(mod\ N \right)$$

   Onde '**m'** é o bloco a ser cifrado, '**e'** é uma das variáveis calculadas a partir dos primos dados ao algoritmo e **'N'** é outra variável proveniente dos primos.

4. Após essa cifragem, temos como resultado nossa mensagem criptografada, que é informada ao usuário de modo que seja impresso na tela todos os blocos já cifrados, separados por uma barra ("/") em uma só linha.

1 Existem casos tratados no item "Decisões na Implementação" que necessitam conversão entre bases númericas além da conversão de caractere para ASCII.

5. Para que seja possível decifrar a mensagem, também é criado um arquivo de nome “private.txt” que contém duas linhas: Na primeira, o valor de d e na segunda o valor de N, ambas variáveis que são obtidas a partir dos primos que foram indicadas ao programa e são necessárias no processo de decodificação.

- Decodificador:

Já o processo de decodificar uma mensagem é dado por:

1. É necessário ter os valores das variáveis calculadas durante a codificação para decifrar a mensagem, dito isso, o primeiro passo é ler do arquivo indicado os valores de d e N.

2. Logo após a consulta das variáveis, é necessário primeiro separar a string contendo os blocos na forma numérica, para isso separamos a string onde existem barras (“/”). Com os blocos já em mãos, aplicamos o processo reverso do algoritmo RSA encarregado de decodificar os blocos que obtemos, que estão codificados visto que vieram do codificador. Esse processo é dado pela fórmula:

   $$m = c^d \pmod N$$

   Onde **‘c’** é o bloco a ser decodificador, ‘**d’** é a variável obitida no arquivo tal como **‘N’**.

3. Com os blocos já decodificados em mãos, é necessário agora remontar a string completa com todos os números em sequência e depois descobrir onde separar essa grande string para que cada partição resulte em cada um dos caracteres originais da mensagem, para isso temos que seguir o seguinte padrão da tabela ASCII:

   Números abaixo de 32 não são caracteres como letras e números, e sim códigos de sistema, que não serão encontrados em um texto. Dito isso, ao começar a leitura da string procurando os caracteres, existem dois casos:

   - Se a string começar com 1, a única opção pláusivel é que seja um número com três digitos, (dito que caso contrário, o número seria entre 10 e 19 ou seria apenas o número 1, intervalos onde não existem caracteres utilizáveis como dito antes), portanto assumimos que os dois dígitos seguintes do número 1 encontrado fazem parte do número correspondente ao caractere em questão. Logo convertemos o número e os dois dígitos seguintes encontrados de ASCII para caractere e assim o caractere original foi encontrado.

   - Não existirá casos onde a string começa com 2 visto que as opções seriam simplesmente o número 2, que não possuí caractere relacionado, os números entre 20 e 29, que também não se relacionam com nenhum caractere ou números entre 200 e 299, que vão além do limite máximo da tabela ASCII de 127.

   - Se a string começar com um número maior ou igual a 3, portanto é pláusivel assumir que seria um número com dois dígitos (dito que caso contrário, ou o número teria apenas um dígito, colocando o caractere correspondente a ele no intervalo onde não existem caractéres de texto, ou o número teria três dígitos e não começaria com 1, portanto estaria além dos limites da tabela ASCII visto que ela suporta valores de 0 até 127), logo convertemos apenas o primeiro e o segundo número do formato ASCII para caractere e assim o caractere original foi encontrado.

   Repetindo esse algorítmo para todos os caracteres da string com todos os números, teremos então o texto original.

Decisões na implementação:

1. Utilização de métodos mais eficientes de exponenciação.

   Ao utilizar a fórmula da cifragem e decifragem do RSA, caso os valores para os primos sejam relativamente grandes, operações de exponenciação com expoentes cada vez maiores vão aparecendo. Mas devido ao limite do tamanho de tipos de variáveis como int e long na linguagem C, exponenciações relativamente pequenas já não eram possíveis dito que ultrapassavam o limite do valor máximo da variável.

   Para ajudar a melhorar a situação, foi utilizado a estratégia denominada como “Fast Modular Exponentiation” que reduz qualquer exponenciação a várias exponenciações de exponente 2. Dessa forma podemos trabalhar com valores bem maiores dito que não ultrapassaremos os limites das variáveis tão facilmente quando utilizamos apenas exponenciações de ordem 2.

   Após isso, não existem mais problemas quanto ao limite de valor de tipos de variáveis durante a exponenciação.

2. Utilização de números em bases diferentes de 10 quando os primos são pequenos.

   Dito que precisamos quebrar a mensagem em blocos cujo valor de cada bloco é menor que N (como dito na seção de codificação), quando N for igual a 6 por exemplo (p = 2 e q = 3), é possível que exista um caractere de valor 98 por exemplo, que seja incapacitado de ser quebrado em blocos dito que tanto 9 quanto 8 são maiores que N. Logo, a solução encontrada foi que quando o caso é N menor que 10, ou seja, existe possibilidade de não ser possível representar algum número escrito no sistema usual (base 10) em blocos de valor menor que N, todos os valores ASCII obitidos pelos caracteres da mensagem são convertidos para uma base menor que N, possibilitando assim a separação dos blocos corretamente.

   Na hora de decodificar a mensagem, como temos o valor de N, também é possível identificar se essa conversão foi feita, e se foi, desfaze-la, não causando problemas para a integridade do programa.

3. Restrição do usuário a primos p e q tal que p * q < 4294967296

   Dito que o programa tem que dividir a mensagem em blocos cujo valor de cada bloco não excede o valor de N, caso N (ou seja, p * q) seja maior ou igual a raiz quadrada do valor máximo do tipo de variável unsigned long que é 18446744073709551615 (a raiz desse número é 4294967296), temos um problema dito que, na hora de utilizar o algoritmo de “Fast Modular Exponentiation”, precisamos elevar o valor do bloco ao quadrado, e caso esse bloco tenha a possibilidade de ter o valor superior a 4294967296, ao computar o quadrado desse número, é possível ter um valor que excede 18446744073709551615, tornando impossível armazena-lo com as variáveis nativas do C.

   Logo é informado uma mensagem de erro ao usuário quando ele tenta utilizar valores para p e q tal que p * q supere 4294967296.

Detalhes sobre cada função utilizada:

Os detalhes e a explicação de todas as funções criadas para a resolução do problema se encontram nos comentários acima do protótipo de cada uma delas no código. Para acha-las, segue o guia:

- Para funções exclusivas do codificador (mais especificamente a função criada com fim de criar o arquivo com as variáveis calculadas), o arquivo onde se deve procurar a documentaçao é o **codificador.c**

- Para funções exclusivas do decodificador (mais especificamente a função criada com fim de ler as variáveis calculadas durante a codificação e salvas no arquivo de texto), o arquivo onde se deve procurar a documentação é o **decodificador.c**

- Para funções relacionadas a todo os passos da codificação e decodificador (alguns exemplos são os códigos para quebrar e mensagem em blocos e reconstruir a mensagem em blocos, aplicar as fórmulas RSA, calcular as variáveis derivadas dos primos, algoritmo da “Fast Modular Exponentiation”, etc…), o arquivo onde se deve procurar a documentação é o **rsa.h**