# Redes de Computadores II: Segundo Trabalho

Prof. Diego Passos, Universidade Federal Fluminense 2/2015

## Descrição e Objetivo

Este trabalho consiste na implementação de um programa (ou um par de programas) que cifre/decifre arquivos utilizando o algoritmo de criptografia de chave pública RSA. Seja na forma de um único programa parametrizável ou de um par de programas, a implementação deve:

- cifrar um arquivo qualquer especificado pelo usuário;
- decifrar um arquivo previamente cifrado especificado pelo usuário.

Em ambos os casos de uso, o usuário especificará a chave a ser utilizada na forma de um par de números inteiros ($n$ e $d$ ou $n$ e $e$). A implementação pode assumir que $255 < n < 65536$.

O programa deve cifrar o arquivo de entrada utilizando **blocos de 1 byte**. Em outras palavras, cada byte será cifrado individualmente, independente dos demais. Como $n < 65536 = 2^{16}$, o texto cifrado correspondente terá, no máximo, 2 bytes. Logo, cada byte original cifrado deve ser representado como um **número inteiro de 16 bits** no arquivo de saída. **Note que, mesmo que o texto cifrado seja menor ou igual a 255, o mesmo deve ser escrito no arquivo de saída com 2 bytes**.

Por outro lado, a parte da implementação responsável por decifrar um arquivo cifrado deve ler o arquivo de entrada especificado pelo usuário em blocos de 2 bytes. Cada bloco deve ser decifrado usando a chave provida pelo usuário. O programa **deve assumir que o texto plano correspondente sempre terá comprimento máximo de 1 byte**. Se, ao decifrar um bloco, o programa encontrar um texto plano maior que 255, **a execução deve ser abortada**, exibindo-se uma mensagem apropriada ao usuário, explicando que o arquivo de entrada era inválido. Arquivos também devem ser considerados inválidos por este módulo se possuírem um número ímpar de bytes.

Cuidados especiais devem ser tomados para evitar a ocorrência de *overflow* durante as exponenciações realizadas pelo código. Um método rápido para execução destas exponenciações modulares sem risco de *overflow* é a chamada *exponenciação binária*. Na sua forma mais simples, este método pode ser implementado conforme mostrado no Algoritmo 1.

Embora o código mostrado seja escrito em C, implementações equivalentes podem ser realizadas em qualquer linguagem. O código se baseia na seguinte propriedade da exponenciação:

$$b^e = \begin{cases} \left(b^2\right)^{\frac{e}{2}} & \text{se } e \text{ é par} \\ b \cdot \left(b^2\right)^{\frac{e-1}{2}} & \text{se } e \text{ é ímpar} \end{cases} \quad (1)$$

Algoritmo 1: Implementação simples do método de exponenciação binária em C.

```c
/*
 * Argumentos:
 *  - b: base da exponenciação.
 *  - e: expoente.
 *  - n: módulo (n > 1).
 * Retorna: b^e (mod n).
 */
unsigned short binExp(unsigned short b, int e, unsigned short n) {

    unsigned int res = b;
    unsigned int y = 1;

    /* Caso base. */
    if (e == 0) return(1);

    while(e > 1) {

        if (e & 1) {

            /*
             * Caso especial: expoente é ímpar.
             * Acumular uma potência de 'res' em 'y'.
             */
            y = (y * res) % (unsigned int) n;
            e = e - 1;
        }

        /*
         * Elevamos 'res' ao quadrado, dividimos expoente por 2.
         */
        res = (res * res) % (unsigned int) n;
        e = e / 2;
    }

    return((unsigned short) ((res * y) % n));
}
```

A ideia do algoritmo é progressivamente reduzir o expoente por um fator de 2, enquanto a base é aumentada de acordo. A cada iteração, obtemos um novo valor para a base elevando seu valor atual ao quadrado. Como consequência, o expoente atual pode ser dividido por 2. Um caso especial ocorre quando, em uma dada iteração, o expoente é ímpar: neste caso, para evitar expoentes fracionários, a base é colocada em evidência e acumulada (através de uma multiplicação) em uma variável auxiliar. Ao final do processamento (*i.e.*, quando o expoente chega a 1), a base atual é multiplicada pela variável auxiliar para se chegar ao resultado final.

Considere, por exemplo, o problema de calcular a potência $2^11$. Na primeira iteração, como 11 é ímpar, coloca-se a base (2, neste ponto) em evidência, multiplicando seu valor pelo valor atual da variável auxiliar (inicializada com 1). Agora, como o expoente passa a ser par, pode-se dividi-lo por 2, elevando a base ao quadrado. Obtém-se, então, $2 \cdot \left(2^2\right)^5 = 2 \cdot 4^5$. Em uma nova iteração, nota-se que, novamente, o expoente é ímpar. Repete-se o processo, colocando a base (agora igual a 4) em evidência e alterando base e expoente, obtendo-se $2 \cdot 4 \cdot \left(4^2\right)^2$ = $8 \cdot 16^2$. Na próxima iteração, como o expoente já é par, basta manipular expoente e base, obtendo-se $8 \cdot 256^1$. Com o expoente igual a 1, as iterações terminam e basta multiplicar a base pela variável auxiliar obtendo o resultado final (2048, neste exemplo).

Como no caso de interesse deste trabalho a exponenciação é realizada em módulo $n$, a cada atualização da base ou da variável auxiliar podemos calcular o módulo, de forma a manter os valores de ambas as variáveis relativamente baixos. Particularmente, note que a base (valor a ser cifrado/decifrado) é, necessariamente menor que $n < 2^16$ e isso se mantém verdade a cada nova iteração por conta do módulo. Logo, $b^2 < \left(2^{16}\right)^2 = 2^{32}$. Isto significa que a multiplicação realizada na linha 31 nunca resultará em um número com mais de 32 bits (tamanho do tipo `int` em C), evitando a possibilidade de *overflow* na atualização da base. O mesmo raciocínio pode ser usado para mostrar que nunca haverá *overflow* na atualização da variável auxiliar `y`.

O trabalho poderá ser feito em grupos de **até 4 alunos**. A linguagem de programação é de livre escolha de cada grupo, assim como detalhes de interface com o usuário (*e.g.*, interface gráfica, interface de linha de comando, obtenção dos nomes dos arquivos de entrada e saída, obtenção da chave). No entanto, todos os passos do RSA deverão ser implementados pelo grupo, incluindo os passos de exponenciação modular. **Não serão aceitos trabalhos que utilizem bibliotecas/funções de terceiros para estas funcionalidades**.

Para facilitar a execução de testes para a verificação da corretude da implementação, pode-se utilizar o seguinte par de chaves pública e privada: (2881, 1625) e (2881, 29).

## Requisitos

A implementação deverá cumprir os seguintes requisitos:

1. possuir um módulo para cifrar e um módulo decifrar, seja em um único programa executável ou em dois programas separados;

2. obter os nomes dos arquivos de entrada e saída a partir do usuário (a maneira pela qual isso é feito não é importante);

3. obter o valor da chave a ser usada para cifrar/decifrar o arquivo de entrada (a maneira pela qual isso é feito não é importante);
4. o módulo para cifrar deverá realizar a cifra em blocos de 1 byte e gerar como saída um arquivo no qual cada bloco de 16 bits consecutivos corresponde ao byte do arquivo original corretamente cifrado com o RSA utilizando a chave especificada;
5. o módulo para decifrar deverá operar sobre blocos de 2 bytes, gerando como **saída** bytes individuais correspondentes ao texto plano decifrado pelo RSA utilizando a chave especificada;
6. o módulo para decifrar deverá verificar se o texto plano associado a um bloco de entrada é maior que 255; neste caso, a execução deverá ser abortada, informando o erro ao usuário;
7. o módulo para decifrar deverá verificar o arquivo de entrada possui um número ímpar de bytes; neste caso, a execução deverá ser abortada, informando o erro ao usuário;
8. ambos os módulos devem gerar os mesmos resultados independentemente do *endianess* do computador no qual são executados.

## Data de Entrega

O prazo para a entrega do trabalho vai até o dia 25 de fevereiro, às 22:00. O trabalho deverá ser entregue por e-mail, através do endereço `dpassos@ic.uff.br`. O e-mail deverá conter:

- identificador do trabalho (*e.g.*, “Trabalho 2”);
- lista dos integrantes do grupo;
- código fonte da implementação; e
- instruções de compilação/execução/uso da implementação.

Os e-mails de entrega de trabalho terão seus recebimentos devidamente confirmados. **É responsabilidade do grupo garantir que o trabalho seja recebido, aguardando pela confirmação e reenviando a mensagem caso não a recebam**.

Em caso de dúvidas ou correções relacionadas a esta especificação, também é responsabilidade de cada grupo entrar em contato (seja pessoalmente, ou através do mesmo endereço de e-mail) requisitando esclarecimentos **dentro do prazo de entrega do trabalho**.

## Critério de Avaliação

Os trabalhos serão avaliados em uma escala de 0 a 10 pontos. A avaliação será dividida em duas partes:

- Aderência aos requisitos (até 1 ponto por requisito).

- Existência e qualidade das instruções de compilação/execução/uso da implementação (até 2 pontos).

A cada item avaliado, poderão ser atribuídas frações das pontuações máximas. Trabalhos entregues fora da data serão aceitos, **mas com uma penalidade de 1 ponto por dia (ou fração) de atraso**.