Universidade Federal de Minas Gerais
Instituto de Ciências Exatas – ICEx
Departamento de Matemática

Álgebra A
Trabalho Prático 3: RSA livro–texto

Apesar de uma função `codifica` de boa qualidade (com técnicas como *padding*) ser essencial para a segurança do RSA na vida real, os detalhes de implementação de uma versão `codifica` robusta fogem ao escopo do nosso curso, que é introduzir os fundamentos matemáticos de aritmética modular. O PDF de avisos contém mais detalhes, incluindo implementações robustas do RSA que você pode usar caso queira proteger dados na prática.

Questão 1. Implemente uma função `codifica`, que recebe um texto com até 500 caracteres e retorna um número correspondente a ver esse texto como um número em base 256.

Mais precisamente, cada um dos caracteres da string `s` tem um código ASCII; você pode obter o código ASCII do `i`-ésimo caractere convertendo `s[i]` para um inteiro. Se o código ASCII do `i`-ésimo caractere da string é s_i e a string tem `n` bytes, você deve retornar o número

$$\sum_{i=0}^{n-1} s_i \cdot 256^i.$$

Valor de retorno	Não há
Assinatura	`void codifica(mpz_t r, const char *str)`

	Nome	Tipo	Descrição
Entrada	`str`	`const char *`	O texto a ser codificado.
Saída	`r`	`mpz_t`	O resultado da fórmula acima.

Questão 2. Implemente uma função `decodifica`, que desfaz a função `codifica`, retornando um `char *`. A função deve alocar memória, e não é responsabilidade da função liberar tal memória. Você pode assumir que o resultado tem no máximo 500 caracteres.

Valor de retorno	Uma string, o número `n` decodificado.
Assinatura	`char *decodifica(const mpz_t n)`

	Nome	Tipo	Descrição
Entrada	`n`	`mpz_t`	O número a ser decodificado.

Como testar: Para qualquer string `str` de até 500 caracteres, tem que valer que `decodifica(codifica(str)) == str`.

As funções abaixo irão fazer a criptografia RSA em si.

Questão 3. Implemente uma função `criptografa`, que recebe números `n`, `e` e M e retorna C, a versão criptografada do número M.

Valor de retorno Não há
Assinatura
```
void criptografa(mpz_t C,
                 const mpz_t M,
                 const mpz_t n,
                 const mpz_t e)
```

	Nome	Tipo	Descrição
Entrada	`M`	`mpz_t`	O número a ser criptografado.
	`n`	`mpz_t`	(n, e) é a chave pública.
	`e`	`mpz_t`	
Saída	`C`	`mpz_t`	Versão criptografada de M.

Questão 4. Implemente uma função `descriptografa`, que recebe `n`, `d` e C e retorna M, a versão descriptografada do número C.

Valor de retorno Não há
Assinatura
```
void descriptografa(mpz_t M,
                    const mpz_t C,
                    const mpz_t n,
                    const mpz_t d)
```

	Nome	Tipo	Descrição
Entrada	`C`	`mpz_t`	O número a ser descriptografado.
	`n`	`mpz_t`	(n, d) é a chave privada.
	`d`	`mpz_t`	
Saída	`M`	`mpz_t`	Versão descriptografada de C.

Questão 5. (Não vale ponto no Moodle, mas é divertida de fazer com as funções dos TPs antigos) Escreva uma função `gera_chaves`, que faz o seguinte:

1. Gera dois primos aleatórios p e q no intervalo $[2^{2047}, 2^{2048})$. Seja $n = p \cdot q$.
2. Acha o menor $e \geqslant 65537$ tal que (n, e) é uma chave pública válida.
3. Gera d tal que (n, d) seja a chave privada correspondente à chave pública (n, e).

Valor de retorno Não há

Assinatura
```
void gera_chaves(mpz_t n, mpz_t e, mpz_t d,
                 gmp_randstate_t rnd)
```

	Nome	Tipo	Descrição
Saída	`n`	`mpz_t`	Um número da forma $n = p \cdot q$.
	`e`	`mpz_t`	Um número $e \geqslant 65537$ tal que (n, e) é uma chave pública válida.
	`d`	`mpz_t`	Um número d tal que (n, d) é a chave privada correspondente a (n, e).
E/S	`rnd`	`gmp_randstate_t`	O estado do gerador aleatório.