

Álgebra A

Trabalho Prático 3: RSA livro–texto

Nas próximas questões, você vai gerar um par de chaves e ser capaz de criptografar uma mensagem. Apesar de ser essencial para a segurança do RSA na vida real, não usaremos *padding* ao criptografar nossas mensagens (vide PDF de avisos).

Questão 1. Escreva uma função `gera_chaves`, que faz o seguinte:

1. Gera dois primos aleatórios p e q no intervalo $[2, 2^{2048})$. Seja $n = p \cdot q$.
2. Acha o menor $e \geq 65537$ 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	<code>mpz_t</code>	Um número da forma $n = p \cdot q$.
	e	<code>mpz_t</code>	Um número $e \geq 65537$ tal que (n, e) é uma chave pública válida.
	d	<code>mpz_t</code>	Um número d tal que (n, d) é a chave privada correspondente a (n, e) .
E/S	<code>rnd</code>	<code>gmp_randstate_t</code>	O estado do gerador aleatório.

Questão 2. 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 do texto tem um código ASCII. 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	<code>str</code>	<code>const char *</code>	O texto a ser codificado.
Saída	<code>r</code>	<code>mpz_t</code>	O resultado da fórmula acima.

Questão 3. 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	<code>n</code>	<code>mpz_t</code>	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 4. 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	<code>M</code>	<code>mpz_t</code>	O número a ser criptografado.
	<code>n</code>	<code>mpz_t</code>	(<code>n</code> , <code>e</code>) é a chave pública.
	<code>e</code>	<code>mpz_t</code>	
Saída	<code>C</code>	<code>mpz_t</code>	Versão criptografada de <code>M</code> .

Questão 5. 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	<code>C</code>	<code>mpz_t</code>	O número a ser descriptografado.
	<code>n</code>	<code>mpz_t</code>	(<code>n</code> , <code>d</code>) é a chave privada.
	<code>d</code>	<code>mpz_t</code>	
Saída	<code>M</code>	<code>mpz_t</code>	Versão descriptografada de <code>C</code> .