Cibersegurança (CS) **Problemas**

Módulo I - **Mecanismos para proteção da informação**

A avaliação prática do Módulo I - Mecanismos para proteção da informação, da U.C. de CiberSegurança, consiste **na entrega das implementações indicadas em 4 problemas** desta lista até o dia **4 de novembro de 2020**.

Cada problema consta de uma primeira parte teórica, que serve de apoio à compreensão da matéria e de modelo para as questões do exame final, e de uma implementação prática, preferentemente em Python3. As implementações devem incluir, **obrigatoriamente**, testes que verifiquem os resultados obtidos na primeira parte correspondente.

Cada implementação terá uma cotação máxima de 5 valores. Será valorizada a clareza do código, os comentários do mesmo e a inclusão de outros testes para além dos obrigatórios.

Salienta-se que as implementações das cifras solicitadas nesta lista têm como intuito a compreensão dos mecanismos de cifra e não devem ser usadas **NUNCA** no âmbito profissional.

---

1. **A cítala espartana.**

   Recorde-se que cifra de permutação definida por uma cítala espartana é uma cifra de transposição, definida colocando o texto limpo numa matriz com colunas com uma altura determinada (dependente da espessura da cítala). O texto cifrado é o obtido a partir da leitura vertical das colunas.

   (a) Dado o texto limpo

   abatalhacomospersasteralugarnodesfiladeirodastermopilas

   i. obtenha o texto cifrado usando uma cítala espartana com espessura de 4 caracteres;

   ii. obtenha o texto cifrado usando uma cítala espartana com espessura de 6 caracteres.

   (b) [**Implementação**] Implemente uma função `citala_espartana` que permita realizar a cifra de permutação obtida com uma cítala espartana de diferentes espessuras. Mais precisamente, `citala_espartana` deverá ser uma função com:

   - Argumentos : `texto_claro` (sequência de caracteres minúsculas, do alfabeto standard, sem espaços), `n` (número positivo maior que 2, será o parâmetro que depende da *espessura* da cítala);
   - Retorno: `texto_cifrado` (sequência de caracteres maiúsculas, do alfabeto standard, sem espaços, obtido a partir do `texto_claro` com cifra da Cítala Espartana cuja espessura determina uma coluna de altura `n`)

2. **A cifra de Belaso-Vigènere.**

   Recorde-se que a cifra de sustituição poli-alfabética de Belaso-Vigenère usa uma palavra chave para trocar entre os alfabetos de deslocação definidos pela *Tabua Recta*.

   (a) Dado o texto limpo

   primeiracifrapolialfabeticacomtrocadechave

   obtenha o texto cifrado usando a cifra de Belaso-Vigenère e como chave ZAR

(b) [**Implementação**] Implemente uma função `belaso_vigenere` que permita realizar a cifra polialfabética de Vigenère-Belaso. Mais precisamente, `belaso_vigenere` deverá ser uma função tal que:

- Argumentos: `texto_claro` (sequência de caracteres minúsculas, do alfabeto standard, sem espaços) e `palavra_chave`(sequência de caracteres maiúsculos, do alfabeto standard, sem espaços);
- Retorno: `texto_cifrado` (sequência de caracteres maiúsculos, alfabeto standar, sem espaços, obtido a partir do `texto_claro` com cifra de Belaso-Vigenère com palavra chave o argumento `palavra_chave`).

3. **Cifra de Vernam e geração de *keystream* por LFSR**

Recorde-se que a Cifra de Vernam é uma cifra stream que realiza um XOR do texto claro (em bits) com a *keystream* correspondente e que os LFSR são mecanismos de geração de chaves usando relações de recorrência.

(a) Considere a LFSR de comprimento 3 definida pela relação de recorrência:

$$k_i = k_{i-2} + k_{i-3}$$

e o texto claro

$$010101010.$$

Dados os valores iniciais $k_0 = 0$, $k_1 = 1$, $k_2 = 1$ para o LFSR e determine os primeiros 9 termos da *keystream* gerados. Use esta *keystream* para cifrar, com a cifra de Vernam, o texto claro anterior.

(b) [**Implementação**] Implemente uma função `gerador_lfsr` que, a partir de uma relação de recorrência e dos dados inciais, gera uma *keystream* de determinado comprimento. Mais precisamente, `gerador_lfsr` deverá ser uma função com:

- Argumentos: `relacao` (sequência bits de comprimento), `sequencia_inicial` (sequência de bits com o mesmo comprimento) e `length_key` (tamanho da key stream desejada)
- Retorno: `key_stream` (sequência de bits de comprimento `length_key`)

Note que uma relação de recorrência, em bits, que relacione um elemento r com os r bits anteriores pode representar-se efetivamente através de uma sequência de r bits. Por exemplo:

$$k_i = k_{i-1} + k_{i-2} \leftrightarrow (11) \quad k_i = k_{i-2} + k_{i-3} \leftrightarrow (011) \quad k_i = k_{i-2} + k_{i-4} + k_{i-5} \leftrightarrow (01011)$$

4. **Auto-chave de Vigenère**

Recorde-se que a cifra de auto-chave de Vigenère é uma cifra *stream* baseada na *Tabula Recta* que utiliza um caracter como chave inicial e depois cifra usando consecutivamente os caracteres do próprio texto limpo.

(a) Cifre o texto em claro

aideiamaisbrilhantedevigenere

usando a cifra de auto-chave de Vigenere com chave inicial B.

(b) [**Implementação**] Implemente uma função `vigenere_autokey` que permita realizar a cifra *stream* com auto-chave de Vigenère. Mais precisamente, `vigenere_autokey` deverá ser uma função com:

- Argumentos: `texto_claro` (sequência de caracteres) e `chave_inicial` (um caracter standard, maiúscula)
- Retorno: `texto_cifrado` (sequência de caracteres obtida pela cifra com auto chave de Vigenère e chave inicial `chave_inicial`)

5. **Comparação de cifras de substituição em *bytes***

   Consideremos as seguintes cifras de substituição em blocos de 8-bits (bytes):

   - A cifra definida pelo XOR-bitwise, denotada por $\oplus$;
   - A cifra definida pela adição módulo 2^2, nos sub-blocos de 2 bits, denotada por $\boxplus_2$;
   - A cifra definida pela adição módulo 2^4, nos sub-blocos de 4 bits, denotada por $\boxplus_4$;
   - A cifra definida pela adição módulo 2^8, denotada por $\boxplus_8$.

   (a) Dado o texto claro
   $$m = 11111110$$
   e a chave $k = 11010101$ indique:

   i. O texto cifrado obtido usando o XOR-bitwise e a chave k;
   ii. O texto cifrado obtido usando o $\boxplus_2$ e a chave k;
   iii. O texto cifrado obtido usando o $\boxplus_4$ e a chave k;
   iv. O texto cifrado obtido usando o $\boxplus_8$ e a chave k

   (b) [**Implementação**] Implemente uma função `cifras_sustituicao_bytes` que permita realizar, a partir de uma chave K, as quatro sustituições anteriores. Mais precisamente, a função deverá ter:

   - Argumentos: `byte` (sequência de 8 bits) e `n` (com $n = 1, 2, 4, 8$, que indica a operação a realizar, XOR, $\boxplus_2$,$\boxplus_4$ e $\boxplus_8$ respectivamente)
   - Retorno: `cifra_byte`(sequência de 8 bits, obtida pela sustituição definida pelo `n`).

6. **Paddings - blocos de n bits**

   Os métodos de preenchimento, *paddings*, para blocos de n bits mais usados são o **OneAndZeroes** e o **Trailing Bit Complement**.

   (a) Considere os array de bits
   $$\begin{aligned} \text{m} &= 00101111101 \\ \text{m'} &= 01011111 \end{aligned}$$

   i. Realize o *padding* dos arrays m e m' usando o OneAndZeroes para blocos de 8-bits;
   ii. Realize o *padding* do arrays m e m' usando o TrailingBitComplement para blocos de 8-bits.

   Dado o array de 16 bits
   $$m'' = 0000111100001111,$$
   indique:

   i. O array original se m'' for o resultado de um **OneAndZeroes** para blocos de 4 bits;
   ii. O array original se m'' for o resultado de um **TrailingBitComplement** para blocos de 4-bits.

   (b) [**Implementação**] Implemente quatro funções, `padd_oneandzeroes`, `padd_trailingbitcom`, `unpadd_oneandzeroes` e `unpadd_trailingbitcom`, que permitam realizar os *paddings* e *unpaddings* OneAndZeroes e TrailingBitComplement. Mais precisamente, cada uma das funções deverá ter:

   - Argumentos: `sequencia_bits` (sequência de bits) e `n_bits` (comprimento n de cada bloco do *padding*)
   - Retorno: `sequencia_pad`(sequência de bits, com o preenchimento completo).

7. **Cifra de Hill, módulo 2, e modos de operação ECB e CBC**

   A cifra de Hill, módulo 2, por blocos de comprimento n, cifra um texto claro no alfabeto $\{0, 1\}$ (bits) multiplicando os blocos de comprimento n por uma matriz quadrada de ordem n, invertível módulo 2.

(a) Considere o texto claro 01001101 e a matriz chave

$$K = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

i. Cifre o texto em claro, usando uma cifra de Hill de 2-blocos, com o modo de operação ECB, e a matriz chave K.

ii. Cifre o mesmo texto em claro usando a mesma matriz chave mas o modo de operação CBC, com bloco inicial $IV = 01$.

(b) [**Implementação**] Implemente uma função `cifra_hill_mod2` que permita realizar a cifra de Hill por blocos de n-bits, no modo CBC, de um texto em claro de r-bits (com r múltiplo de n), a partir de uma matriz chave K. Mais precisamente, `cifra_hill_mod2` deverá ser uma função com:

- Argumentos: `n` (comprimento de cada bloco), `matriz_chave`, `texto_limpo` (sequência de bits com comprimento múltiplo de n), `bloco_inicial`;
- Retorno: `texto_cifrado` (sequência de bits, com comprimento múltiplo de `n`)

8. **Cifra de Hill, módulo 16 (hexadecimal)**

A cifra de Hill, módulo 16, por blocos de comprimento n, cifra um texto claro em $\mathbf{Z}_{16}$ (alfabeto hexadecimal) multiplicando os blocos de comprimento n por uma matriz quadrada de ordem n, invertível módulo 16.

(a) Considere o texto claro $AB08F12C$ e a matriz chave

$$K = \begin{bmatrix} 3 & F \\ 0 & 1 \end{bmatrix}.$$

i. Cifre o texto em claro, usando uma cifra de Hill de 2-blocos e matriz chave K.

ii. Verifique que a matriz inversa de K, módulo 16, é $K = \begin{bmatrix} B & B \\ 0 & 1 \end{bmatrix}$.

(b) [**Implementação**] Implemente uma função `cifra_hill_mod16` que permita realizar a cifra de Hill por blocos de n-bits, módulo 16, de um texto em claro de r-bits (com r múltiplo de n), a partir de uma matriz chave K. Mais precisamente, `cifra_hill_mod16` deverá ser uma função com:

- Argumentos: `n` (comprimento de cada bloco), `matriz_chave`, `texto_limpo` (sequência de hexadecimais, com comprimento múltiplo de `n`);
- Retorno: `texto_cifrado` (sequência de hexadecimais com comprimento múltiplo de `n`)

9. **Uma cifra round (hexadecimal)**

Considere a cifra *round* por blocos de 4-bytes definida pela composição das cifras:

[C1] a cifra por blocos de 4 bytes definida pela troca de posição dos sub-blocos de 2 bytes esquerdo e direito;

[C2] a cifra de substituição que consiste em identificar um byte com um par de elementos (x, y) de Z_{16} e realizar a transformação $(x, y) \to (x, x + y)$, em $\mathbf{Z}_{16}$.

Por exemplo, o byte $A9$ identifica-se com $(10, 9)$ e a cifra substitui este byte por $(10, 10 + 9) \equiv (10, 3)$ em Z_{16}, ou seja, por $A3$.

(a) Determine o texto cifrado com esta cifra *round*, a partir do texto em claro

$$AA\,03\,FA\,7B \quad E3\,2C\,00\,11$$

(b) [**Implementação**] Implemente uma função `cifra_c1_c2` que permita realizar esta cifra *round*. Mais precisamente, `cifra_c1_c2` deverá ser uma função com:

- Argumento: `texto_claro` (sequência de bytes, com comprimento múltiplo de 4),
- Retorno: `texto_cifrado` (sequência de bytes, com comprimento múltiplo de 4, obtida pela cifra round C_1C_2)

10. **Modo de operação CTR e PKCS7 *padding***

Considere-se a cifra por blocos de dois bytes (notação hexadecimal) que consiste em realizar um XOR-bitwise com a chave K, sendo K também um bloco de dois bytes.

(a) Cifre, usando um modo de operação CTR, com *padding PKCS7*, o texto claro

$$AA\,03 \quad FA\,7B \quad E3\,2C$$

considerando como chave $K = FF\,00$ e como bloco inicial $A0\,10$.

(b) [**Implementação**] Implemente uma função `cifra_CTR` que permita realizar esta cifra por blocos de 2-bytes (modo CTR, PKCS7 *padding*). Mais precisamente, `cifra_blocos_2bytes` deverá ser uma função com:

- Argumentos: `texto_claro` , `chave`, `bloco_inicial` (sequência de bytes, com comprimento arbitrário, chave e bloco inicial com comprimento 2 bytes),
- Retorno: `texto_cifrado` (sequência de bytes, com comprimento múltiplo de 2, obtida após a realização do *padding* e do cifrado modo CTR)

11. **O algoritmo extendido de Euclides**

Recorde-se que o algoritmo de Euclides permite, dados dois números inteiros a, b, encontrar o seu máximo comum divisor $d = \gcd(a, b)$ e coeficientes de Bezout, isto é, os inteiros x, y que verificam $ax + by = d$

(a) Determine, usando o algoritmo de Euclides e indicando todos os passos, o $d = \gcd(26, 7)$ e os coeficientes x, y que verificam $d = 26x + 7y$. Deduza o inverso módulo 26 de 7.

(b) [**Implementacao**] Implemente numa função `algoritmo_euclides_ext`, o Algoritmo de Euclides extendido. Mais precisamente, `algoritmo_euclides_ext` deverá ser uma função com:

- Argumentos: `a, b` (dois inteiros)
- Retorno: `d, x, y` (inteiros, máximo comum divisor e coeficientes de Bezout).

Nota: A descrição deste algoritmo em pseudo-código encontra-se detalhada em 2.107 de [1]

12. **Potências, inversos módulo n e função de Euler**

Recorde-se que a^k está definido para todo o $a \in \mathbf{Z}_n$, se $k > 0$, e para a invertível se $k \leq 0$ e que $\phi(n)$ denota a função de Euler de um inteiro positivo n. **Sem apoio computacional:**

(a) Calcule usando o método de quadrados repetidos, as seguintes potências:

i. 7^{50} em $\mathbf{Z}_{15}$; ii. 8^{2020} em $\mathbf{Z}_9$; iii. 10^8 em $\mathbf{Z}_{35}$.

(b) Determine $\phi(n)$ para $n = 15, 9, 35$. Deduza o número de elementos invertíveis em $\mathbf{Z}_{15}$, $\mathbf{Z}_9$ e $\mathbf{Z}_{35}$.

(c) Determine, usando o teorema de Euler, os inversos modulares:

i. 7^{-1} em $\mathbf{Z}_{15}$;
ii. 8^{-1} em $\mathbf{Z}_9$;
iii. 34^{-1} em $\mathbf{Z}_{35}$;
iv. 7^{-50} em $\mathbf{Z}_{15}$;
v. 8^{-2019} em $\mathbf{Z}_9$

(d) [**Implementação**] Encontre, na sua linguagem de programação preferida, uma biblioteca que permita realizar todas as operações modulares, e implemente um script para calcular as inversas de matrizes com coeficientes módulo n.

13. **A cifra RSA**

(a) Considere os primos $p = 5$ e $q = 7$ e o módulo $n = 35$. Sem apoio computacional,

i. determine uma chave pública e uma chave privada para a cifra RSA a partir de p,q e n indicados.

ii. cifre o texto claro $x = 10$ com a chave pública obtida e verifique, a partir do texto cifrado, que a chave privada decifra adequadamente.

(b) [**Implementação**] Implemente a função `keys_rsa` com as seguintes especificações:

- Argumentos: dois números primos `p,q`
- Retorno: chaves pública `k=(n,e)` e privada `K=(n,d)` necessárias na cifra RSA.

O *script* deve incluir testes de cifrado e de decifrado RSA com as verificações da alínea anterior. Note-se que, no caso do Pyhton3, a cifra a partir dos parâmetros e do texto claro, consiste simplesmente em usar a função *built-in* `pow(x,e,n)`, para calcular $x^e \bmod n$ e $y^d \bmod n$.

14. **Protocolo de Diffie-Hellman**

O protocolo de intercâmbio de chaves de Diffie e Hellman permite estabelecer uma chave secreta entre duas entidades através de um canal aberto.

(a) Considere o primo $p = 13$ e $\alpha = 6$. Suponha que as chaves secretas de Alice e Bob são respetivamente $x = 4$ e $y = 2$. Determine a chave partilhada K de Alice e Bob calculada através do protocolo DH.

(b) [**Implementação**] Implemente uma função `chave_partilhada_DH` com as seguintes especificações:

- Argumentos: `p` (um número primo), `alpha` (um gerador multiplicativo de $\mathbf{Z}_p^*$);
- Retorno: `K` (chave partilhada), `x`, `y` (chaves privadas de Alice e Bob).

15. **Cifra ElGamal**

(a) Considere o primo $p = 17$. Sem apoio computacional,

i. Verifique que 10 é um gerador multiplicativo de $\mathbf{Z}_{17}^*$.

ii. Determine uma chave pública e uma chave privada para a cifra ElGamal com $p = 17$ e $\alpha = 10$.

iii. Cifre o texto claro $x = 4$ com a chave pública obtida e verifique, a partir do texto cifrado, que a chave privada decifra adequadamente.

(b) [**Implementação**] Implemente duas funções `cifra_elgamal` e `decifra_elgamal` com as seguintes especificações:

- A função `cifra_elgamal` tem como argumentos um texto claro `x` e uma chave pública `(p,alpha,beta)` e como retorno o texto cifrado `(y,z)` com a cifra ElGamal;
- A função `decifra_elgamal` tem como argumentos um texto cifrado `(y,z)` e uma chave privada ElGamal `(p,a)` e como retorno o texto claro decifrado com a chave privada.

O *script* deve incluir testes de cifrado e de decifrado ElGamal e as verificações da alínea anterior.

16. **Construção de Merkle-Damgård**

Considere a função de compressão que transforma blocos de 16 bits em blocos de 4 bits adicionando todos os sub-blocos de 4-bits módulo $\mathbf{Z}_{2^4}$:

$$f(B_1 B_2 B_3 B_4) = B_1 \boxplus_4 B_2 \boxplus_4 B_3 \boxplus_4 B_4$$

(a) Determine o *hash* da mensagem

$$m = 010101010101010000$$

usando a contrução de Merkle-Damgard a partir da função f.

(b) [**Implementação**] Implemente uma função `hash_MD_funcaof` com as seguintes especificações:

- Argumentos: `m`, (sequência de bits de comprimento arbitrário) e `H0` (sequência de 4 bits, *hash* inicial);
- Retorno: `H` (sequência de bits de comprimento 4, *hash* da mensagem `m`).

17. **Esquema de Davies-Meyer**

Considere a função de compressão que transforma blocos de 16 bits em blocos de 4 bits adicionando todos os sub-blocos de 4-bits módulo $\mathbf{Z}_{2^4}$:

$$f(B_1 B_2 B_3 B_4) = B_1 \boxplus_4 B_2 \boxplus_4 B_3 \boxplus_4 B_4$$

(a) Determine o *hash* da mensagem

$$m = 010101010101010000$$

usando o esquema de Davies-Meyer a partir da função f.

(b) [**Implementação**] Implemente uma função `hash_DM_funcaof` com as seguintes especificações:

- Argumentos: `m`, (sequência de bits de comprimento arbitrário) e `H0` (sequência de 4 bits, *hash* inicial);
- Retorno: `H` (sequência de bits de comprimento 4, *hash* da mensagem `m`).

18. **A curva elíptica** $y^2 \equiv x^3 - 7x + 10 (\mathbf{mod}\, p)$

Recorde-se que os algoritmos de cifrado baseados no logaritmo discreto podem ser definidos usando a estrutura de grupo de uma curva elíptica.

- Considere o primo $p = 19$, verifique que os pontos

$$A(7, 0) \qquad \text{e} \qquad B(1, 2)$$

pertencem a esta curva elíptica e calcule $2A$ e $A + B$.

- [**Implementação**] Implemente uma função `potencia_curva` com as seguintes especificações:
  - Argumentos: `A,B`, (dois pontos com coordenadas em $\mathbf{Z}_{19}$);
  - Retorno: `A+B` (coordenadas do ponto soma, considerando a operação na curva elíptica).

Referências

[1] A. Meneses, P. van Oorschot, S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.