Pràctica 3: Criptografia de clau pública

Funcions útils per a realitzar la pràctica:

- is_prime (n)
- random_prime (n, proof=None, lbound=2)
- randint (a, b)
- power_mod (a, n, m)
- inverse_mod (a, m)
- multiplicative_order ()
  Si $a \in \mathbb{Z}_n$ és primitiu, aleshores `a.multiplicative_order()` $= \phi(n)$
- primitive_root (n)

1. Intercanvi de claus Diffie-Hellman

   (a) Implementeu la funció `UAB_check_Diffie_Hellman(p,alpha,a,b,k_pubA,k_pubB)` que validi els paràmetres de l'algorisme d'intercanvi de claus Diffie-Hellman i generi la clau secreta compartida. Els paràmetres que rep aquesta funció són:

   - `p`: nombre primer
   - `alpha`: nombre primitiu
   - `a,b`: claus privades d'A i B
   - `k_pubA,k_pubB`: claus públiques d'A i B

   Com a resultat de la validació, aquesta funció generarà els següents missatges:

   - `Invalid p`
   - `Invalid alpha`
   - `Invalid private keys`
   - `Invalid public keys`
   - `All correct`

   Tingueu en compte que caldrà generar més d'un missatge si hi ha més d'un paràmetre incorrecte.

   El retorn d'aquesta funció serà la clau secreta compartida entre A i B si tots els paràmetres són correctes. Retornarà `None` en cas contrari.

   (b) Implementeu la funció `UAB_find_discrete_log(p,alpha,k_pubA)` que calculi el logaritme de `k_pubA` en base `alpha` en el conjunt dels enters mòdul `p`, la qual cosa ens permet obtenir la clau privada `a` utilitzada per generar una clau pública `k_pubA`. La cerca d'aquesta clau privada l'heu de fer per força bruta. Els paràmetres que rep aquesta funció són:

   - `p`: nombre primer
   - `alpha`: nombre primitiu
   - `k_pubA`: clau pública

(c) Utilitzeu la funció `UAB_find_discrete_log(p,alpha,k_pubA)` per estimar (en nombre d'anys) el temps que trigaria la vostra màquina a trobar la clau privada corresponent a una clau pública calculada a partir d'un nombre primer `p` de 1024 bits. Per realitzar aquesta estimació, realitzeu diferents proves utilitzant nombres primers més petits. Amb el resultat d'aquestes proves, ompliu la següent taula en la qual heu d'indicar la mida del nombre primer `p` i el temps que ha trigat l'execució de mitjana.

Mida de `p` (bits)	Mitjana temps (segons)
8	...
12	...
16	...
20	...
24	...

Realitzeu com a mínim 3 proves per a cada nombre de bits. En cadascuna de les proves, tant `p` com la clau privada `a` hauran de ser nombres aleatoris. La mesura del temps d'execució de la funció `UAB_find_discrete_log` es pot realitzar utilitzant `%time` de la manera següent:

```
%time private_key=UAB_find_discrete_log(p,alpha,k_pubA)
```

Després de cada test, comproveu que la clau obtinguda amb aquesta funció coincideix amb `a`. En aquest cas, mostreu el següent missatge: "`Correct private key`".

Tant la taula com el temps estimat han de constar en el vostre notebook en forma de comentari.

2. Xifrat amb RSA

Imaginem que un professor vol enviar les notes dels seus estudiants a un company seu utilitzant la seva clau pública (n, e). Per protegir la informació, utilitza xifratge RSA i codifica el NIU i la nota de la següent manera: `m = 10*NIU + nota`, on *nota* és un enter de 0 a 9. Per reduir el temps d'execució, suposarem que els NIUs van del 1600000 al 1700000. Per tant, els possibles valors de `m` aniran del 16000000 al 17000009.

(a) Implementeu la funció `UAB_gen_RSA_NIU_nota_table(n,e,NIU_nota_min,NIU_nota_max)` que generi tots els possibles `c`=RSA(NIU|nota). Aquests valors es guardaran en un *dictionary* (`c` $\rightarrow$ `m`) que permetrà obtenir el missatge `m` (NIU|nota) a partir del corresponent xifrat `c`.

(b) Implementeu la funció `UAB_RSA_decrypt(n,c,d)` que desxifri un missatge xifrat `c` amb la clau privada `d`. El primer paràmetre `n` és el mòdul.

(c) Implementeu la funció `UAB_RSA_table_tests(z,NIU_nota_min,NIU_nota_max)` que compari `z` vegades el resultat obtingut de desxifrar `c` a través de la taula anterior amb el resultat de desxifrar utilitzant la clau privada. Per a fer-ho, seguiu els passos següents:

i. Genereu la clau pública (n, e) i la corresponent clau privada d. Per al càlcul de n, utilitzeu valors aleatoris p i q de **32 bits**.

ii. Creeu la taula de valors `c` $\rightarrow$ `m` a través de la funció `UAB_gen_RSA_NIU_nota_table`.

iii. Genereu `z` missatges aleatoris `m` i imprimiu per pantalla la següent informació per a cadascun d'aquests missatges:

`m` | `c` | `m_table` | `m_decrypt` | `test`

on `m_table` és el valor de `m` obtingut a través de la taula, i `m_decrypt` és el valor obtingut desxifrant `c` amb la funció `UAB_RSA_decrypt`. El camp `test` és el resultat de comparar `m_table` i `m_decrypt`, el qual serà sempre `True` si la vostra implementació està ben feta.

3. Signatures amb ElGamal

L'algorisme de signatura d'ElGamal és susceptible a atacs de falsificació existencial.

En efecte, donada una clau pública $k_{pub} = (p, \alpha, \beta)$, un atacant pot procedir de la següent manera:

1. Tria dos enters i i j tals que $mcd(j, p-1) = 1$
2. Calcula la signatura:

$$r = \alpha^i \beta^j \mod p$$
$$s = -rj^{-1} \mod p-1$$

3. Calcula el missatge:

$$m = si \mod p-1$$

4. L'atacant obté una signatura vàlida (r, s) per al missatge m
5. La signatura és vàlida, ja que la igualtat $\beta^r r^s = \alpha^m \mod p$ es manté.

Creeu les següents funcions per generar i validar signatures creades utilitzant l'atac de falsificació existencial de signatures ElGamal.

(a) Implementeu la funció `UAB_ElGamal_verify(r,s,p,alpha,beta,m)` que verifiqui que `(r,s)` és una signatura vàlida de `m` utilitzant la clau pública `(p,alpha,beta)`. La funció ha de retornar `True` o `False`.

(b) Implementeu la funció `UAB_ElGamal_sign_existential_forgery(p,alpha,beta,i,j)` que, a partir de la clau pública i els paràmetres `i,j`, retorni una signatura `(r,s)` i el missatge `m`.

Com a resultat, aquesta funció retornarà la tupla `(r,s,m)`. Si $mcd(j, p-1) \neq 1$, aleshores retornarà `None`.

(c) Implementeu la funció `UAB_ElGamal_sign_exist_forgery_test(z)` que comprovi `z` vegades que les signatures generades amb la funció anterior són vàlides. A cada iteració, caldrà imprimir per pantalla la informació següent:

```
p | alpha | beta | r | s | m | test
```

Per a fer-ho, seguiu els següents passos:

i. Genereu la clau pública $(p, alpha, beta)$ i la corresponent clau privada d. Per al càlcul de p, utilitzeu un primer aleatori de **16 bits**.

ii. Calculeu el missatge `m` i la seva signatura `(r,s)` utilitzant valors aleatoris per a `i,j`.

iii. Comproveu que la signatura anterior és vàlida amb la funció `UAB_ElGamal_verify`. El resultat d'aquesta comprovació serà el valor `test` que mostrareu per pantalla. Si la vostra implementació és correcta, `test` serà `True` per a totes les `z` iteracions.