Universidad Nacional Autónoma de México
Facultad de Ciencias

Criptografía y Seguridad
Semestre: 2025-1

Tarea 3

García Ponce José Camilo 319210536

Equipo Pingüicoders
Arrieta Mancera Luis Sebastian
Cruz Cruz Alan Josué
García Ponce José Camilo
Matute Cantón Sara Lorena

1. El siguiente texto está cifrado mediante RSA.
   138527, 171279, 138664, 242409, 103298, 171279, 27261, 103786, 0, 103298, 0, 103298, 242409, 224525, 188808, 171279, 27261
   Los parámetros públicos son $n = 256961$ y $e = 59029$. Calcula la llave privada d y descifra el mensaje.

Primero para calcular d necesitamos resolver esto $d * e \equiv 1 \, mod \, \varphi(n)$, entonces sustituimos los valores que ya conocemos $d * 59029 \equiv 1 \, mod \, \varphi(256961)$, ahora recordando las clases sabemos que $\varphi(pq) = \varphi(p) * \varphi(q)$, con p y q primos, por lo tanto usamos la tarea anterior para saber que la factorización en primos de 256961 es $293 * 877 = 256961$, entonces $\varphi(256961) = \varphi(293) * \varphi(877)$, y como 293 y 877 son primos entonces $\varphi(293) = 292$ y $\varphi(877) = 876$, entonces $\varphi(256961) = \varphi(293) * \varphi(877) = 292 * 876 = 255792$. Entonces para calcular $d * 59029 \equiv 1 \, mod \, 255792$ usamos el siguiente código (todo el código de este ejercicio está en ejercicio1.py), lo cual nos da que $d = 13$.

```python
phi= (292)*(876)
#print(phi) #255792


def d(e,phi):
    """
    Fucnion que encuentra el valor de D siguiendo la siguiente formula
    d*e=1 mod phi(n)
    No es el mas eficiente pero es el mas sencillo de implementar,
    con número muy grandes puede tardar mucho tiempo en encontrar el valor de d
    """
    d=0
    while True:
        if (d*e)%phi==1: #Vamos revisando si d*e cumple con la condicion
            break
        d+=1 # Si no cumple, aumentamos el valor de d
    return d

print(d(e,phi)) #13
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$ /bin/python /home/camilo/CC/septimoSeme
13
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$
```

Ahora para descifrar el mensaje usamos que el mensaje original M se obtiene con $M = C^d \, mod \, n$ (con C el mensaje cifrado), por lo tanto usamos $M = C^{13} \, mod \, 256961$, por lo tanto usamos el siguiente código para descifrar el mensaje.

```python
def decipher(c,d,n):
    """
    Funcion que desencripta un mensaje cifrado en RSA
    """
    return (c**d)%n

def decipherWholeMessage(cipheredMessage,d,n):
    """
    Funcion que desencripta un mensaje completo
    """
    return [decipher(c,d,n) for c in cipheredMessage]
```

```python
alfabeto = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

def numberTostring(numbers):
    """
    Funcion que convierte una lista de numeros en una cadena de texto
    """
    return "".join([alfabeto[n] for n in numbers])
```

Y ejecutando todo obtuvimos

```python
message = [ 138527, 171279, 138664, 242409, 103298, 171279, 27261
,103786, 0, 103298, 0, 103298, 242409, 224525, 188808, 171279, 27261
]

print(numberTostring(decipherWholeMessage(message,13,256961)))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$ /bin/python /home/camilo/CC/
FELICESVACACIONES
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$
```

Obteniendo que el mensaje es “felicesvacaciones” o “FELICESVACACIONES”

Fuentes usadas para este ejercicio:
Clases de teoría
Ayudantías

2. Considera la curva elíptica E_p con parámetros $a = 1, b = 17, p = 23$.

Para este ejercicio usaremos los archivos generados de hacer la practica 10 (EllipticCurve.py, Point.py y tools.py) y para los ejercicios usaremos el codigo en curvas.py

   a. Calcula y exhibe todos los puntos de la curva.

Para esto generamos un objeto EllipticCurve con los parámetro $p = 23$, $a = 1$ y $b = 17$. Y usamos points para obtener los puntos de la curva. Notemos que None representa el punto al infinito.

```
# Generamos la curva eliptica
p = 23
a = 1
b = 17
eli = EllipticCurve(p, a, b)

# Mostramos los puntos de la curva
for punto in eli.points:
    print(punto)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMEN
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$ /b
None
(2, 2)
(2, 21)
(3, 1)
(3, 22)
(4, 4)
(4, 19)
(5, 3)
(5, 20)
(6, 3)
(6, 20)
(8, 10)
(8, 13)
(11, 5)
(11, 18)
(12, 3)
(12, 20)
(15, 7)
(15, 16)
(16, 9)
(16, 14)
(17, 5)
(17, 18)
(18, 5)
(18, 18)
(19, 8)
(19, 15)
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$
```

b. Especifica una codificación para cada letra del alfabeto.
Para esto usaremos el alfabeto "abcdefghijklmnopqrstuvwxyz ", usaremos la función table() para generar la codificación de letras a puntos.

```python
# Alfabeto
alfabeto = "abcdefghijklmnopqrstuvwxyz "

# Generamos la tabla de codificacion
tabla = table(eli, alfabeto)

# Mostramos la tabla de codificacion
for key, value in tabla.items():
    print(f"\"{key}\" -> {value}")

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENT
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$ /bi
"a" -> None
"b" -> (2, 2)
"c" -> (2, 21)
"d" -> (3, 1)
"e" -> (3, 22)
"f" -> (4, 4)
"g" -> (4, 19)
"h" -> (5, 3)
"i" -> (5, 20)
"j" -> (6, 3)
"k" -> (6, 20)
"l" -> (8, 10)
"m" -> (8, 13)
"n" -> (11, 5)
"o" -> (11, 18)
"p" -> (12, 3)
"q" -> (12, 20)
"r" -> (15, 7)
"s" -> (15, 16)
"t" -> (16, 9)
"u" -> (16, 14)
"v" -> (17, 5)
"w" -> (17, 18)
"x" -> (18, 5)
"y" -> (18, 18)
"z" -> (19, 8)
" " -> (19, 15)
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$
```

c. Cifra el mensaje ”quiero vacaciones”.
Para esto hacemos el intercambio de llaves de la manera vista en clase. Usamos el siguiente código.

```python
# Punto base G
g = eli.points[-1]

# Alicia llave privada n_a
n_a = 2

# Alicia llave publica P_a
p_a = eli.mult(n_a, g)

# Bob llave privada n_b
n_b = 3

# Bob llave publica P_b
p_b = eli.mult(n_b, g)

# Alicia calcula k
k_a = eli.mult(n_a, p_b)

# Bob calcula k
k_b = eli.mult(n_b, p_a)
```

Y para poder cifrar el mensaje usamos las siguientes funciones.

```python
# Funcion para cifrar una letra
def cifrar_letra(letra, g, p_x, curva):
    # Convertimos la letra a un punto
    p_m = letra_a_punto(letra)
    # Escogemos un k entero
    k = random.randint(1, curva.order(g) - 1)
    # Calculamos el par de puntos (kG, P_m + kP_a)
    punto1 = curva.mult(k, g)
    kp_a = curva.mult(k, p_x)
    punto2 = curva.sum(p_m, kp_a)
    # Regresamos el par de puntos
    return punto1, punto2

# Funcion para cifrar un mensaje
def cifrar_mensaje(mensaje, g, p_a, curva):
    # Inicializamos el mensaje cifrado
    mensaje_cifrado = ""
    # Ciframos cada letra del mensaje
    for letra in mensaje:
        punto1, punto2 = cifrar_letra(letra, g, p_a, curva)
        mensaje_cifrado += punto_a_letra(punto1) + punto_a_letra(punto2)
    return mensaje_cifrado
```

Por último ciframos el mensaje.

```
113  # Mostramos el mensaje
114  print(f"Mensaje: {mensaje}")
115
116  # Ciframos el mensaje
117  mensaje_cifrado = cifrar_mensaje(mensaje, g, p_a, eli)
118
119  # Mostramos el mensaje cifrado
120  print(f"Mensaje cifrado: {mensaje_cifrado}")
121
122  # Desciframos el mensaje
123  mensaje_descifrado = descifrar_mensaje(mensaje_cifrado, n_a, eli)
124
125  # Mostramos el mensaje descifrado
126  print(f"Mensaje descifrado: {mensaje_descifrado}")
127
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$ /bin/python /home/cami
Mensaje: quiero vacaciones
Mensaje cifrado: xlrjulvurglfiuyxtypllukpmdphrmfcnl
Mensaje descifrado: quiero vacaciones
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$
```

Mensaje: quiero vacaciones
Mensaje cifrado: xlrjulvurglfiuyxtypllukpmdphrmfcnl
Mensaje descifrado: quiero vacaciones
Nota, algunas veces el descifrado no funciona bien, esto debido a los números aleatorios escogidos al cifrar, que puede causar que algunos puntos se vuelvan el punto al infinito al multiplicar.

Fuentes usadas para este ejercicio:
Clases de teoría
Practica 10

3. Supongamos que Alice quiere enviar a Bob la palabra secreta ”lunes”. Explica mediante un ejemplo la manera de compartir la palabra secreta por un canal inseguro. Debes utilizar:
    a. El protocolo de Diffie Hellman con curvas elípticas.
    b. Tu implementación de DES.

Todo el código usado para esta práctica será el de la práctica 10 (EllipticCurve.py, Point.py y tools.py), la tarea anterior (des.py) y el archivo ejercicio3.py
Veamos cómo será el ejemplo. Primero Alice y Bob van a realizar el protocolo de Diffie Hellman con curvas elípticas para acordar en un punto secreto.

Primero Alice y Bob eligen un entero positivo p tal que p sea primo o $p = 2^m$, para diremos que eligen $p = 23$ (este número es primo).

Después veremos qué curva elíptica van a usar, digamos que usaran $a = 1$ y $b = 7$, por lo tanto la curva elíptica usada será $E_{23}(1, 7)$, es decir la curva $y^2 = x^3 + 1x + 17\ mod\ 23$ (la curva es pública).
Notemos que la curva tiene los siguientes puntos, por lo tanto podemos usar el alfabeto “abcdefghijklmnopqrstuvwxyz ”.

```
Los puntos de la curva son los siguientes:

[None, (2, 2), (2, 21), (3, 1), (3, 22), (4, 4), (4, 19), (5, 3), (5, 20), (6, 3), (6, 20), (8, 10), (8, 13), (11, 5), (11, 18), (12, 3),
 (12, 20), (15, 7), (15, 16), (16, 9), (16, 14), (17, 5), (17, 18), (18, 5), (18, 18), (19, 8), (19, 15)]
```

Luego Alice y Bob se tiene que poner de acuerdo en un punto G para usar, diremos que será el punto $G = (19, 15)$, notemos que este si es un punto de la curva (en particular es el último).
Posteriormente Alice y Bob eligen numeros aleatorias que serán sus llaves privadas, n_a para Alice y n_b para Bob, digamos que $n_a = 2$ y $n_b = 3$.
Después Alice y Bob van a generar sus llaves públicas, P_a para Alice y P_b para Bob, este punto privado lo calculan de la siguiente manera $P_a = n_a \times G$ y $P_b = n_b \times G$, por lo cual $P_a = (11, 18)$ y $P_b = (5, 20)$.

```
84   # Alicia llave publica
85   p_a = eli.mult(n_a,g)
86   # Bob llave publica
87   p_b = eli.mult(n_b,g)
88
89   print(p_a)
90   print(p_b)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMEN

camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$ /b
(11, 18)
(5, 20)
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$
```

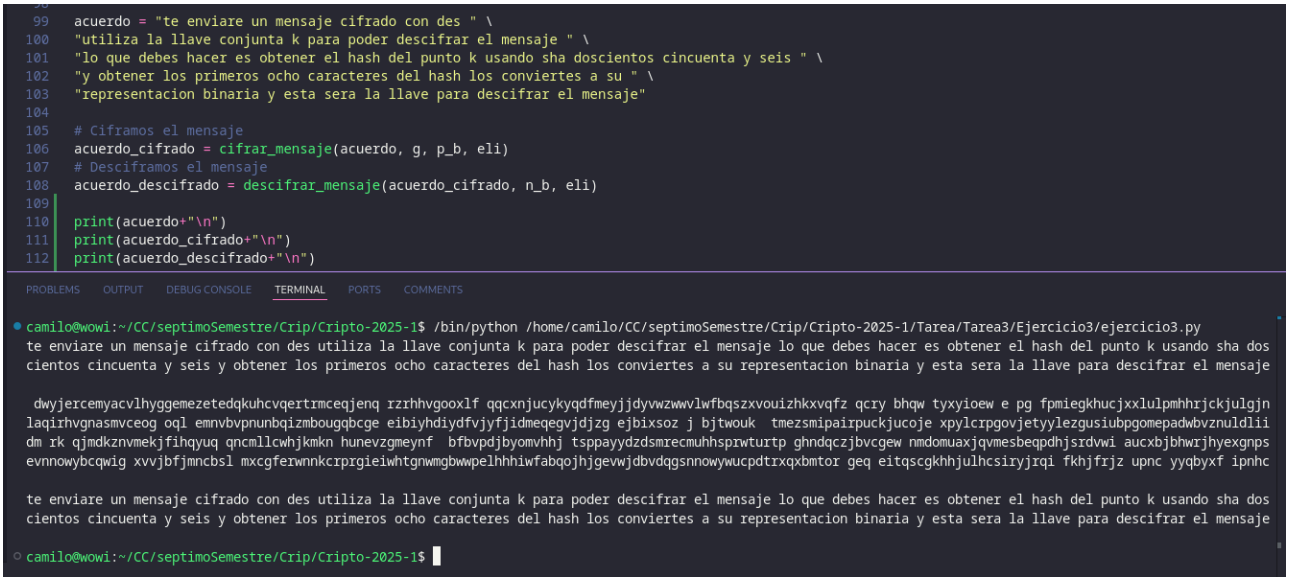
Luego Alice y Bob se comparten sus llaves públicas para así poder generar el punto secreto K que van a compartir, este punto se calcula con $n_a \times P_b = K = n_b \times P_a$, por lo tanto $K = (15, 7)$.

```
89   # Alicia calcula k
90   k_a = eli.mult(n_a, p_b)
91
92   # Bob calcula k
93   k_b = eli.mult(n_b, p_a)
94
95   print(k_a)
96   print(k_b)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMEN

camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$ /b
(15, 7)
(15, 7)
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$
```

Posteriormente Alice y Bob usarán el punto K para generar una llave que usarán para el cifrado DES. Para esto Alice le envía un mensaje con las instrucciones para poder usar el punto K para la clave de DES, el mensaje cifrado usando la curva elíptica.
El mensaje es “te enviare un mensaje cifrado con des utiliza la llave conjunta k para poder descifrar el mensaje lo que debes hacer es obtener el hash del punto k usando sha doscientos cincuenta y seis y obtener los primeros ocho caracteres del hash los conviertes a su representacion binaria y esta sera la llave para descifrar el mensaje”.



Por lo tanto Bob recibe el mensaje cifrado y lo descifra para enterarse de la manera de usar el punto K como llave del cifrado DES.
Luego Alice usa el punto K para obtener la llave usada en el cifrado DES, esta la obtiene usando sha256 para obtener el hash de K, se obtienen los primeros 8 caracteres en su representación binaria y se llena hasta completar 64 bits.
Por lo tanto la llave sería
“0010100000110001001101010010110000100000001101110010100100000000”.

Posteriormente Alice cifra, usando DES, el mensaje “lunes” usando la llave obtenida y envía el mensaje cifrado a Bob. El mensaje cifrado es “\ú;ò|çW”.

```
118   # Mensaje a enviar
119   mensaje = 'lunes'
120   mensaje_cifrado = cifrar(mensaje, llave_des)
121
122   print(mensaje_cifrado)
123
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMME

camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$ /b
\ú;ò|çW
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$
```

Por último Bob descifra el mensaje, usando DES y la llave generada con el punto K, obteniendo el mensaje “lunes”.

```
118   # Mensaje a enviar
119   mensaje = 'lunes'
120   mensaje_cifrado = cifrar(mensaje, llave_des)
121   mensaje_descifrado = descifrar(mensaje_cifrado, llave_des)
122
123   print(f"Mensaje original: {mensaje}")
124   print(f"Mensaje cifrado: {mensaje_cifrado}")
125   print(f"Mensaje descifrado: {mensaje_descifrado}")
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   COMMENTS

camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$ /bin/python /ho
Mensaje original: lunes
Mensaje cifrado: \ú;ò|çW
Mensaje descifrado: lunes
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$
```

De esta manera Alice y Bob se pudieron comunicar el mensaje “lunes”, mediante el canal inseguro, ya que por este solo pasaron sus llaves públicas de la curva, el acuerdo cifrado y el mensaje cifrado.

Fuentes usadas para este ejercicio:
Clases de teoría
Practica 10
Tarea 2
Tarea Moral de DES

4. Sean $p = 1217$, $\alpha = 3$ y $\beta = 37$. Calcula $log_{\alpha}\beta$ mediante el algoritmo de Cálculo de índices.

Para este ejercicio usamos la siguiente función(se encuentra en logaritmo.py), notemos que el algoritmo no es el mismo visto en la clase, si no usamos una combinación rara de lo que encontramos, esto debido a que el visto en la clase está algo complicado entenderlo e implementarlo. Además usamos las bibliotecas sympy,

solo para poder resolver matrices (para ejecutar el código se necesita tener sympy instalado) y random, solo para generar números aleatorios.
Algoritmo para el cálculo del logaritmo discreto.

```python
# Funcion para calcular logaritmo discretos usando index-calculus
# Calcular log_a b mod p
# Fuente https://github.com/david-r-cox/pyDLP
def index_calc(p, a, b):
    # Paso 1, generamos la base y las congruencias, se generan 100 congruencias
    congruencias = []
    base = set()
    while len(congruencias) < 100:
        # Paso 2, generamos un numero aleatorio k
        k = randint(2, p)
        valor = pow(a, k, p)
        # Paso 3, verificamos si es B-suave
        es_suave, factores = fact_b_suave(100, valor)
        if es_suave:
            # Paso 4, añadimos la congruencia a la lista y actualizamos la base
            congruencia = (agrupar_factores(factores), k)
            congruencias.append(congruencia)
            base.update(congruencia[0].keys())
    base = list(base)
    # Paso 5, construimos la matriz
    matriz = []
    vector = []
    for congruencia, k in congruencias:
        fila = [congruencia.get(b, 0) for b in base]
        matriz.append(fila)
        vector.append(k)
    # Paso 6, resolvemos el sistema
    matriz_s = Matrix(matriz)
    vector_s = Matrix(vector)
    solucion = matriz_s.solve_least_squares(vector_s)
    # Paso 7, generamos los exponentes y logaritmos
    exponentes = [int(x) % (p - 1) for x in solucion]
    logaritmos = {base: exp for base, exp in zip(base, exponentes)}
    # Paso 8, intentamos resolver el logaritmo, en un numero maximo de 100000000 iteraciones
    for i in range (100000000):
        k = randint(2, p)
        # Paso 9, generamos un candidato
        inv = pow(a, -1, p)
        candidato = (b * pow(inv, k, p)) % p
        # Paso 10, verificamos si es B-suave
        es_suave, factores = fact_b_suave(100, candidato)
        if es_suave:
            # Paso 11, verificamos si es un logaritmo valido
            factores_exp = agrupar_factores(factores)
            eval = sum(logaritmos.get(base, 0) * exp for base, exp in factores_exp.items()) % (p - 1)
            eval = (eval + k) % (p - 1)
            if pow(a, eval, p) == b:
                return eval
    return None
```

Y las siguientes funciones auxiliares:
Función para factorizar un número y ver si es B-smooth.

```python
# Funcion para factorizar un numero y ver si es B-suave
def fact_b_suave(b, n):
    factores = factorizar_n(n)
    if len(factores) > 0 and max(factores) <= b:
        return True, factores
    return False, factores
```

Función para factorizar un número.

```python
# Funcion para factorizar un numero
def factorizar_n(n):
    factores = []
    factor = 2
    while factor * factor <= n:
        while n % factor == 0:
            factores.append(factor)
            n //= factor
        factor += 1
    if n > 1:
        factores.append(n)
    return factores
```

Función para agrupar factores y obtener exponentes.

```python
# Funcion para agrupar los factores y asi tener un exponente
def agrupar_factores(factores):
    factores_agrupados = {}
    for factor in factores:
        factores_agrupados[factor] = factores_agrupados.get(factor, 0) + 1
    return factores_agrupados
```

Nuestro código nos dio el siguiente resultado.

```python
# Calculamos log_3 37 mod 1217
a = 3
b = 37
p = 1217
for i in range(0, p-1):
    if pow(a, i, p) == b:
        print(f"log_{a} {b} mod {p} = {i}, usando fuerza bruta")
        break
log = index_calc(p, a, b)
if log is not None:
    print(f"log_{a} {b} mod {p} = {log}, usando index-calculus")
else:
    print("No se logro encontrar el logaritmo, intenta con mas iteraciones")
```

```
PROBLEMS 4  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$ /bin/python /home/camilo/CC/septim
log_3 37 mod 1217 = 588, usando fuerza bruta
log_3 37 mod 1217 = 588, usando index-calculus
camilo@wowi:~/CC/septimoSemestre/Crip/Cripto-2025-1$
```

Por lo tanto $log_3 37 \; mod \; 1217 \; = \; 588$.

Nota, a veces el algoritmo se tarda mucho, en ese caso, mejor detenerlo y volver a ejecutarlo.

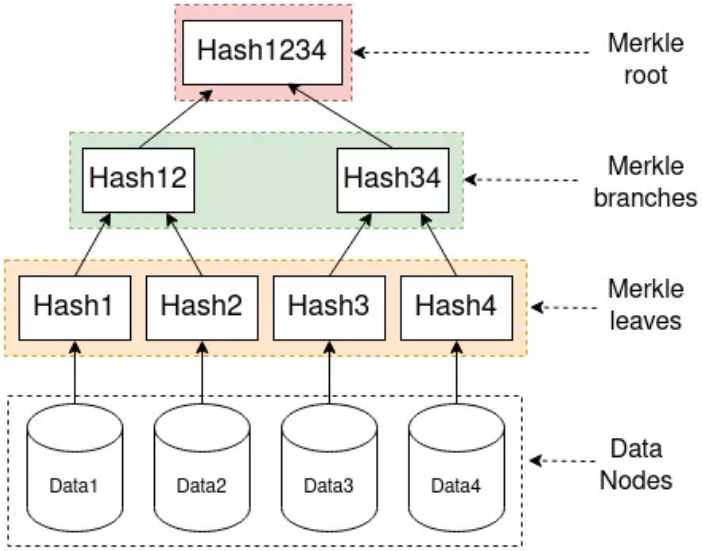
Fuentes usadas para este ejercicio:

Ayudantías
https://github.com/david-r-cox/pyDLP
Padmavathy, R., & Bhagvati, C. (2012). Discrete logarithm problem using index calculus method. Mathematical and Computer Modelling, 55(1-2), 161–169. https://doi.org/10.1016/j.mcm.2011.02.022

Wikipedia Contributors. (2024, January 15). Index calculus algorithm. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Index_calculus_algorithm

5. Realiza una breve investigación acerca del esquema de firma de Merkle (MSS). La investigación debe responder como mínimo las siguientes preguntas
    a. ¿Qué es un árbol de Merkle?
    b. ¿De qué manera se generan las firmas?
    c. ¿De qué manera se verifican las firmas?

Los árboles de Merkle son esquemas de firmas, los cuales se basan en el uso de funciones hash y en formar estructuras muy similares a árboles binarios, con la peculiaridad de que las hojas del árbol contienen hashes de información o datos y los nodos internos contienen la concatenación del contenido de sus hijos. Y por último en la raíz del árbol van a estar en conjunto todos los hashes de los datos.
En la siguiente imagen podemos ver una representación de cómo sería, notando que en las hojas están los hashes y en los nodos intermedios tenemos la concatenación de lo de abajo (los hijos).



Una estrategia para poder completar algunos árboles desbalanceados es copiar lo que tiene el nodo hermano.
El proceso de generar firmas empieza con tener información que se va a usar para generar los hashes de las hojas del árbol o tener/generar firmas privadas de uso único, esto se guardará en nodos de información (data nodes). Después usaremos los nodos de información para poder generar hashes, a partir de aplicar la función hash a la información de los nodos, de esta manera generamos hashes que van a ser guardados en las hojas del árbol (Merkle leaves). Luego se van a generar los nodos internos del árbol, esto se realiza al juntar las hojas en parejas, pero si el número de hojas es impar entonces la hoja que quede sin pareja se juntara consigo mismo, de esta forma todas las hojas están en parejas y formarán un nuevo nodo el cual tendrá la concatenación de los hashes que contengan los nodos de la pareja.

Posteriormente se va a repetir el proceso de juntar nodos en parejas (como esta arriba) hasta que solo lleguemos a tener un nodo (la raíz), siempre respetando que el contenido de un nodo intermedio es la concatenación del contenido de sus hijos. Y por último la llave generada será el contenido de la raíz de árbol (la imagen de arriba es un ejemplo de cómo quedaría todo el árbol al final). Pero algo importante que tenemos que notar, es que si el contenido de un nodo es modificado entonces el contenido de los nodos que forman el camino de la raíz al nodo modificado también se verán modificados.
Ahora veamos cómo se verifican las llaves. Para este proceso se usa un camino de verificación de Merkle o prueba de Merkle, el cual funciona empezando desde una hoja del árbol y sube el árbol hasta llegar a la raíz de este, revisando los nodos para poder verificar que este la información o hash de la llave que se va a verificar. Estas verificaciones empiezan con la hoja donde se revisa si la hoja contiene el hash inicial, después se verifica usando el nodo padre de la hoja (por lo cual estaremos revisando también el hash del otro nodo hijo), este proceso se va a repetir hasta llegar a la raiz del arbol. En caso de que se llegue a la raíz y todas las verificaciones fueron válidas entonces tendremos que la llave es válida, pero en caso de que alguna validación sea inválida entonces la llave no es válida.
Algunos de los usos de los árboles de Merkle son en criptomonedas, blockchains y bases de datos distribuidas.

Fuentes usadas para este ejercicio:
Kanstrén, T. (2021, February 20). Merkle Trees: Concepts and Use Cases. Medium. https://medium.com/coinmonks/merkle-trees-concepts-and-use-cases-5da873702318
Prahalad, B. (2018, December 13). Merkle proofs Explained. Crypto-0-Nite. https://medium.com/crypto-0-nite/merkle-proofs-explained-6dd429623dc5
Rouse, M. (2024, March 26). Árbol de Merkle (árbol hash de Blockchain). Techopedia Español. https://www.techopedia.com/es/definicion/arbol-merkle
Becker, G. (n.d.). Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d7c3aa65bc5df32d94dcc8b29dceca240bdf8bef