INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE MONTERREY

ESCUELA CIENCIAS APLICADAS

Uso de álgebras modernas para seguridad y criptografía

Grupo: 01

Documentación
Unidad MA2006B

8 de febrero de 2023

Profesores:
Dr.Alberto F. Martinez
Dr.Salvador Mancilla

Alumnos:
Pablo Ramirez Santaella Urencio A01197501
Francisco Leonid Gálvez Flores A01174385
Francisco Castorena Salazar A00827756
Daniel Sánchez Villarreal A01197699
Iker Ledesma Durán A01653115

Índice

Uso de álgebras modernas para seguridad y criptografía
Documentación

05 de febrero de 2023

1. Introduccion

En este documento se describirán las funciones utilizadas para la realización del proyecto de firmado digital. En este se comentarán sobre las librerías y funciones que fueron de utilidad para el centro de control y auditor.

2. Auditor

Para los códigos que ayudan a que el auditor funcione, se requieren de 4 archivos.

1. Algoritmo ECDSA
2. ufirebase
3. Producción
4. Consumo

2.1. Algoritmo ECDSA

2.1.1. Código Completo

El código completo que utilizamos para utilizar curvas elípticas para generación de firmas y verificación es el siguiente. (Los siguientes apartados lo explican más a detalle):

```
import hashlib
import random
```

```
import binascii

def extended_gcd(a, b):
    x0, x1, y0, y1 = 1, 0, 0, 1
    while b != 0:
        q, a, b = a // b, b, a % b
        x0, x1 = x1, x0 - q * x1
        y0, y1 = y1, y0 - q * y1
    return a, x0, y0


def calc_inv(a, Zn):
    gcd, x, y = extended_gcd(a, Zn)
    if gcd != 1:
        return False
    return x % Zn


def inv_aditivo(x, p):
    if x <= p - 1:
        return p - x
    if x > p:
        return p - (x % p)


def exp_binaria(x, n):
    resultado = 1
    q = n - 2
    while q > 0:
        if q % 2 == 1:
            resultado = (resultado * x) % n
        x = (x * x) % n
        q = q // 2
    return resultado


def pendiente(x1, y1, x2, y2, a, p):
    if (x1 and y1) is None or (x2, y2) is None:
        return (x1, y1) or (x2, y2)
    if (x1 == x2 and y1 == y2):
        return (((3 * x1 * x1) + a) * exp_binaria(2 * y1, p)) % p
    else:
        return ((y2 + inv_aditivo(y1, p)) * exp_binaria(x2 + inv_aditivo(x1, p),
    p)) % p


def suma_de_la_curva(x1, y1, x2, y2, s, p):
    x3 = 0
    y3 = 0
```

```
51
52      x3 = (s * s + inv_aditivo(x1, p) + inv_aditivo(x2, p)) % p
53      y3 = (s * (x1 + inv_aditivo(x3, p)) + inv_aditivo(y1, p)) % p
54      return x3, y3
55
56
57  def mult_binaria(x, y, a, p, n):
58      n = bin(n)[3:]
59      x2 = x
60      y2 = y
61      for i in n:
62          if i == '1':
63              s = pendiente(x2, y2, x2, y2, a, p)
64              x2, y2 = suma_de_la_curva(x2, y2, x2, y2, s, p)
65              s = pendiente(x2, y2, x, y, a, p)
66              x2, y2 = suma_de_la_curva(x, y, x2, y2, s, p)
67
68          if i == '0':
69              s = pendiente(x2, y2, x2, y2, a, p)
70              x2, y2 = suma_de_la_curva(x2, y2, x2, y2, s, p)
71
72      return x2, y2
73
74
75
76  def sign_message(message, private_key):
77      # Definiendo curva NIST P-256
78      p = int(0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff)
79      a = int(0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc)
80      b = int(0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b)
81      gx = int(0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296)
82      gy = int(0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5)
83      n = int(0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551)
84      x = private_key
85      # = 1
86
87      # k = random.randint(1, n - 1)
88      k = int(0xA6E3C57DD01ABE90086538398355DD4C3B17AA873382B0F24D6129493D8AAD60)
89      x1, y1 = mult_binaria(gx, gy, a, p, k)
90      r = x1 % n
91
92      # Se regresa al paso 1 en caso de que r==0
93      if r == 0:
94          return sign_message(message, private_key)
95
96      # Generando k inversa
97      k_inv = calc_inv(k, n)
98      # k_inv = pow(k,-1,n)
99      # Convirtiendo mensaje hasheado a entero
```

```
    hash_object = hashlib.sha256()
    hash_object.update(message.encode())
    hexa = binascii.hexlify(hash_object.digest())
    e = int(hexa,16)
    s = (k_inv * (e + x * r)) % n

    # Se regresa al paso 1 en caso de que s==0
    if s == 0:
        return sign_message(message, private_key)
    return (r, s)

def verify_ecdsa_signature(r, s, m, n, G, Q, a):
    # Paso 1: Verify que r y s esten en el intervalo [1, n-1]
    if  not(1 <= r < n and 1 <= s < n):
        return 'Step 1 Failed'

    # Paso 2: Hashear mensaje con SHA-256 y convertirlo a decimal
    hash_object = hashlib.sha256()
    hash_object.update(m.encode())
    hexa = binascii.hexlify(hash_object.digest())
    e = int(hexa,16)

    # Paso 3: Computar w = s^-1 mod n
    w = calc_inv(s, n)

    # Paso 4: Computar u1 = (e * w) mod n and u2 = (r * w) mod n
    u1 = (e * w) % n
    u2 = (r * w) % n

    # Paso 5: Computar X = u1G + u2Q
    x1, y1 = mult_binaria(G[0],G[1],a,p,u1)
    x2, y2 = mult_binaria(Q[0],Q[1],a,p,u2)
    pend = pendiente(x1,y1,x2,y2,a,p)
    x3, y3 = suma_de_la_curva(x1,y1,x2,y2,pend,p)

    # Paso 6: If X == O, rechazar la firma
    if (x3, y3) == (0, 0):
        return 'Rechazar firma'

    # Paso 7: Computar v = x3 mod n
    v = x3 % n
    # Step 8: Aceptar la firma si y solo si v = r
    return v==r

p = int(0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff)
a = int(0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc)
b = int(0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b)
gx = int(0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296)
gy = int(0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5)
```

```
n = int(0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551)
x = int(0xC9AFA9D845BA75166B5C215767B1D6934E50C3DB36E89B127B8A622B120F6721)
Ux,Uy = mult_binaria(gx,gy,a,p,x)


r,s = sign_message("sample", x)

print(hex(r))
print(hex(s))

verify_ecdsa_signature(r,s,"sample",n,(gx, gy),(Ux, Uy), a)
```

2.2. Función para calcular inverso multiplicativo

Tenemos dos funciones para calcular el inverso multiplicativo, tenemos una función que utiliza la función de Euclides, la cual nos ayuda a poder resolver el inverso multiplicativo de no solo números primos sino también de números compuestos. También tenemos la función de exponenciación binaria, la cual toma números primos y les calcula el inverso por exponenciación binaria. Las funciones son las siguientes:

2.2.1. Función extended gcd

```
def extended_gcd(a, b):
    x0, x1, y0, y1 = 1, 0, 0, 1
    while b != 0:
        q, a, b = a // b, b, a % b
        x0, x1 = x1, x0 - q * x1
        y0, y1 = y1, y0 - q * y1
    return a, x0, y0
```

La función toma como parámetros 'a' y 'b' estos dos parámetros son la 'a' y 'b' de la curva elíptica. Después asigna valores a las x's y a las y's para meterlos a un ciclo while en el cual mientras 'b' sea diferente a 0 entonces siga corriendo lo siguiente, 'q' es igual 'a' entre 'b' solo dando la cantidad de divisiones de 'b' que caben en 'q', 'a' es igual a 'a' entre 'b' solo dando la cantidad de divisiones.

2.2.2. Función exp binaria

```
def exp_binaria(x, n):
    resultado = 1
    q = n - 2
    while q > 0:
        if q % 2 == 1:
            resultado = (resultado * x) % n
        x = (x * x) % n
        q = q // 2
    return resultado
```

Esta función va a regresar el inverso multiplicativo de x módulo n, siempre y cuando n sea primo, esta función puede ser más rápida de calcular que por medio de el algoritmo extendido de Euclides, sin embargo, no va a calcular el inverso de forma correcta si el parámetro n no es primo.

2.3. Función para calcular inverso aditivo

```
def inv_aditivo(x, p):
    if x <= p-1:
        return p-x
    if x > p:
        return p - (x%p)
```

Esta función tiene como parámetros los valores x y p, y esta función lo que hace es que calcula y regresa al usuario el inverso aditivo de x módulo p. Para esto, se utiliza un condicional if para que en el caso de que el valor del parámetro x sea igual o menor que el valor resultante de p-1, entonces la función regresa el valor resultante de p-x. Por otra parte, en un segundo condicional if, subsecuente al primero, se evalúa el posible caso de que el valor del parámetro x sea mayor que p, lo cual de ser así, la función en este caso regresaría el valor resultante de p - (x mod p).

2.4. Función para calcular la pendiente

```
def pendiente(x1, y1, x2, y2, a, p):
    if (x1 and y1) is None or (x2, y2) is None:
        return (x1, y1) or (x2, y2)
    if (x1 == x2 and y1 == y2):
        return (((3 * x1 * x1) + a) * exp_binaria(2 * y1, p)) % p
    else:
        return ((y2 + inv_aditivo(y1, p)) * exp_binaria(x2 + inv_aditivo(x1, p),
   p)) % p
```

La pendiente se calcula dependiendo si los puntos en la curva son iguales en coordenadas o diferentes. Se tienen tres casos:

- P==Q: En este caso se calcula la pendiente de la siguiente manera:

$$s = \frac{3x_1^2 + a}{2y_1} \tag{1}$$

- $P \neq Q$ y ninguno es elemento neutro: En este caso se calcula la pendiente de la siguiente manera:

$$s = \frac{y_2 - y_1}{x_2 - x_1} \tag{2}$$

- $P \neq Q$ y uno es el elemento neutro: Aquí simplemente se regresa el elemento no neutro.

Las fórmulas para calcular la pendiente pueden ser consultadas aquí [13]. Cabe mencionar que las operaciones de pendiente son sobre un módulo p, el cual se define por la curva elíptica en la que está trabajando.

2.5. Función para sumar puntos de curva

```
def suma_de_la_curva(x1, y1, x2, y2, s, p):
    x3 = 0
    y3 = 0

    x3 = (s * s + inv_aditivo(x1, p) + inv_aditivo(x2, p)) % p
    y3 = (s * (x1 + inv_aditivo(x3, p)) + inv_aditivo(y1, p)) % p
    return x3, y3
```

Esta función va a necesitar como parámetro a la pendiente de la curva, por lo que primero se debe de calcular esta. Una vez calculada la pendiente se calcula $(x_3, y_3$, el cual es el resultado de la suma de dos puntos. Se calculan según las siguientes ecuaciones:

$$x_3 = s^2 - x_1 - x_2 \tag{3}$$

$$y_3 = s(x_1 - x_3) - y_1 \tag{4}$$

De igual forma, las fórmulas para calcular la suma de puntos pueden ser consultadas aquí [13]. Los resultados de las sumas son sobre el respectivo módulo de la curva elíptica definida.

2.6. Función para multiplicación binaria

```
def mult_binaria(x, y, a, p, n):
    n = bin(n)[3:]
    x2 = x
    y2 = y
    for i in n:
        if i == '1':
            s = pendiente(x2, y2, x2, y2, a, p)
            x2, y2 = suma_de_la_curva(x2, y2, x2, y2, s, p)
            s = pendiente(x2, y2, x, y, a, p)
            x2, y2 = suma_de_la_curva(x, y, x2, y2, s, p)

        if i == '0':
            s = pendiente(x2, y2, x2, y2, a, p)
            x2, y2 = suma_de_la_curva(x2, y2, x2, y2, s, p)

    return x2, y2
```

Esta función retorna la multiplicación de un punto de una curva elíptica G=(x,y) predefinida por los parámetros a y p, por un escalar n, de tal forma que:

$$nG = \sum_{1}^{n} G \tag{5}$$

La cualidad de esta función es que la computación no va a tener que ser generada n veces, sino que se disminuirá el número de operaciones necesarias para calcular el resultado de acuerdo al algoritmo double and add, el funcionamiento del algoritmo puede consultar aquí [1].

2.7. Función Generación de firma

```python
def sign_message(message, private_key):
    # Definiendo curva NIST P-256
    p = int(0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff)
    a = int(0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc)
    b = int(0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b)
    gx = int(0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296)
    gy = int(0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5)
    n = int(0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551)
    x = private_key
    # = 1

    # k = random.randint(1, n - 1)
    k = int(0xA6E3C57DD01ABE90086538398355DD4C3B17AA873382B0F24D6129493D8AAD60)
    x1, y1 = mult_binaria(gx, gy, a, p, k)
    r = x1 % n

    # Se regresa al paso 1 en caso de que r==0
    if r == 0:
        return sign_message(message, private_key)

    # Generando k inversa
    k_inv = calc_inv(k, n)
    # k_inv = pow(k,-1,n)
    # Convirtiendo mensaje hasheado a entero
    hash_object = hashlib.sha256()
    hash_object.update(message.encode())
    hexa = binascii.hexlify(hash_object.digest())
    e = int(hexa,16)
    s = (k_inv * (e + x * r)) % n

    # Se regresa al paso 1 en caso de que s==0
    if s == 0:
        return sign_message(message, private_key)
    return (r, s)
```

Para la generación de la firma definimos los parámetros de la curva a utilizar dentro de la función, para el caso del reto, estaremos utilizando la NIST P-256, sus parámetros pueden ser consultados aquí [3]. Por otra parte, la función recibe dos parámetros, el mensaje a encriptar y la clave privada x con la cual se quiere encriptar la información, una vez definidos los parámetros de la curva y el mensaje, se siguen los siguientes pasos para la generación de la firma[13]:

1. Se define una llave privada x de forma pseudo-aleatoria entre 1 y $n-1$, donde n es el número

de elementos que cumplen con la condición de congruencia

$$y^2 \equiv x^3 + ax + b \mod p \tag{6}$$

2. Se computa la llave pública xG, donde G es el punto generador de la curva, en nuestro caso G se definió de acuerdo a la curva P-256.

$$Ux, Uy = x(Gx, Gy) \tag{7}$$

3. Se define k de forma pseudo-aleatoria entre 1 y $n - 1$.

4. Se computa la llave pública kG, de tal forma que:

$$x_1, y_1 = k(Gx, Gy) \tag{8}$$

5. Se calcula $r = x_1 \mod n$

6. En caso de que $r == 0$ se regresa al paso número 3.

7. Se calcula $k^{-1} \mod n$ con la función calc_inv() o con exponenciación binaria.

8. Se calcula el hash del mensaje a encriptar, en nuestro caso con el algoritmo SHA-256, después pasamos el resultado de hexadecimal a un entero.

9. Se calcula $s = k^{-1}(e + xr) \mod n$.

10. En caso de que $s == 0$ se regresa al paso número 3.

11. En caso de que $s \neq 0$ se regresa r y s.

2.8. Función Verificación de firma

```
def verify_ecdsa_signature(r, s, m, n, G, Q, a):
    # Paso 1: Verify que r y s esten en el intervalo [1, n-1]
    if  not(1 <= r < n and 1 <= s < n):
        return 'Step 1 Failed'

    # Paso 2: Hashear mensaje con SHA-256 y convertirlo a decimal
    hash_object = hashlib.sha256()
    hash_object.update(m.encode())
    hexa = binascii.hexlify(hash_object.digest())
    e = int(hexa,16)

    # Paso 3: Computar w = s^-1 mod n
```

```
    w = calc_inv(s, n)

    # Paso 4: Computar u1 = (e * w) mod n and u2 = (r * w) mod n
    u1 = (e * w) % n
    u2 = (r * w) % n

    # Paso 5: Computar X = u1G + u2Q
    x1, y1 = mult_binaria(G[0],G[1],a,p,u1)
    x2, y2 = mult_binaria(Q[0],Q[1],a,p,u2)
    pend = pendiente(x1,y1,x2,y2,a,p)
    x3, y3 = suma_de_la_curva(x1,y1,x2,y2,pend,p)

    # Paso 6: If X == O, rechazar la firma
    if (x3, y3) == (0, 0):
        return 'Rechazar firma'

    # Paso 7: Computar v = x3 mod n
    v = x3 % n
    # Step 8: Aceptar la firma si y solo si v = r
    return v==r
```

Para esta función, se toman como parámetros r y s, los cuales deben de estar en un intervalo de [1, n-1], además de un mensaje de prueba m en forma de string, así como también otros parámetros que ya se habían visto anteriormente como G, que representa la multiplicación de un punto de la curva elíptica, así como otros parámetros previamente mencionados como n, a, etc. Para esta función, lo primero que se hace es que se debe de revisar si r y s efectivamente se encuentran en el intervalo mencionado, esto por medio de un condicional if para poder confirmar esto. En caso de que no, la función va a regresar un mensaje notificando al usuario que está fallando esta cuestión. Posteriormente, se hashea el mensaje m usando SHA-256 y se convierte a formato decimal. Después, se manda a llamar la función calc inv con el fin de utilizarla para calcular el inverso multiplicativo de s y usando el parámetro n para calcular la parte de módulo n. Asimismo, se calcula después u1 y u2 utilizando algunos de estos parámetros mencionados, como r y n por ejemplo, así como la variable w, que representa el inverso multiplicativo que se calculó anteriormente, haciendo uso de los parámetros s y n. Tras esto, se calculan x1,y1 y x2,y2 mediante la función mult binaria, empleándose para la multiplicación binaria como se explicó anteriormente. Asimismo, se usa también la función para suma de la curva para calcular x3 y y3. Después, se verifica si estas coordenadas equivalen a 0,0 para, en caso de que sea así, imprimir un mensaje notificando al usuario que se rechaza la firma. Finalmente, se usa nuevamente el parámetro n para calcular x3 módulo n y se mete esto en una variable para posteriormente aceptar la firma, lo cual se hace de manera válida solamente si esta variable en cuestión resulta ser igual que el parámetro r.

2.9. Ejemplo de uso

```
p = int(0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff)
a = int(0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc)
```

```
b = int(0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b)
gx = int(0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296)
gy = int(0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5)
n = int(0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551)
x = int(0xC9AFA9D845BA75166B5C215767B1D6934E50C3DB36E89B127B8A622B120F6721)
Ux,Uy = mult_binaria(gx,gy,a,p,x)


r,s = sign_message("sample", x)

print(hex(r))
print(hex(s))

verify_ecdsa_signature(r,s,"sample",n,(gx, gy),(Ux, Uy), a)
```

2.10. ufirebase

El algoritmo de üfirebase"sirvió para que se pudiera conectar de manera efectiva la placa ESP-32 a las bases de datos en "Google Firebase". Esto con la finalidad de no darle una alta demanda de trabajo a la placa ya que cuenta con una escasa memoria, la cuál se puede aprovechar mejor al quitarle esta carga.

```
class INTERNAL:
  def connect(id):
      LOCAL_ADINFO=usocket.getaddrinfo(FIREBASE_GLOBAL_VAR.GLOBAL_URL_ADINFO["
    host"], FIREBASE_GLOBAL_VAR.GLOBAL_URL_ADINFO["port"], 0, usocket.SOCK_STREAM)
    [0]
      FIREBASE_GLOBAL_VAR.SLIST["S"+id] = usocket.socket(LOCAL_ADINFO[0],
    LOCAL_ADINFO[1], LOCAL_ADINFO[2])
      FIREBASE_GLOBAL_VAR.SLIST["S"+id].connect(LOCAL_ADINFO[-1])
      if FIREBASE_GLOBAL_VAR.GLOBAL_URL_ADINFO["proto"] == "https:":
          try:
            FIREBASE_GLOBAL_VAR.SLIST["SS"+id] = ussl.wrap_socket(
    FIREBASE_GLOBAL_VAR.SLIST["S"+id], server_hostname=FIREBASE_GLOBAL_VAR.
    GLOBAL_URL_ADINFO["host"])
          except:
            print("ENOMEM, try to restart. If you make to many id's (sokets)
    simultaneously (bg=1 and id=x), try to use less or use a board with more ram!\
    nSome emulation software limits the RAM.")
            FIREBASE_GLOBAL_VAR.SLIST["S"+id].close()
            FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=None
            FIREBASE_GLOBAL_VAR.SLIST["S"+id]=None
            raise MemoryError

      else:
          FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=FIREBASE_GLOBAL_VAR.SLIST["S"+id]

  def disconnect(id):
```

```
      FIREBASE_GLOBAL_VAR.SLIST["SS"+id].close()
      FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=None
      FIREBASE_GLOBAL_VAR.SLIST["S"+id]=None
```

En este apartado se puede ver la función çonnect", que sirve para poder conectarse a una firebase que sea proporcionada. De igual forma viene su alternativa para poder desconectarse de la firebase, que es "disconnect".

```

def put(PATH, DATA, id, cb):
    try:
      while FIREBASE_GLOBAL_VAR.SLIST["SS"+id]:
        time.sleep(2)
      FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=True
    except:
      FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=True
    INTERNAL.connect(id)
    LOCAL_SS=FIREBASE_GLOBAL_VAR.SLIST["SS"+id]
    LOCAL_SS.write(b"PUT /"+PATH+b".json HTTP/1.0\r\n")
    LOCAL_SS.write(b"Host: "+FIREBASE_GLOBAL_VAR.GLOBAL_URL_ADINFO["host"]+b"\r
 \n")
    LOCAL_SS.write(b"Content-Length: "+str(len(DATA))+"\r\n\r\n")
    LOCAL_SS.write(DATA)
    LOCAL_DUMMY=LOCAL_SS.read()
    del LOCAL_DUMMY
    INTERNAL.disconnect(id)
    if cb:
      try:
        cb[0](*cb[1])
      except:
        try:
          cb[0](cb[1])
        except:
          raise OSError("Callback function could not be executed. Try the
 function without ufirebase.py callback.")


def patch(PATH, DATATAG, id, cb):
    try:
      while FIREBASE_GLOBAL_VAR.SLIST["SS"+id]:
        time.sleep(1)
      FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=True
    except:
      FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=True
    INTERNAL.connect(id)
    LOCAL_SS=FIREBASE_GLOBAL_VAR.SLIST["SS"+id]
    LOCAL_SS.write(b"PATCH /"+PATH+b".json HTTP/1.0\r\n")
    LOCAL_SS.write(b"Host: "+FIREBASE_GLOBAL_VAR.GLOBAL_URL_ADINFO["host"]+b"\r
 \n")
```

```
      LOCAL_SS.write(b"Content-Length: "+str(len(DATATAG))+"\r\n\r\n")
      LOCAL_SS.write(DATATAG)
      LOCAL_DUMMY=LOCAL_SS.read()
      del LOCAL_DUMMY
      INTERNAL.disconnect(id)
      if cb:
        try:
          cb[0](*cb[1])
        except:
          try:
            cb[0](cb[1])
          except:
            raise OSError("Callback function could not be executed. Try the function without ufirebase.py callback.")
```

Previamente se vieron las funciones "patchz "put", estas funciones sirven para agregar información, valores o datos que deseemos agregar a la base de datos.

```
  def get(PATH, DUMP, id, cb, limit):
      try:
        while FIREBASE_GLOBAL_VAR.SLIST["SS"+id]:
          time.sleep(1)
        FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=True
      except:
        FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=True
      INTERNAL.connect(id)
      LOCAL_SS=FIREBASE_GLOBAL_VAR.SLIST["SS"+id]
      LOCAL_SS.write(b"GET /"+PATH+b".json?shallow="+ujson.dumps(limit)+b" HTTP/1.0\r\n")
      LOCAL_SS.write(b"Host: "+FIREBASE_GLOBAL_VAR.GLOBAL_URL_ADINFO["host"]+b"\r\n\r\n")
      LOCAL_OUTPUT=ujson.loads(LOCAL_SS.read().splitlines()[-1])
      INTERNAL.disconnect(id)
      globals()[DUMP]=LOCAL_OUTPUT
      if cb:
        try:
          cb[0](*cb[1])
        except:
          try:
            cb[0](cb[1])
          except:
            raise OSError("Callback function could not be executed. Try the function without ufirebase.py callback.")

  def getfile(PATH, FILE, bg, id, cb, limit):
      try:
        while FIREBASE_GLOBAL_VAR.SLIST["SS"+id]:
          time.sleep(1)
        FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=True
```

```
      except:
        FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=True
      INTERNAL.connect(id)
      LOCAL_SS=FIREBASE_GLOBAL_VAR.SLIST["SS"+id]
      LOCAL_SS.write(b"GET /"+PATH+b".json?shallow="+ujson.dumps(limit)+b" HTTP
  /1.0\r\n")
      LOCAL_SS.write(b"Host: "+FIREBASE_GLOBAL_VAR.GLOBAL_URL_ADINFO["host"]+b"\r
  \n\r\n")
      while not LOCAL_SS.readline()==b"\r\n":
        pass
      LOCAL_FILE=open(FILE, "wb")
      if bg:
        while True:
          LOCAL_LINE=LOCAL_SS.read(1024)
          if LOCAL_LINE==b"":
            break
          LOCAL_FILE.write(LOCAL_LINE)
          time.sleep_ms(1)
      else:
        while True:
          LOCAL_LINE=LOCAL_SS.read(1024)
          if LOCAL_LINE==b"":
            break
          LOCAL_FILE.write(LOCAL_LINE)
      LOCAL_FILE.close()
      LOCAL_DUMMY=LOCAL_SS.read()
      del LOCAL_DUMMY
      INTERNAL.disconnect(id)
      if cb:
        try:
          cb[0](*cb[1])
        except:
          try:
            cb[0](cb[1])
          except:
            raise OSError("Callback function could not be executed. Try the
  function without ufirebase.py callback.")
```

Las función "get", sirve para obtener un valor con el cuál se desea trabajar o visualizar en la base de datos, en cambio, si lo que se desea el visualizar un archivo en la base de datos, se debería usar "getfile".

```
  def delete(PATH, id, cb):
      try:
        while FIREBASE_GLOBAL_VAR.SLIST["SS"+id]:
          time.sleep(1)
        FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=True
      except:
        FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=True
```

```
      INTERNAL.connect(id)
      LOCAL_SS=FIREBASE_GLOBAL_VAR.SLIST["SS"+id]
      LOCAL_SS.write(b"DELETE /"+PATH+b".json HTTP/1.0\r\n")
      LOCAL_SS.write(b"Host: "+FIREBASE_GLOBAL_VAR.GLOBAL_URL_ADINFO["host"]+b"\r
  \n\r\n")
      LOCAL_DUMMY=LOCAL_SS.read()
      del LOCAL_DUMMY
      INTERNAL.disconnect(id)
      if cb:
        try:
          cb[0](*cb[1])
        except:
          try:
            cb[0](cb[1])
          except:
            raise OSError("Callback function could not be executed. Try the
  function without ufirebase.py callback.")
```

En dado caso que se requieran eliminar ciertos datos, valores o archivos de la base de datos en la que uno esté trabajando, la función "delete"permite realizar esto.

```
def addto(PATH, DATA, DUMP, id, cb):
    try:
      while FIREBASE_GLOBAL_VAR.SLIST["SS"+id]:
        time.sleep(1)
      FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=True
    except:
      FIREBASE_GLOBAL_VAR.SLIST["SS"+id]=True
    INTERNAL.connect(id)
    LOCAL_SS=FIREBASE_GLOBAL_VAR.SLIST["SS"+id]
    LOCAL_SS.write(b"POST /"+PATH+b".json HTTP/1.0\r\n")
    LOCAL_SS.write(b"Host: "+FIREBASE_GLOBAL_VAR.GLOBAL_URL_ADINFO["host"]+b"\r
  \n")
    LOCAL_SS.write(b"Content-Length: "+str(len(DATA))+"\r\n\r\n")
    LOCAL_SS.write(DATA)
    LOCAL_OUTPUT=ujson.loads(LOCAL_SS.read().splitlines()[-1])
    INTERNAL.disconnect(id)
    if DUMP:
      globals()[DUMP]=LOCAL_OUTPUT["name"]
    if cb:
      try:
        cb[0](*cb[1])
      except:
        try:
          cb[0](cb[1])
        except:
          raise OSError("Callback function could not be executed. Try the
  function without ufirebase.py callback.")
```

La función .ªddto"sirve para poder agregar en un lugar en específico de la base de datos algún valor o dato que se necesite para complementar la base.

```
def setURL(url):
    FIREBASE_GLOBAL_VAR.GLOBAL_URL=url
    try:
        proto, dummy, host, path = url.split("/", 3)
    except ValueError:
        proto, dummy, host = url.split("/", 2)
        path = ""
    if proto == "http:":
        port = 80
    elif proto == "https:":
        import ussl
        port = 443
    else:
        raise ValueError("Unsupported protocol: " + proto)

    if ":" in host:
        host, port = host.split(":", 1)
        port = int(port)

    FIREBASE_GLOBAL_VAR.GLOBAL_URL_ADINFO={"proto": proto, "host": host, "port":
   port}
```

Para conectarse a un url en específico de la firebase con la que se esté trabajando, con la función "setURL"se puede lograr la conexión proporcionando dicho url.

```
def put(PATH, DATA, bg=True, id=0, cb=None):
    if bg:
      _thread.start_new_thread(INTERNAL.put, [PATH, ujson.dumps(DATA), str(id),
   cb])
    else:
      INTERNAL.put(PATH, ujson.dumps(DATA), str(id), cb)

def patch(PATH, DATATAG, bg=True, id=0, cb=None):
    if bg:
      _thread.start_new_thread(INTERNAL.patch, [PATH, ujson.dumps(DATATAG), str(
   id), cb])
    else:
      INTERNAL.patch(PATH, ujson.dumps(DATATAG), str(id), cb)

def getfile(PATH, FILE, bg=False, id=0, cb=None, limit=False):
    if bg:
      _thread.start_new_thread(INTERNAL.getfile, [PATH, FILE, bg, str(id), cb,
   limit])
    else:
      INTERNAL.getfile(PATH, FILE, bg, str(id), cb, limit)

def get(PATH, DUMP, bg=False, cb=None, id=0, limit=False):
```

```
    if bg:
      _thread.start_new_thread(INTERNAL.get, [PATH, DUMP, str(id), cb, limit])
    else:
      INTERNAL.get(PATH, DUMP, str(id), cb, limit)

def delete(PATH, bg=True, id=0, cb=None):
    if bg:
      _thread.start_new_thread(INTERNAL.delete, [PATH, str(id), cb])
    else:
      INTERNAL.delete(PATH, str(id), cb)

def addto(PATH, DATA, DUMP=None, bg=True, id=0, cb=None):
    if bg:
      _thread.start_new_thread(INTERNAL.addto, [PATH, ujson.dumps(DATA), DUMP,
   str(id), cb])
    else:
      INTERNAL.addto(PATH, ujson.dumps(DATA), DUMP, str(id), cb)
```

En este apartado se modificaron las funciones anteriormente explicadas para que se pueda acoplar de una manera más efectiva al proyecto

```
def get_wifi_ssid():
    return "Tenda Flamin' Hot"

def get_wifi_password():
    return "34751725"

def get_base_de_datos():
    return "https://cripto-project-5f94c-default-rtdb.firebaseio.com/"

def get_centro_de_control():
    return "https://receiving-hashes-default-rtdb.firebaseio.com/"
```

Finalmente, en esta sección lo que se hizo fue crear las fuciones "get_wifi_ssidz "get_wifi_password"para poder obtener la información del internet con el que se va a conectar la placa ESP. Por otro lado, las funciones "get_base_de_datosz "get_centro_de_control.ªyudan a que pueda haber una conexión directa a la base de datos proporcionada por el socio formador y al centro de control que ayudará a guardar los datos hasheados.

2.11. Consumo

En este código lo que se hace es identificar en la base de datos cuáles datos corresponden a si son producción o consumo. Una vez habiendo hecho esto filtrado, lógicamente se usarán los datos clasificados como Consumo. Ya que se tienen estos datos se les aplicará un proceso de hasheo para que estos su integridad no sea vulnerada.

```
wlan = network.WLAN(network.STA_IF)
```

```
wlan.active(True)
if not wlan.isconnected():
    wlan.connect(firebase.get_wifi_ssid(), firebase.get_wifi_password())
    print("Waiting for Wi-Fi connection", end="...")
    while not wlan.isconnected():
        print(".", end="")
        time.sleep(1)
    print()

wlan_sta = network.WLAN(network.STA_IF)
wlan_sta.active(True)
wlan_mac = wlan_sta.config('mac')
MAC_add = ubinascii.hexlify(wlan_mac).decode()

pk = ecdsa.get_private_key()
Ux, Uy = ecdsa.mult_binaria(ecdsa.get_gx(), ecdsa.get_gy(), ecdsa.get_a(), ecdsa.
    get_p(), pk)
firebase.setURL(firebase.get_centro_de_control())
firebase.put("Public key", {"Ux" : hex(Ux), "Uy" : hex(Uy) }, bg=0)
```

El primer paso en este proceso es realizar la conexión a WiFi con ayuda del repositorio Firebase Auth Implementation for Micropython en la plataforma GitHub.

```
for j in range(364):

    firebase.setURL(firebase.get_base_de_datos())

    firebase.get('Cons_'+str(j), "var1", bg=0)
    current_day = firebase.var1

    cont = 0

    firebase.setURL(firebase.get_centro_de_control())
    print('----- DAY ' + str(j+1) + ' -----')

    for i in range(1,97):

        curr_str = "{}-{}-{} {:02d}:{:02d}/{}_{}={}".format(current_day['Anio'],
    current_day['Mes '], current_day['Dia'], int(cont/60), cont%60, current_day['
    ID'], current_day['ConsProd'], current_day[str(i)])
        curr_str = MAC_add + " " + curr_str
        hash_object = hashlib.sha256()
        hash_object.update(curr_str)
        hexa = binascii.hexlify(hash_object.digest())

        r, s = ecdsa.sign_message(curr_str,pk)

        data = {
        "Dato": curr_str,
```

```
        "hash": hexa,
        "Key": {'r': hex(r),'s': hex(s)}
        }

        public_id = "Cons_" + str(j+1) + '_' + str(i)

        firebase.put(public_id, data, bg=0)
        cont +=15
```

Una vez establecida la conexión se genera el bucle por el cual se accederá posteriormente a los 364 datos disponibles para trazas de consumo. En cada uno de estos ciclos se fijan las credenciales para acceder a la base de datos originales y se realiza la petición del dato a la base de datos. Es importante mencionar que el contador, establecido en 0, así como la actualización de su valor ayudará a mantener registros sobre la hora de los datos. A continuación se fijan las credenciales para acceder al centro de control, y se genera otro bucle para acceder a los 96 datos disponibles de cada uno de los 364 anteriormente mencionados. En estos nuevos ciclos se crea el ID del dato específico y se crea un objeto de tipo hashlib.sha256() para hashear el ID. Posteriormente se realiza el firmado con ECDSA para así crear el paquete de datos a enviar al centro de control, el public ID del paquete de datos y finalmente realizar la petición para escribir datos sobre el centro de control.

2.12. Producción

Este código es bastante parecido al que se describió anteriormente en el proceso de Consumo.

Primero se puede apreciar que se establece una conexión WiFi con la ayuda del repositorio de GitHub antes mencionado.

```
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
if not wlan.isconnected():
    wlan.connect(firebase.get_wifi_ssid(), firebase.get_wifi_password())
    print("Waiting for Wi-Fi connection", end="...")
    while not wlan.isconnected():
        print(".", end="")
        time.sleep(1)
    print()

wlan_sta = network.WLAN(network.STA_IF)
wlan_sta.active(True)
wlan_mac = wlan_sta.config('mac')
MAC_add = ubinascii.hexlify(wlan_mac).decode()

pk = ecdsa.get_private_key()
Ux, Uy = ecdsa.mult_binaria(ecdsa.get_gx(), ecdsa.get_gy(), ecdsa.get_a(), ecdsa.
   get_p(), pk)
firebase.setURL(firebase.get_centro_de_control())
```

```
firebase.put("Public key", {"Ux" : hex(Ux), "Uy" : hex(Uy) }, bg=0)
```

Posteriormente se realiza el mismo procedimiento que se describe en la sección anterior de Consumo, pero en esta ocasión se filtran los datos de Producción y de nuevo se hace el hasheo de dichos datos.

```
for j in range(364):

    firebase.setURL(firebase.get_base_de_datos())

    firebase.get('Prod_+str(j), "var1", bg=0)
    current_day = firebase.var1

    cont = 0

    firebase.setURL(firebase.get_centro_de_control())
    print('----- DAY ' + str(j+1) + ' -----')

    for i in range(1,97):

        curr_str = "{}-{}-{} {:02d}:{:02d}/{}_{}={}".format(current_day['Anio'],
   current_day['Mes '], current_day['Dia'], int(cont/60), cont%60, current_day['
   ID'], current_day['ConsProd'], current_day[str(i)])
        curr_str = MAC_add + " " + curr_str
        hash_object = hashlib.sha256()
        hash_object.update(curr_str)
        hexa = binascii.hexlify(hash_object.digest())

        r, s = ecdsa.sign_message(curr_str,pk)

        data = {
        "Dato": curr_str,
        "hash": hexa,
        "Key": {'r': hex(r),'s': hex(s)}
        }

        public_id = "Cons_" + str(j+1) + '_' + str(i)

        firebase.put(public_id, data, bg=0)
        cont +=15
```

3. Centro de Control

3.1. Verificación de firma

Este apartado se puede revisar en la sección de ECDSA, este proceso tiene un rol importante en el centro de control ya que se verificaría la integridad de los datos y que la información que se

recibe sea congruente con la que se envía.

3.2. Centro de control

El contenido de este archivo de texto es un URL que permite hacer una conexión más rápida y eficaz a la firebase.

3.3. Credenciales bases de datos

En los archivos del proyecto se puede encontrar un documento tipo JSON llamado receiving-hashes-credentials el cual contiene las credenciales para poder acceder de manera segura al firebase y de esta forma cualquier persona autorizada puede acceder a este y de igual manera se protege ante quien no tiene autorización.

4. Montaje de Base de Datos y Almacenamiento de Datos

Se accedió a la herramienta de firebase desde el siguiente link: **https://console.firebase.google.com/**, seguido de esto se creó un proyecto y se accedió a la herramienta Realtime Database, después se creó la base de datos en test mode, para así poder fácilmente acceder a los datos para los primeros prototipos de arquitectura de red, ya una vez creada la base de datos se importaron los archivos json generados a la base de datos, como se muestra en la Figura 1.

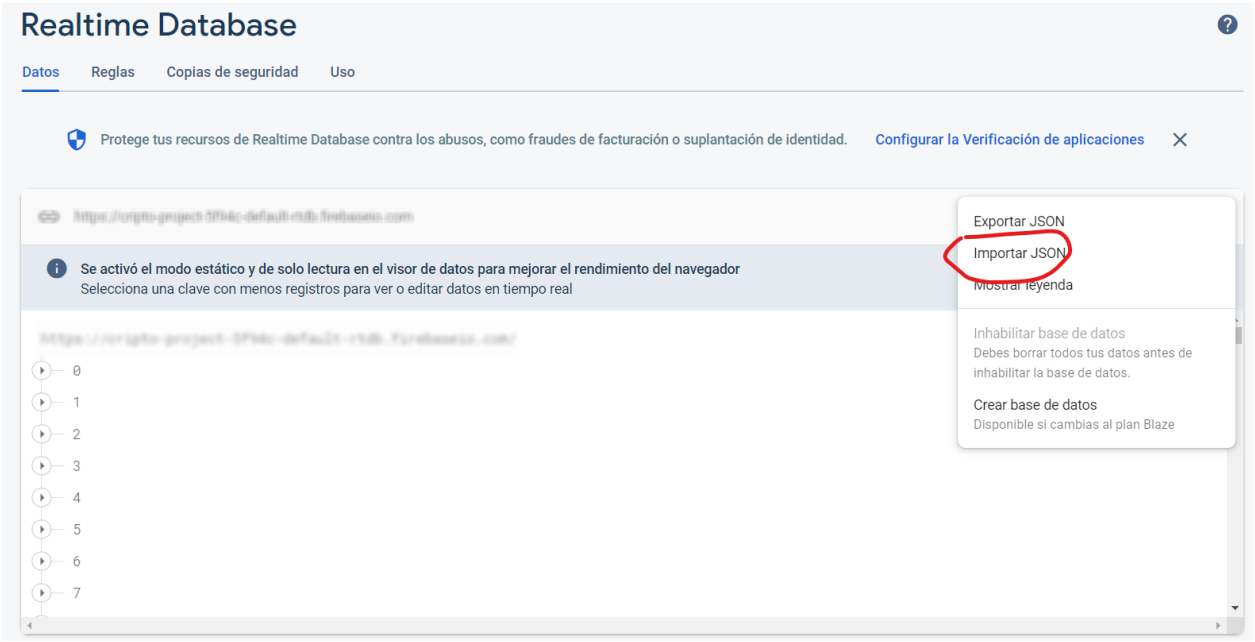

Figura 1: Opción para importar bases de datos a firebase.

Debido a que esta base de datos ya había sido previamente cargada con sus respectivos datos, aparecen los índices de los datos en la imagen, sin embargo, si es la primera vez que se cargan los datos en la herramienta, la consola aparecería vacía.

Cabe mencionar que los datos únicamente pueden ser accedidos desde la cuenta que creó la base de datos.

Una vez importadas ambas bases de datos a firebase se pudo proceder a montar el centro de control.

4.1. Montaje de Segunda Base de Datos

El montaje de esta base es muy similar al paso anterior debido a que se utiliza de igual forma firebase para almacenar los paquetes de datos transformados, la mayor diferencia entre ambas bases de datos es que a la base de datos que contiene los datos originales, se le hacen requests de datos, mientras que en el centro de control se escriben los datos transformados.

Para mostrar de mejor forma que la conexión entre auditores y bases de datos pueden ser totalmente remotas independientemente de la red a la que se este conectado alguno de los componentes, la segunda base se creó en otra cuenta diferente a la cuenta con la que se creó la base de los datos originales, sin embargo un usuario de una cuenta no puede modificar la base de datos creada con la otra cuenta y viceversa.

La segunda base de datos contará con un total de 728 datos, todos en formato json, donde cada dato contendrá la cadena de datos a asegurar, así como su hasheo y datos de firmado, debido a que el proyecto busca ilustrar el funcionamiento correcto de los algoritmos de hasheo y autenticación, en la base de datos de centro de control se incluye la string de datos original por si se quisiese comprobar o hacer cambios en los algoritmos de ciberseguridad en un futuro.

Referencias

[1] Elliptic curve cryptography: a gentle introduction. https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/. (Accessed: 02/03/2023).

[2] Repositorio del proyecto de la materia ma2006b. https://github.com/pablormzsantaellau/InvCripto3. (Accessed: 02/06/2023).

[3] Standard curve database. https://neuromancer.sk/std/nist/P-256. (Accessed: 02/03/2023).

[4] Time access and conversions. https://docs.python.org/3/library/time.html/, 2018. (Accessed: 04/07/2023).

[5] Python-firebase. https://pypi.org/project/python-firebase/, 2019. (Accessed: 04/07/2023).

[6] Secure hashes and message digests. https://docs.python.org/3/library/hashlib.html, 2019. (Accessed: 04/07/2023).

[7] Convert between binary and ascii. https://docs.python.org/3/library/binascii.html, 2020. (Accessed: 04/07/2023).

[8] Miscellaneous operating system interfaces. https://docs.python.org/3/library/os.html, 2022. (Accessed: 04/07/2023).

[9] Network basics. https://docs.micropython.org/en/latest/esp8266/tutorial/network$_b$*asics.html*/, 2022.(*Acc* 04/07/2023).

[10] Micropython external c modules¶. https://docs.micropython.org/en/latest/develop/cmodules.html, 2023. (Accessed: 02/03/2023).

[11] Privacidad y seguridad en firebase. https://firebase.google.com/support/privacy?hl=es-419, 2023. (Accessed: 02/07/2023).

[12] V. Dubey. Firebase micropython esp-32. https://github.com/vishal-android-freak/firebase-micropython-esp32, 2019. (Accessed: 02/05/2023).

[13] D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, 2001.