Analisis Penggunaan *Elliptic Curve Digital Signature Algorithm* (ECDSA) dan SHA-256 dalam Keamanan Firmware Update pada Perangkat IoT

Gratia Nindyaratri, 18220017 (*Author*)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : gratia.nindya@gmail.com

***Abstract*—Teknologi IoT semakin berkembang dan diadaptasi secara luas di masyarakat, baik oleh rumah tangga maupun industri yang canggih. Dalam pemanfaatan IoT, tantangan keamanan perlu diperhatikan. Salah satu ancaman keamanan yang perlu diatasi adalah ancaman keamanan dalam proses *firmware update* perangkat IoT. Ancaman tersebut kebanyakan berupa ancaman *integrity* dan *authentication*. Oleh karena itu, kriptografi, terutama proses tanda tangan digital dengan *hashing* dipilih menjadi solusi yang sesuai karena layanan yang diberikannya tepat dengan kebutuhan. Pemilihan algoritma ECDSA dan SHA-256 juga dipilih atas pertimbangan kebutuhan performansi dan standardisasi untuk sistem IoT. Oleh karena itu, dibangun sebuah skema pengamanan *firmware update* dengan algoritma ECDSA dan SHA-256 dan program simulasinya menggunakan Python. Berdasarkan implementasi dan studi literatur tersebut, dapat disimpulkan bahwa ECDSA dan SHA-256 merupakan algoritma yang dapat diterapkan untuk mengamankan proses *firmware update* perangkat IoT.**

Keywords—IoT; firmware update; tanda tangan digital; kriptografi; hash; Elliptic Curve Digital Signature Algorithm; SHA-256

I. PENDAHULUAN

Penggunaan teknologi Internet of Things atau IoT semakin meluas dan menjadi salah satu teknologi yang paling penting dalam beberapa tahun ke belakang. Pemanfaatannya yang luas dapat menyentuh teknologi yang digunakan oleh lapisan masyarakat awam dalam benda-benda yang digunakan sehari-hari, sehingga teknologi ini juga populer tidak hanya di kalangan ahli dan industri tetapi juga dikenal oleh masyarakat luas. Dengan adanya IoT, dapat terjadi hubungan antara objek-objek fisik dengan internet dalam bentuk perangkat *embedded* sehingga tercipta komunikasi antara *people*, *process*, dan *things* [1].

Dalam pemanfaatan teknologi IoT, terdapat banyak ancaman keamanan. Salah satu kerentanan dalam keamanan IoT adalah kurangnya mekanisme yang aman dalam melakukan *firmware update* [3]. Terdapat ancaman keamanan yang dapat membahayakan proses *firmware update*, seperti kurangnya proses validasi *firmware* dalam perangkat, keamanan dalam proses pengiriman, mekanisme *anti-rollback*, dan notifikasi perubahan keamanan karena *update* [2]. Salah satu usaha yang dapat dilakukan dalam mengatasi celah keamanan IoT adalah pemanfaatan teknologi kriptografi untuk memberikan layanan keamanan berupa *confidentiality*, *data integrity*, *authentication*, dan *nonrepudiation.* Pada makalah ini, akan ditelusuri penggunaan teknologi kriptografi dalam menjaga integritas dan autentikasi *firmware update* dan implementasinya dengan menggunakan algoritma *Elliptic Curve Cryptography* (ECC).

II. METODE PENELITIAN

A. Studi Literatur

Studi literatur dilakukan untuk mendapatkan informasi yang dibutuhkan dalam penyusunan makalah. Informasi yang digunakan dapat diambil dari *website*, publikasi, buku, atau materi ajar yang tersedia di internet. Informasi yang akan dicari mencakup Internet of Things (IoT), kriptografi, algoritma kriptografi seperti *Elliptic Curve Cryptography* (ECC) dan SHA-256, dan pemanfaatan algoritma kriptografi tersebut yaitu *Elliptic Curve Digital Signature Algorithm* (ECDSA).

B. Eksperimen

Eksperimen dilakukan berdasarkan studi literatur yang telah dilakukan, yaitu dengan memanfaatkan informasi untuk membangun skema penggunaan ECDSA dan SHA-256 untuk mengamankan proses *firmware update.* Skema akan dikembangkan menjadi program simulasi skema ECDSA dan SHA-256 menggunakan bahasa pemrograman Python dan *library* yang tersedia di publik.

III. DASAR TEORI

A. Internet of Things (IoT)

Internet of Things atau IoT adalah jaringan kolektif dari perangkat berupa objek fisik yang tertanam *(embedded)* dengan sensor, software, dan teknologi lainnya agar saling terhubung dan dapat bertukar data antar perangkat dan sistem melalui internet. Perangkat ini termasuk objek rumah tangga yang umum hingga alat industri yang canggih dan mereka semua

dapat terhubung ke internet karena munculnya teknologi cip komputer yang murah dan telekomunikasi dengan *bandwidth* yang rendah [1][4].

Sistem IoT secara umum bekerja dengan melakukan pengumpulan dan pertukaran data secara *realtime*. Terdapat tiga komponen dalam sebuah sistem IoT:

- Perangkat pintar

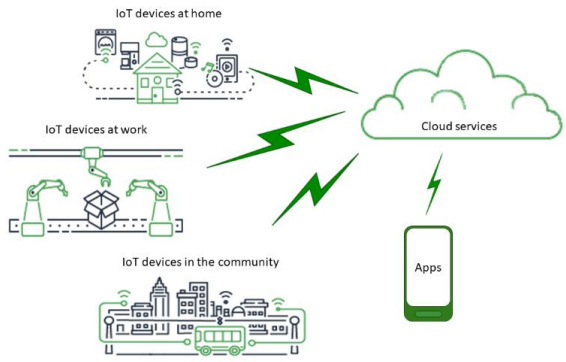
  Perangkat yang telah diberikan kapabilitas komputasi dan dapat berupa apapun seperti televisi, peralatan olahraga, dll. Perangkat ini mengumpulkan data dari lingkungan, input pengguna, atau pola dari penggunaan dan mengkomunikasikan data tersebut dengan aplikasinya lewat internet.

- Aplikasi IoT

  Aplikasi yang merupakan koleksi dari layanan dan perangkat lunak yang mengintegrasikan data yang diterima dari berbagai perangkat IoT. Aplikasi ini dapat menggunakan teknologi *machine learning* atau *artificial intelligence* untuk menganalisis data dan memberikan keputusan berdasarkan data. Keputusan ini dikomunikasikan kembali ke perangkat IoT dan perangkat akan memberikan respon pintar terhadap input tersebut.

- Antarmuka dengan pengguna

  Antarmuka yang dapat digunakan untuk melakukan pengelolaan terhadap perangkat IoT. Dapat berupa aplikasi *mobile* atau *website* yang dapat digunakan untuk mendaftarkan atau mengontrol perangkat pintar.



Gambar III.1 Sistem IoT
(sumber: https://aws.amazon.com/id/what-is/iot/)

Setelah melakukan peluncuran sistem IoT, perangkat IoT tetap memerlukan pemeliharaan konstan dan *firmware update* agar tetap modern dan bisa diandalkan. *Firmware update* biasanya berisi perbaikan *bug*, pembaruan keamanan, fitur baru, atau perubahan teknologi [5]. *Update* dapat dilakukan dengan berbagai cara dan salah satu cara yang populer digunakan adalah OTA (Over-The-Air) [6]. Dalam melakukan *update*, terdapat masalah keamanan yang dapat terjadi, seperti kurangnya proses validasi *firmware* dalam perangkat, keamanan dalam proses pengiriman, mekanisme *anti-rollback*, dan notifikasi perubahan keamanan karena *update* [2].

B. Kriptografi

Kriptografi adalah ilmu atau seni untuk menjaga keamanan pesan. Terdapat empat layanan yang disediakan dalam kriptografi, yaitu menjaga kerahasiaan pesan *(confidentiality/privacy/secrecy)*, keaslian pesan *(data integrity)*, keaslian pengirim dan penerima pesan *(authentication)*, dan anti penyangkalan *(non-repudiation)*.

Dalam kriptografi, pesan adalah data dan informasi yang selanjutnya akan disebut sebagai plainteks. Plainteks yang telah disandikan sehingga tidak bermakna lagi akan disebut chiperteks. Tujuannya agar pesan tidak dapat dibaca oleh pihak yang tidak memiliki hak. Untuk menyandikan plainteks menjadi cipherteks, dilakukan proses enkripsi. Sedangkan untuk mengembalikan cipherteks menjadi plainteks semula dapat dilakukan proses dekripsi. Agar enkripsi dan dekripsi hanya dapat dilakukan oleh pihak yang melakukan komunikasi, diperlukan kunci rahasia, yaitu parameter yang digunakan untuk proses enkripsi dan dekripsi. Proses ini adalah dasar dari kriptografi.

Terdapat tiga jenis algoritma kriptografi, yaitu algoritma kriptografi simteri, algoritma kriptografi nir-simetri (kriptografi kunci publik), dan fungsi hash. Pada algoritma kriptografi simetri, kunci enkripsi sama dengan kunci dekripsi sehingga harus dijaga agar tetap rahasia. Algoritma kriptografi nir-simetri memiliki kunci enkripsi yang berbeda dengan kunci dekripsi. Kunci enkripsi tidak rahasia dan kunci dekripsi dirahasiakan. Sedangkan pada fungsi hash, pesan dikompresi menjadi *message-digest* berukuran *fixed.* Pada fungsi hash, pesan yang sudah melalui fungsi hash tidak dapat dikembalikan menjadi pesan semula [7].

C. Fungsi Hash

Fungsi adalah sebuah fungsi yang mengkonversi sebuah pesan berukuran sembarang menjadi pesan ringkas *(message-digest)* atau nilai hash *(hash value)* dengan panjang yang tetap *(fixed).* Fungsi hash bersifat *irreversible* sehingga tidak bisaa dikembalikan menjadi pesan semula.

$$h = H(M) \tag{1}$$

Lebih lengkapnya, fungsi hash memiliki sifat-sifat sebagai berikut.

1. *Collision resistance:* sangat sulit untuk menemukan dua input *a* dan *b* sedemikian sehingga $H(a) = H(b)$
2. *Preimage resistance:* untuk sembarang output *y*, sulit untuk menemukan input *a* sedemikian sehingga $H(a) = y$
3. *Second preimage resistance:* untuk input *a* dan output $y = H(a)$ sulit untuk menemukan input kedua *b* sedemikian sehingga $H(b) = y$

Fungsi hash diaplikasikan dalam menjaga integritas pesan karena fungsi hash sangat peka terhadap perubahan pada pesan. Fungsi hash juga dapat menghemat waktu pengiriman karena dengan mengirimkan hanya *message digest* saja, dapat dilakukan perbandingan salinan arsip dengan yang dimiliki di master. Fungsi hash juga digunakan untuk menormalkan

panjang data yang beraneka ragam karena hasil nilai hash memiliki panjang yang tetap [7].

D. SHA-256

SHA-256 merupakan salah satu algoritma fungsi hash SHA versi SHA-2 yang didasarkan pada algoritma MD4 yang dibuat oleh Ronald L, Rivest [8]. SHA-256 sendiri dikembangkan oleh lembaga National Institute of Standards and Technology (NIST). Algoritma ini menerima input dengan panjang maksimal 264-1 bit dan memberikan output dengan panjang tetap yaitu 256bit. Langkah-langkah algoritma SHA-256 adalah sebagai berikut [13].

1. Menambahkan *padding* bit pada pesan awal sehingga panjangnya sesuai dengan kebutuhan fungsi hash. Penambahan bit dilakukan agar pesan memiliki panjang 64 bit kurang dari kelipatan 512
2. Menambahkan panjang bit pesan awal sepanjang 64 bit sehingga total panjang pesan menjadi tepat kelipatan 512.
3. Mempersiapkan *buffer* dengan nilai *default* sebagai berikut.

   a = 0x6a09e667

   b = 0xbb67ae85

   c = 0x3c6ef372

   d = 0xa54ff53a

   e = 0x510e527f

   f = 0x9b05688c

   g = 0x1f83d9ab

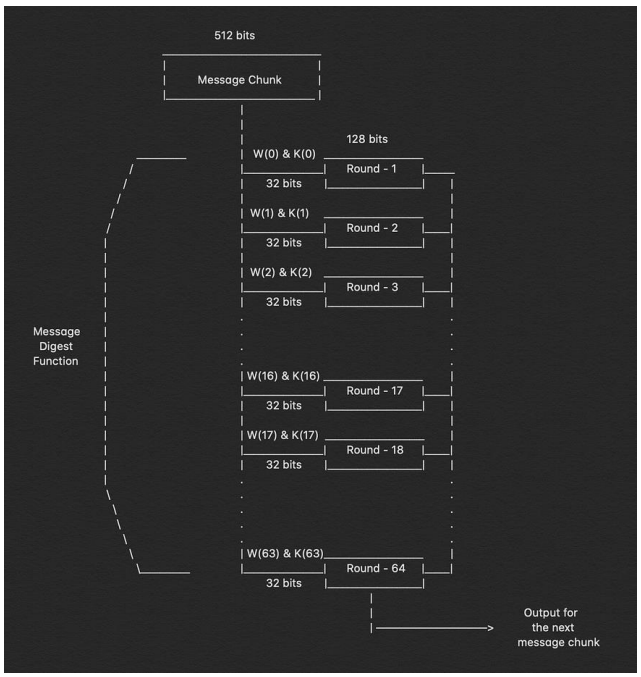
   h = 0x5be0cd19

   Selain itu, disiapkan 64 nilai yang akan menjadi kunci.

```
k[0..63] :=
   0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
   0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
   0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
   0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
   0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
   0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
   0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
   0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
```

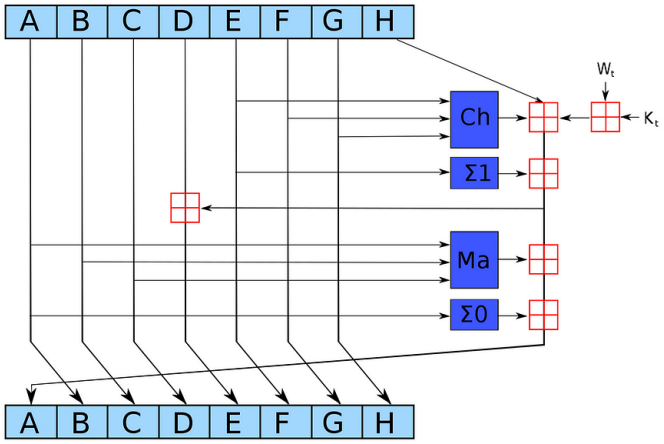
Gambar III.2 Key dalam putaran SHA-256
(sumber: https://infosecwriteups.com/breaking-down-sha-256-algorithm-2ce61d86f7a3)

4. Melakukan fungsi kompresi dengan membagi blok pesan sepanjang *n * 512* bit menjadi *n* buah *message chunk* 512 bit yang kemudian melewati 64 putaran operasi.



Gambar III.3 Putaran untuk tiap *message chunk*
(sumber: https://infosecwriteups.com/breaking-down-sha-256-algorithm-2ce61d86f7a3)

Oleh karena itu, setiap putaran akan ditunjukkan dalam gambar berikut.



Gambar III.4 Fungsi untuk tiap putaran
(sumber: https://infosecwriteups.com/breaking-down-sha-256-algorithm-2ce61d86f7a3)

Fungsi dasar yang dilakukan dalam setiap putaran adalah sebagai berikut.

```
Ch(E, F, G) = (E AND F) XOR ((NOT E) AND G)
Ma(A, B, C) = (A AND B) XOR (A AND C) XOR (B AND C)
    ∑(A)    = (A >>> 2) XOR (A >>> 13) XOR (A >>> 22)
    ∑(E)    = (E >>> 6) XOR (E >>> 11) XOR (E >>> 25)
     +      = addition modulo 2^32
```

Gambar III.5 Fungsi dasar dalam putaran SHA-256
(sumber: https://infosecwriteups.com/breaking-down-sha-256-algorithm-2ce61d86f7a3)

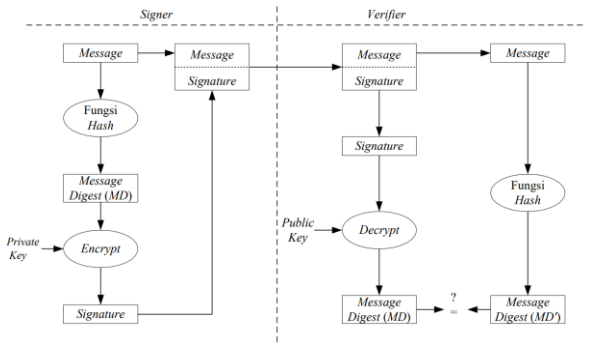
5. Output dari setiap putaran akan menjadi input untuk putaran selanjutnya hingga mencapai bit terakhir pesan. Hasil dari putaran terakhir akan menjadi hasil *hash* dari seluruh pesan yang memiliki panjnag 256 bit.

E. Digital Signature

Tanda tangan memiliki fungsi memberikan otentikasi pada dokumen. Tanda tangan memiliki karakteristik sebagai berikut.

1. Tanda tangan adalah bukti yang otentik
2. Tanda tangan tidak dapat dilupakan
3. Tanda tangan tidak dapat dipindah untuk digunakan ulang
4. Dokumen yang telah ditandatangani tidak dapat diubah
5. Tanda tangan tidak dapat disangkal

Tanda tangan digital adalah nilai kriptografis yang bergantung pada isi pesan dan kunci yang digunakan. Tanda tangan digital selalu berubah mengikuti pesan dan kunci, sedangkan tanda tangan pada dokumen cetak akan selalu sama untuk orang yang sama meskipun isi dokumen berbeda. Dalam tanda tangan digital, terdapat dua proses yaitu menandatangani pesan dan melakukan verifikasi tanda tangan. Proses penandatanganan ini dapat dilakukan dengan menggunakan kombinasi kriptografi kunci publik dan fungsi hash untuk menjaga *authenticity* dan *integrity* dari pesan [1].



Gambar III.6 Tanda Tangan Digital
(sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2022-2023/16-Tanda-tangan-digital-2023.pdf)

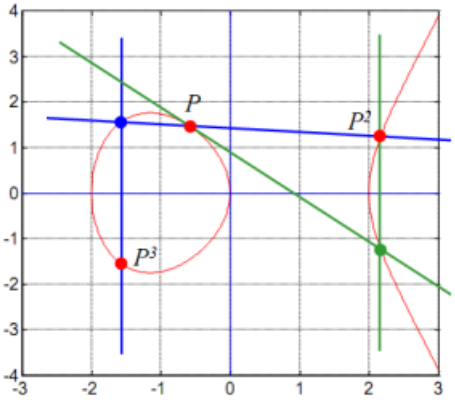
F. Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) adalah sebuah tipe dalam kriptografi kunci publik yang memanfaatkan konsep *elliptic curve*. Kriptografi ini diusulkan oleh Neal Koblitz dan Victor Miller pada tahun 1985. ECC memenuhi karakteristik kriptografi kunci publik yaitu dengan adanya *trap-door function*, di mana komputasi mudah dilakukan secara satu arah tetapi sulit didekripsi dari arah lainnya. Artinya, dekripsi mudah dilakukan jika kunci privat diketahui, tetapi akan sangat sulit silakukan jika kunci privat tidak diketahui [9].

Kriptografi kurva eliptik didasarkan pada konsep operasi aritmatika titik pada kurva eliptik. Sebuah kurva eliptik memiliki bentuk umum persamaan sebagai berikut.

$$y^2 = x^3 + ax + b \quad (2)$$

Berdasarkan persamaan tersebut, dapat dilakukan operasi penjumlahan titik. Tetapi, yang menjadi pokok dari penggunaan kurva eliptik pada kriptografi adalah pelelaran titik, yaitu menjumlahkan sebuah titik pada dirinya sendiri sebanyak k-1 kali.



Gambar III.7 Pelelaran Titik Kurva Eliptik
(sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2022-2023/12-ECC-2023.pdf)

Dari pelelaran titik, dapat disimpulkan bahwa konsepnya sama dengan persamaan perkalian titik sebagai berikut.

$$kP = Q \quad (3)$$

Persamaan tersebut kemudian menjadi konsep dari *Elliptic Curve Discrete Logarithm Problem* (EDLP) yang menjadi dasar dari ECC. Pada konsep ini, menghitung kP = Q akan mudah, tetapi menghitung k dari P dan Q akan sulit. Pada algoritma ECC, Q menjadi kunci publik, k adalah kunci privat, dan P adalah sembarang titik pada kurva eliptik. Pada ECC, dua pihak yang berkomunikasi akan menyepakati parameter data sebagai berikut:

1. Persamaan kurva eliptik $y^2 = x^3 + ax + b \bmod p$

   Nilai dari a dan b dan bilangan prima p.
2. Himpunan titik-titik yang dihitung dari persamaan kurva eliptik
3. Titik basis *(base point)* B, dipilih dari grup eliptik untuk melakukan operasi kriptografi.

Setiap pengguna juga membangkitkan sepasang kunci publik dan kunci privat. Kunci privat merupakan integer *x* yang dipilih dari selang *[1, p-1]* dan kunci publik adalah titik *Q* yang merupakan hasil kali antara *x* dan titik basis *B*. $Q = x \bullet B$ [7].

ECC menjadi pilihan yang populer dibanding RSA untuk keamanan sistem IoT karena beberapa alasan. ECC tidak

memerlukan angka yang sangat besar untuk menjaga keamanan. Energi yang diperlukan untuk memecahkan kunci ECC dengan panjang yang sama dengan kunci RSA juga jauh lebih besar, sehingga jauh lebih sulit ditembus. ECC juga memiliki kebutuhan data yang rendah sehingga kriptoporsesor yang digunakan akan lebih kecil, cepat, dan murah [10].

G. *Elliptic Curve Digital Signature Algorithm (ECDSA)*

Algoritma ECDSA adalah salah satu algoritma dari kriptografi kurva eliptik, yaitu menerapkan *Elliptic Cuver Discret Logarithm Problem* (ECDLP) pada penggunaan tanda tangan digital. Dalam algoritma ini, pihak yang memberikan tanda tangan digital akan mempunyai parameter domain kurva eliptik berupa $D = \{q, FR, a, b, G, n, h\}$ dan pasangan kunci rahasia d_A dan kunci publik Q_A. Pihak yang melakukan verifikasi tanda tangan akan mendapatkan dokumen D dan kunci publik Q_A. Proses dalam ECDSA adalah sebagai berikut [11].

1) Key Generation.

a) Memilih bilangan bulat acak d_A di antara $[1, n-1]$

b) Menghitung $Q_A = d_A \bullet G = (x_1, y_1)$

c) Kunci rahasia = d_A dan kunci publik Q_A

2) Signing

a) Memilih sebuah bilangan bulat random k yang memiliki nilai di antara $[1, n-1]$

b) Menghitung $Q_A = k \bullet G = (x_1, y_1)$ dan $r = x_1 \bmod n$, jika $r = 0$, maka kembali ke langkah 1

c) Menghitung $k^{-1} \bmod n$

d) Menghitung $e = H(m)$

e) Menghitung $s = k^{-1} (e + d_A \bullet r) \bmod n$

Tanda tangan pengirim untuk pesan m adalah (r, s)

3) Verifying

a) Melakukan verifikasi bahwa r dan s adalah bilangan bulat yang antara $[1, n-1]$

b) Menghitung $e = H(m)$

c) Menghitung $w = s^{-1} \bmod n$.

d) Menghitung $u_1 = ew \bmod n$ dan $u_2 = rw \bmod n$

e) Menghitung $X = u_1 \bullet G + u_2 \bullet Q_A = (x_1, y_1)$

f) Menghitung $v = x_1 \bmod n$

g) Menerima tanda tangan jika dan hanya jika $v = r$

IV. IMPLEMENTASI

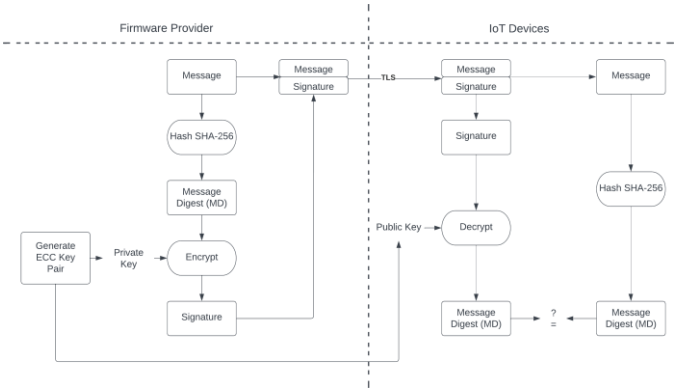
A. *Rancangan Solusi*

Dalam proses *update firmware*, salah satu bahaya yang dihadapi adalah memastikan bahwa *update package* yang dikirimkan berasal dari sumber yang valid *(authenticity)* dan konten yang terjamin keasliannya *(integrity)*. Oleh karena itu, mekanisme yang cocok untuk proses *firmware update* perangkat IoT adalah *digital signature* dan *hashing*.

Skema yang diusulkan dalam implementasi pengamanan *firmware update* adalah sebagai berikut.

1. Membangkitkan pasangan kunci ECDSA dengan kunci privat yang disimpan oleh pengelola *firmware* dan membagikan kunci publik pada perangkat IoT.
2. Melakukan proses *hashing* pada *firmware* menggunakan algoritma SHA-256.
3. Memberikan *digital signature* pada *firmware*.
4. Mengirimkan *firmware update* menggunakan TLS.
5. Melakukan verifikasi *authenticity* dan *integrity* dari *firmware update* yang diterima.

Skema tersebut dapat dijelaskan melalui diagram berikut.



Gambar IV.1 Diagram Skema Rancangan Solusi

B. *Desain Program*

Berdasarkan skema usulan yang dijelaskan, akan disusun program sederhana dengan bahasa Python yang dapat dijalankan pada terminal. Program ini akan mensimulasikan proses ECDSA dengan fungsi hash SHA-256. Untuk itu, disusun modul yang akan diimplementasikan dalam program simulasi. Modul-modul disusun dengan bersumber pada kode di internet dengan penyesuaian [12].

- Message Hashing

  Modul ini akan melakukan fungsi hash SHA-256 pada input *message* yang diberikan oleh pengguna. Pada situasi real, input yang diberikan akan berupa *firmware*.

```
def hash(message):
    return
int(hashlib.sha256(str(message).encode("utf
-8")).hexdigest(), 16)
```

- Generating Pair Key

  Modul ini akan membangkitkan pasangan kunci publik dan kunci privat menggunakan algoritma ECC.

```
private_key = random.getrandbits(256)
public_key = doubleAndAdd(base,
private_key, a, d, p)
```

- Signing

Modul ini akan membangkitkan *digital signature* dari *message* yang telah dikonversi oleh fungsi hash.

```
message = stringToInt(input("Message: "))
r = hash(hash(message) + message) % p
R = doubleAndAdd(base, r, a, d , p)
h = hash(R[0] + public_key[0] + message) %
p
s = r + h * private_key
```

- Verifying Sign

  Modul ini akan melakukan verifikasi dari *message* dan *digital signature* dari *message*.

```
verify_message = stringToInt(input("Message
to Verify: "))
xv = int(input("Public key (x): "))
yv = int(input("Public key (y): "))
public_key_verify = xv, yv
h = hash(R[0] + public_key_verify[0] +
verify_message) % p
p1 = doubleAndAdd(base, s, a, d, p)
p2 = addPoint(R,
doubleAndAdd(public_key_verify, h, a, d,
p), a, d, p)
if p1[0] == p2[0] and p1[1] == p2[1]:
    print("Signature valid")
else:
    print("Signature invalid")
```

- Utilities

  Fungsi yang membantu proses ECDSA.

```
# cast string to int
def stringToInt(string):
    return int(string.encode('utf-
8').hex(), 16)

# get positive modulus
def getPositiveModulus(a, p):
    if a<0:
        a = (a+p*int(abs(a)/p)+p)%p
    return a

# get gcd
def gcd(a,b):
    while a!=0:
        a,b = b%a, a
    return b

# get modular inverse of a mod b
def getModularInverse(a, b):

    a = getPositiveModulus(a, b)

    # if not relatively prime, no mod
inverse
    if gcd(a, b) != 1:
        return None

    # extended euclidean algorithm
    u1, u2, u3 = 1, 0, a
    v1, v2, v3 = 0, 1, b
    while v3 != 0:
        q = u3 // v3
        v1, v2, v3, u1, u2, u3 = (u1 - q *
v1), (u2 - q * v2), (u3 - q * v3), v1, v2,
v3
    return u1 % b

# calculation to add points
def addPoint(P, Q, a, d, mod):
    x1 = P[0]
    x2 = Q[0]
    y1 = P[1]
    y2 = Q[1]

    x3 = (((x1*y2 + y1*x2) % mod) *
getModularInverse(1+d*x1*x2*y1*y2, mod)) %
mod
    y3 = (((y1*y2 - a*x1*x2) % mod) *
getModularInverse(1- d*x1*x2*y1*y2, mod)) %
mod

    return x3, y3

# double and add function
def doubleAndAdd(P, k, a, d, mod):
    additionPoint = (P[0], P[1])

    # turn k to binary
    kBinary = bin(k)[2:len(bin(k))]

    for i in range(1, len(kBinary)):
```

```
        curr = kBinary[i:i+1]
        additionPoint =
addPoint(additionPoint, additionPoint, a,
d, mod)
        if curr == '1':
            additionPoint =
addPoint(additionPoint, P, a, d, mod)

    return additionPoint

# validate input
def validate_input(x):
    while True:
        try:
            user_input = input(x)
            if not user_input:
                raise ValueError("Input is
empty")
            return user_input
        except ValueError as e:
            print("Error: ", str(e))

# ECDSA curve ed25519
p = pow(2, 255) - 19
base =
151122213495354007725011514095885315114540126930418572060461132839498477622 02,
463168356949264781694283940034751631413079938662562256157830336031652518559 60
a = -1
d = getPositiveModulus(-121665 *
getModularInverse(121666, p), p)
```

Library yang digunakan adalah *hashlib* untuk melakukan fungsi hash SHA-256 dan modul *random* untuk membangkitkan angka secara acak. Dalam pemilihan *curve* untuk ECC, dipilih ed25519 yang menjadi standar dalam ECDSA.

V. Hasil dan Pembahasan

A. Hasil Pengujian

Setelah menyusun program simulasi, dilakukan pengujian untuk melihat apakah program sudah berjalan sesuai spesifikasi fungsionalnya yang sesuai skema.

TABLE I. Pengujian Pembangkitan Pasangan Kunci

No.	Kasus Uji	Input	Expected Output	Output
1.	Pembangkitan pasangan kunci	Memulai program	Pasangan kunci dibangkitkan, kunci publik ditampilkan	Pasangan kunci dibangkitkan, kunci publik ditampilkan

Dari hasil pengujian, kunci sudah berhasil dibangkitkan dan kunci publik ditampilkan. Untuk dokumentasi pengujian dapat dilihat pada gambar berikut.

```
D:\itb\SEM 6\tubes kripto makalah>C:/Users/gratia/AppDat
a/Local/Microsoft/WindowsApps/python3.8.exe "d:/itb/SEM
6/tubes kripto makalah/ecdsa_coba.py"
-----START-----
===Key Generation curve ed25519===
Public key:  (231487023128548233675195771946733069993855
081250062769518568161544651089184 28, 243859571254123850 5
80978666440175560342806549143100535355735100787844554780
4)
```

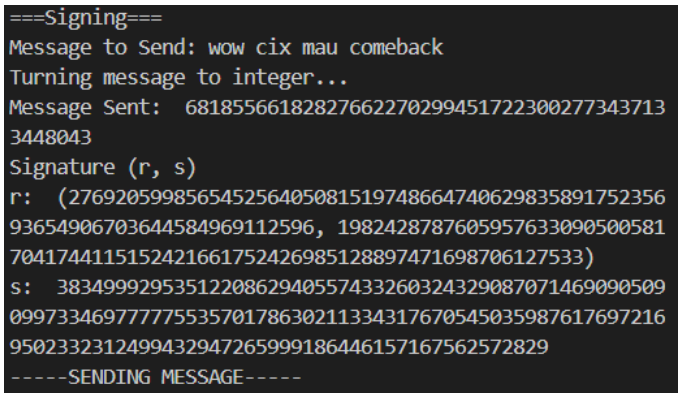
Gambar IV.2 Hasil Pengujian 1 Pembangkitan Pasangan Kunci

TABLE II. Pengujian Pembangkitan Tanda Tangan Digital

No.	Kasus Uji	Input	Expected Output	Output
1.	Input Message dimasukkan	Message to Send: wow cix mau comeback	Tanda tangan dibangkitkan	Tanda tangan dibangkitkan Signature (r, s) r: (27692059985654525640508151974866474062983589175235693654906703644584969112596, 19824287876059576330905005810997334697777755357017863021133431767054503598617697216950233231249432947265991864461571675625728297041744115152421661752426985128897471698706127533) s: 3834999295351220862940557433260324329080707146909050909973346977777553570178630211334317670545035985617697216950233231249943294726599918644615716756257 2829
2.	Input Message tidak dimasukkan	Message to Send:	Tanda tangan belum dibangkitkan dan muncul *error message*	Tanda tangan belum dibangkitkan dan muncul *error message*

Dari hasil pengujian, dapat disimpulkan bahwa pembangkitan tanda tangan digital akan dibangkitkan jika

sudah ada input dan gagal jika belum ada input. Untuk dokumentasi pengujian dapat dilihat pada gambar berikut.



Gambar IV.3 Hasil Pengujian 1 Pembangkitan Tanda Tangan Digital

Gambar IV.4 Hasil Pengujian 2 Pembangkitan Tanda Tangan Digital

TABLE III. PENGUJIAN VERIFIKASI TANDA TANGAN DIGITAL

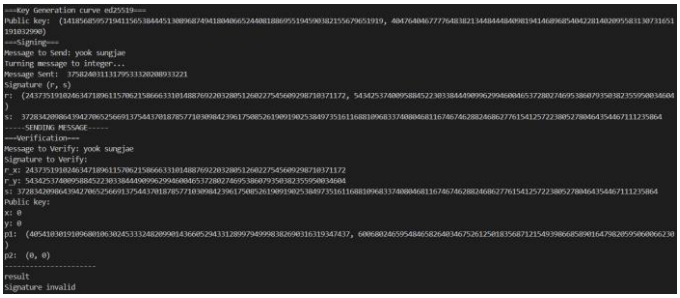
No.	Kasus Uji	Input	Expected Output	Output
1.	Seluruh input valid	Message to Verify: wow cix mau comeback Signature (r, s) r: (27692059985654525640508151974866474062983589175235693654906703644584969112596, 19824287876059576330905005817041744115152421661752426985128897471698706127533) s: 26639900766017089453859351873979571862386164657037002686169841109603493467057410977505421718112517937885977547246938060895406325737750202116772047664 12 public key: (51366757248678328494032772794333650770941850052042118845216756989206787243943, 379832899787816404769558032576283894319530423393216054698033802800940009 69712)	Pesan *signature* valid	Pesan signature valid
2.	Input	Message to Verify:	Pesan	Pesan
3.	Kunci publik tidak sesuai	Message to Verify: yook sungjae Signature to Verify: r_x: 243735191024634718961157062158666331014887692203280512602275456092987103711 72 r_y: 5434253740095884522303384449099629946004653728027469538607935038235595003460 4 s: 372834209864394270652566913754437018785771030984239617508526190919025384973516116881096833740804681167467462882468627761541257223805278046435446 7111235864 Public key: x: 0 y: 0	Pesan signature invalid	Pesan signature invalid
4.	Tanda tangan digital tidak sesuai	Message to Verify: hehe Signature to Verify: r_x: 2 r_y: 2 s: 2 Public key:	Pesan signature invalid	Pesan signature invalid
	message berubah	wow cix gamau comeback R_x: 276920599856545256405081519748664740629835891752356549067036445849691125 96 R_y: 198242878760595763309050058170417441151524216617524269851288974716987061275 33 s: 383499929535122086294055743326032432908707146909050909973346977777553570178630211334317670545035987617697216950233231249943294726599918644615716756257282 9 public key: (23148702312854823367519577194673306999385508125006276951856816154465108918428, 243859571254123850580978666440175560342806549143100535355735100787844554 7804)	signature invalid	signature invalid

No.	Kasus Uji	Input	Expected Output	Output
		x: 129319029244178109642160791650763434772666628244034910098212323019987471275 98 y: 168411350956164650871534535792979895649513470518477142786797400471547397 4199		
5.	Salah satu input kosong	Message to Verify: hehe Signature to Verify: r_x:	Verifikasi belum dilakukan dan muncul *error message*	Verifikasi belum dilakukan dan muncul *error message*

Dari hasil pengujian, dapat disimpulkan bahwa verifikasi tanda tangan digital sudah dilakukan sesuai input yang diberikan. Jika input cocok, maka akan muncul pesan bahwa tanda tangan digital tersebut valid. Jika input tidak cocok, maka pesan bahwa tanda tangan digital tersebut invalid akan muncul. Untuk dokumentasi pengujian dapat dilihat pada gambar berikut.

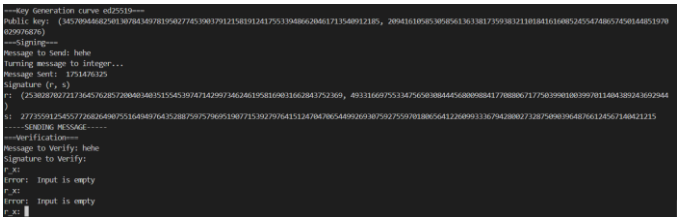
Gambar IV.5 Hasil Pengujian 1 Verifikasi Tanda Tangan Digital

Gambar IV.6 Hasil Pengujian 2 Verifikasi Tanda Tangan Digital



Gambar IV.7 Hasil Pengujian 3 Verifikasi Tanda Tangan Digital

Gambar IV.8 Hasil Pengujian 4 Verifikasi Tanda Tangan Digital



Gambar IV.9 Hasil Pengujian 5 Verifikasi Tanda Tangan Digital

B. Pembahasan

Berdasarkan hasil pengujian program simulasi, ditemukan bahwa program sudah berhasil memenuhi spesifikasi umum untuk setiap modul karena berhasil melakukan kedua fungsi utama yaitu *signing* dan *verifying.* Dengan melakukan kedua fungsi ini, berarti *hashing* yang dilakukan sudah memberikan output yang sesuai dan *pair key* yang dibangkitkan berhasil melakukan enkripsi dan dekripsi yang kompatibel.

Berdasarkan performa, hasil yang diberikan juga sudah sesuai. Hal ini mendukung kecocokkan pemilihan algoritma ECDSA dan SHA-256 yang kini menjadi standar dalam keamanan sistem IoT. ECDSA dipilih di atas RSA karena efisiensinya. ECDSA memiliki ukuran kunci yang lebih kecil dari RSA sehingga sesuai untuk perangkat IoT yang memiliki keterbatasan *computational power*, *memory*, dan *bandwidth.* Oleh karena itu, komputasi dapat dilakukan dengan lebih cepat dan lebih hemat. Keamanan yang ditawarkan oleh ECDSA juga sudah tinggi dan membutuhkan energi yang sangat besar untuk bisa memecahkannya.

Untuk fungsi hash, SHA-256 dipilih dibandingkan SHA-3 yang merupakan teknologi yang lebih baru. Pemilihan ini disebabkan oleh beberapa keunggulan SHA-256 dibandingkan

SHA-3 dalam kecocokkannya untuk diaplikasikan ke sistem IoT. SHA-256 sudah ada sebelum SHA-3 dan sudah melalui lebih banyak analisis dan serangan untuk membuktikan keamanannya. SHA-256 juga tergolong lebih efisien dalam penggunaan sumber daya komputassional sehingga cocok dengan batasan kapabilitas perangkat IoT. Selain itu, SHA-256 sudah merupakan standar yang diterima secara luas sehingga memudahkan interoperabilitas dan kompabilitas yang juga memudahkan integrasi dengan infrastruktur IoT yang sudah ada.

Dengan pemilihan algoritma yang sesuai untuk keamanan sistem IoT secara umum, algoritma ini juga sesuai untuk *firmware update* sistem IoT karena akan bekerja pada perangkat IoT. Konsiderasi yang utama (selain keamanan) dalam pengamanan adalah performa, karena perangkat IoT biasanya memiliki batasan dalam kapabilitas komputasinya seperti *processing power, memory,* dan *energy.* Standardisasi juga menjadi konsiderasi karena akan memudahkan interoperabilitas, kompatibilitas, dan integrasi dengan sistem dan protokol yang ada. Standardisasi dapat mengacu pada lembaga seperti NIST (National Institute of Standards and Technology) atau ISO (International Organization for Standardization). Oleh karena itu, pemilihan algoritma harus disusun dengan memperhatikan beberapa konsiderasi ini.

VI. KESIMPULAN DAN SARAN

Berdasarkan hasil pengujian dan pembahasan, telah disusun program simulasi yang dapat mendemonstrasikan penggunaan ECDSA dan SHA-256 untuk keamanan sistem IoT, dalam kasus ini untuk keamanan *firmware update.* Program jyga telah memiliki performa yang sesuai. Algoritma ECDSA dan fungsi hash SHA-256 juga telah dipilih berdasarkan konsiderasi yang dianggap paling penting yaitu keamanan, performa, dan standardisasi.

Untuk penelitian selanjutnya, dapat dikembangkan batasan penelitian seperti menganalisis tipe *firmware update,* komunikasi yang aman melalui TLS, atau menjamin keaslian kunci publik dengan *digital certificate* dan *public key infrastructure* (PKI). Dapat disusun juga usulan perubahan yang sesuai dengan kebutuhan dan konsiderasi dalam melakukan *firmware update* perangkat IoT.

UCAPAN TERIMA KASIH

Terima kasih kepada Tuhan Yang Maha Esa karena atas kasih karunia-Nya penulis dapat menyelesaikan makalah ini sebagai tugas mata kuliah II4031 Kriptografi dan Koding. Penulis juga ingin mengucapkan terima kasih kepada keluarga yang senantiasa mendukung dan mendoakan dalam perjalanan penulis mencari ilmu. Penulis juga ingin berterima kasih kepada Bapak Rinaldi Munir selaku dosen mata kuliah II4031 Kriptografi dan Koding yang telah memberikan ilmu dan kesempatan sehingga penulis dapat menulis makalah ini. Selain itu, penulis mengucapkan terima kasih kepada rekan-rekan satu jurusan dan satu kelas yang telah bersama saling mendukung dalam menempuh perkuliahan di mata kuliah ini. Semoga hasil makalah ini dapat memberikan manfaat.

REFERENCES

[1] Oracle, “What is the internet of things (IOT)?,” Oracle Indonesia, https://www.oracle.com/id/internet-of-things/what-is-iot/ (accessed May 21, 2023).

[2] The OWASP IoT Security Team, “OWASP IOT top 10,” OWASP, https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf (accessed May 21, 2023).

[3] R. Sanders, “The key to firmware security in connected IOT devices,” Keyfactor, https://www.keyfactor.com/blog/firmware-security-iot-devices/ (accessed May 21, 2023).

[4] [1] AWS, “Apa itu IoT?,” Amazon, https://aws.amazon.com/id/what-is/iot/ (accessed May 22, 2023).

[5] M. Bosson, “Helpful tips for IOT device updates,” Onomondo, https://onomondo.com/blog/iot-device-update-tips (accessed May 21, 2023).

[6] Koen Zandberg, Kaspar Schleiser, Francisco Acosta, Hannes Tschofenig, Emmanuel Baccelli. Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check. IEEE Access, 2019, 7, pp.71907-71920. ff10.1109/ACCESS.2019.2919760ff. ffhal-02351794

[7] Munir, R. “Bahan Kuliah II4031 Kriptografi dan Koding”, Program Studi Sistem dan Teknologi Informasi ITB, 2023

[8] S. Sulastri and R. D. Putri, “Implementasi ENKRIPSI data secure hash algorithm (SHA-256) Dan Message Digest Algorithm (MD5) pada proses Pengamanan Kata Sandi Sistem Penjadwalan Karyawan,” *Jurnal Teknik Elektro*, vol. 10, no. 2, pp. 70–74, 2018. doi:10.15294/jte.v10i2.18628

[9] Grayblock, “Elliptic-Curve Cryptography,” Medium, https://medium.com/coinmonks/elliptic-curve-cryptography-6de8fc748b8b (accessed May 22, 2023).

[10] P. Grubbs, “Why ECC is the solution for IOT Security,” SecureW2, https://www.securew2.com/blog/ecc-solution-iot-security (accessed May 22, 2023).

[11] Triwinarko, A. “Elliptic Curve Digital Signature Algorithm (ECDSA)”, Departemen Teknik Informatika ITB, 2005. https://informatika.stei.itb.ac.id/~rinaldi.munir/TA/Makalah_TA%20Andy%20T.pdf

[12] “Blockchain - Elliptic Curve Digital Signature Algorithm (ECDSA),” GeeksforGeeks, https://www.geeksforgeeks.org/blockchain-elliptic-curve-digital-signature-algorithm-ecdsa/ (accessed May 22, 2023).

[13] A. Anand, “Breaking Down : Sha-256 Algorithm,” Medium, https://infosecwriteups.com/breaking-down-sha-256-algorithm-2ce61d86f7a3 (accessed May 22, 2023).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023

Gratia Nindyaratri -- 18220017