Implementasi dan Analisis Kinerja ECC pada Mikrokontroler ESP32

Aditya Prawira Nugroho - 13520049
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13520049@std.stei.itb.ac.id

***Abstrak*—Makalah ini membahas implementasi dan analisis kinerja kriptografi kurva eliptik (ECC) pada mikrokontroler ESP32. Dengan pertimbangan batasan kemampuan komputasi dan memori yang terbatas pada mikrokontroler, ECC dipilih karena kecepatan komputasinya yang cepat dan penggunaan memori yang relatif lebih sedikit, menjadikannya cocok untuk sistem *embedded* seperti IoT. Penelitian ini mengimplementasikan ECC menggunakan bahasa C++ dan pustaka khusus untuk operasi matematika pada kurva eliptik. Hasil pengujian menunjukkan bahwa ECC mampu melakukan enkripsi dan dekripsi dengan cepat hingga ukuran angka 32 bit, membuktikan keefektifan dan keefisienannya untuk sistem dengan sumber daya terbatas. Namun, terdapat kelemahan pada implementasinya seperti tingkat keamanan yang rendah karena ukuran angka yang kecil.**

Kata kunci—microcontroller; elliptic curve; cryptography; esp32; kunci publik; kunci privat;

I. Pendahuluan

Pengadaptasian teknologi dalam kehidupan sehari-hari berkembang dengan pesat. Salah satu bentuk perkembangan dari adaptasi teknologi saat ini adalah *Internet of Things (IoT)*. Dalam penggunaannya, IoT menggunakan berbagai perangkat keras yang terdiri dari sensor dan sebuah mikrokontroler untuk mengendalikan sensor tersebut. Selain mengendalikan sensor, mikrokontroler juga mampu berkomunikasi melalui jaringan internet atau kabel. Oleh karena itu, mikrokontroler memerlukan mekanisme yang dapat menjaga keamanan dari data yang disimpan. Terlebih lagi, kemungkinan mikrokontroler menyimpan data yang sensitif sangat tinggi karena sistem IoT pada umumnya digunakan di lingkungan pribadi.

Akan tetapi, sebagai salah satu bentuk sistem *embedded*, mikrokontroler memiliki banyak sekali batasan. Beberapa batasan-batasan yang dimiliki oleh mikrokontroler, terutama ESP32, adalah kemampuan komputasi dan memori yang terbatas. Dengan begitu, pertimbangan dalam memilih algoritma dan metode kriptografi yang ringan diprioritaskan. Namun, algoritma dan metode yang dipilih tersebut tidak boleh berkompromi dengan keamanan yang diberikan.

Beragam jenis kriptografi dikembangkan untuk memenuhi berbagai kebutuhan dan meningkatkan keamanan. Salah satu jenis kriptografi yang digunakan adalah *elliptic curve cryptography* (ECC) atau kriptografi berbasis kurva elipsis. Kelebihan ECC dibandingkan algoritma kriptografi yang lain adalah komputasinya yang cepat dan penggunaan memori yang relatif lebih sedikit. Oleh karena itu, ECC dianggap lebih cocok untuk diterapkan pada sistem-sistem *embedded* dibandingkan dengan kriptografi lain seperti RSA dan ElGamal yang bergantung pada bilangan dengan ukuran sangat besar untuk menjaga keamanannya.

II. Dasar Teori

A. Kriptografi Kunci Publik

Kriptografi kunci publik adalah kriptografi yang menggunakan dua jenis kunci dalam enkripsi dan dekripsinya. Kedua kunci tersebut disebut kunci privat dan kunci publik. Kunci privat dan kunci publik berhubungan melalui operasi matematis yang dilakukan. Kriptografi ini diciptakan sebagai perbaikan dari kriptografi klasik yang hanya menggunakan satu kunci. Salah satu ide dasar kriptografi ini adalah keperluan kedua pihak yang berkomunikasi untuk menyetujui sebuah kunci yang digunakan untuk mengenkripsi pesan [1]. Tujuan utama dari kriptografi kunci publik adalah dapat melakukan enkripsi dan bertukar kunci publik melalui siaran komunikasi baik aman maupun tidak aman.

Kunci publik adalah kunci yang digunakan untuk melakukan enkripsi sebuah pesan. Sesuai namanya, kunci publik bertujuan untuk diberikan ke publik sehingga sifatnya tidak rahasia. Sedangkan kunci privat adalah kunci yang bersifat rahasia dan hanya diketahui oleh pemiliknya. Pesan yang dienkripsi oleh kunci publik hanya dapat didekripsi oleh kunci privat yang berhubungan. Oleh karena itu, pada umumnya pembangkitan kunci privat dan kunci publik dilakukan bersama.

Kriptografi kunci publik pada umumnya bergantung pada permasalahan matematis yang sulit untuk dipecahkan. Hal tersebut bertujuan untuk menyulitkan penarikan kunci privat dari kunci publik dan mencegah terjadinya *brute force attack*.

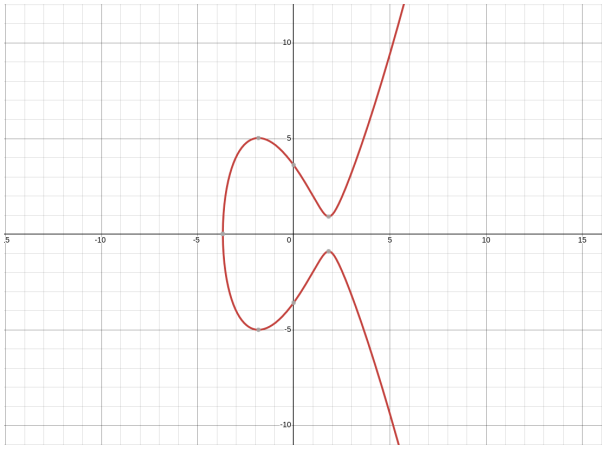
B. Elliptic Curve Cyptography (ECC)

ECC merupakan sistem kriptografi yang memanfaatkan kurva eliptik pada sebuah *Galois Field* (GF). *Galois Field* adalah sebuah medan yang terdiri dari sekumpulan angka terdefinisi yang terbatas. Contoh dari medan Galois adalah kumpulan angka pada medan 2^{16} yang berarti semua angka

pada rentang [0, 65536]. Dengan begitu, definisi formal menurut Koblitz[2] dari sebuah kurva elipsis E_k pada sebuah medan K dimana $(x,\ y) \in K^2$ adalah

$$y^2 = x^3 + ax + b,\ (a,\ b) \in K$$

Seluruh operasi ECC dilakukan dalam bentuk titik yang terletak pada kurva E_k. Dalam sistem kriptografi ini, sebuah titik primitif G digunakan untuk melakukan perhitungan dan membentuk subgrup dari kurva E_k. Titik primitif G dan parameter kurva E_k ditentukan dan dibangkitkan dengan teknik yang memastikan bahwa kurva E_k memiliki sifat prima yang kuat.



Gambar 1. Kurva eliptik $y^2 = x^3 - 10x + 13$

Operasi-operasi yang dapat dilakukan oleh sebuah titik pada kurva E_k adalah penambahan, penggandaan, perkalian, dan pengurangan. Definisi dari penambahan titik pada kurva eliptik adalah

1. Diberikan sebuah titik P dan Q pada kurva. Tarik sebuah garis dari kedua titik.
2. Apabila garis memotong kurva pada titik -R, lakukan pencerminan titik -R terhadap sumbu-x sehingga menghasilkan titik R.
3. Titik R adalah hasil penjumlahan P+Q.

Penggandaan dari titik adalah penambahan titik P terhadap dirinya sendiri. Perkalian pada titik memiliki definisi yang sama dengan perkalian pada umumnya, contoh, 4P = P + P + P + P.

Tingkat kesulitan dari ECC didasarkan pada permasalahan *Elliptic Curve Discrete Logarithm Problem* (ECDLP). Sebagai contoh, untuk menghitung hasil dari titik P dikali dengan bilangan k menjadi titik Q mudah [3]. Namun, menghitung nilai k dengan hanya mengetahui titik P dan Q sulit secara komputasi, terlebih dengan nilai yang besar. Dengan begitu, dalam ECC, dapat dikatakan bahwa Q adalah kunci publik dan bilangan k adalah kunci privat dan P adalah titik sembarang pada kurva.

Untuk melakukan enkripsi dan dekripsi, pesan harus dikodekan ke dalam titik pada kurva eliptik terlebih dahulu. Terdapat banyak cara untuk mengkodekan pesan kedalam titik. Salah satu caranya adalah mengubah karakter ke dalam bentuk representasi ASCII. Setelah itu, representasi angka ASCII digabungkan dan diubah menjadi koordinat x dari sebuah titik. Dengan begitu, hanya perlu mencari koordinat y berdasarkan x. Kebalikannya, untuk mengubah titik kembali menjadi pesan, koordinat x diekstrak dari titik dan dikodekan kembali menjadi karakter.

C. *Mikrokontroler*

Mikrokontroler adalah sebuah perangkat keras yang terdiri dari prosesor dengan memori dan komponen lain seperti I/O yang terintegrasi dalam satu chip [4]. Pada umumnya, sebuah mikrokontroler terdiri dari komponen-komponen sebagai berikut.

1. *Processor Core*, CPU dari mikrokontroler yang memuat unit logika aritmetika, register, dan unit kontrol.
2. *Memory*, memori pada mikrokontroler dipisah menjadi dua, yaitu memori program yang sering disebut juga sebagai *flash memory* dan memori data.
3. *Interrupt Controller*, digunakan untuk melakukan interupsi program utama ketika ada *event* yang masuk.
4. *Digital I/O*, digunakan sebagai *interface* yang menghubungkan komponen eksternal. Jumlah dari pin ini bervariasi tergantung tipe mikrokontroler.
5. *Analog I/O,* digunakan sebagai *interface* yang menghubungkan komponen eksternal dengan sinyal analog.
6. *Interfaces*, berfungsi sebagai sarana komunikasi dengan komputer untuk mengunduh dan mengunggah program. Bentuk paling umum adalah serial interface.

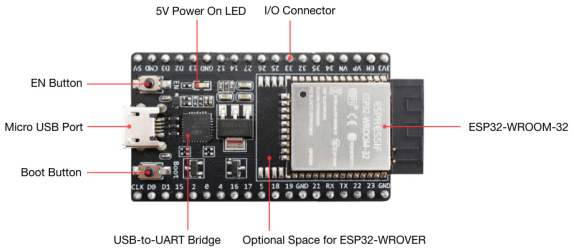
Pengembangan perangkat lunak untuk mikrokontroler berbeda dengan metode dan pengembangan perangkat lunak pada umumnya. Terdapat banyak batasan yang harus dipenuhi agar program dapat berjalan pada mikrokontroler. Salah satu tantangan utamanya adalah limitasi memori program. Pada umumnya, sebuah mikrokontroler hanya memiliki memori program sebesar 520KB. Memori tersebut tidak dapat menampung banyak program dengan pustaka lainnya. Oleh karena itu, kode program harus dibuat seefisien mungkin secara ruang. Tidak hanya efisien secara ruang, kode juga harus efisien secara eksekusi karena prosesor yang dimiliki mikrokontroler tidak sekuat komputer. Selain itu, bahasa pemrograman yang lazim digunakan adalah bahasa *low-level* yang di-*compile* seperti C atau C++.

D. *ESP32*

Salah satu contoh mikrokontroler yang umum digunakan adalah ESP32. ESP32 memiliki banyak jenis mikrokontroler dengan tipe prosesor yang bervariasi. Salah satu jenis mikrokontroler ESP32 adalah ESP32-WROOM32-DevKitC. Jenis mikrokontroler tersebut memiliki tipe prosesor yang bernama WROOM32. Prosesor WROOM32 memiliki *clock speed* sebesar 240 MHz, ROM sebesar 448 KB, dan *flash memory* sebesar 4 MB. Hal tersebut berarti kode program yang diunggah tidak bisa melebihi 4 MB.

Mikrokontroler ini memiliki berbagai macam fitur dan konektivitas. Salah satu konektivitas yang dapat digunakan

adalah WiFi bertipe 802.11b/g/n. Konektivitas tersebut dapat digunakan untuk mengirim data bentuk apapun dan berkomunikasi dengan mikrokontroler. Selain itu, konektivitas nirkabel lainnya yang dapat dimanfaatkan adalah Bluetooth. Meskipun begitu, mikrokontroler ini tetap menyediakan konektivitas kabel seperti UART, I2C, dan SPI. Bentuk dari mikrokontroler dan *layout* pin dapat dilihat pada Gambar 2.



Gambar 2. Mikrokontroler ESP32-WROOM32-DevKitC

Dari gambar tersebut, mikrokontroler memiliki beberapa komponen seperti lampu LED yang dapat digunakan untuk melihat status mikrokontroler. Tombol EN yang digunakan untuk mereset program mikrokontroler sehingga program berjalan dari *entry point*. Tombol Boot yang digunakan untuk melakukan reset *flash memory* dari mikrokontroler. Selain itu, terdapat I/O Connector yang digunakan untuk menghubungkan mikrokontroler dengan komponen eksternal seperti sensor dan layar.

III. Implementasi ECC pada ESP32

Implementasi ECC pada ESP32 membutuhkan pustaka lain untuk melakukan operasi pada kurva eliptik. Pada umumnya, terdapat pustaka yang digunakan untuk menghitung angka berukuran besar seperti 128 bit. Akan tetapi, untuk menghemat memori yang ada di ESP32, implementasi ECC akan menggunakan tipe data long long pada C++ sehingga ukuran angka maksimal adalah 64 bit. Selain itu, untuk mendukung perhitungan yang efisien, algoritma-algoritma perhitungan diimplementasi menjadi pustaka tersendiri. Dengan begitu, pustaka yang dibuat untuk mengimplementasi ECC dalam bahasa C++ adalah *math, point,* dan *ecc*.

A. *Pustaka Point*

Terdapat beberapa operasi yang diimplementasi untuk mendukung ECC, yaitu *add*, *multiply, double,* dan *subtract*. Selain itu, diperlukan beberapa metode untuk mengecek apakah titik ada di kurva. Pustaka ini dibuat dalam bentuk kelas *Point* yang memiliki atribut x dan y. Implementasi operasi *add* adalah sebagai berikut.

```
Point add(const Point& other, LL p){
        if (this->x == other.x && this->y ==
other.y) return this->double_point(2, p);

        LL dy = other.y - this->y;
        LL dx = other.x - this->x;

        if (dx < 0) {
          dx *= -1;
          dy *= -1;
        }

        dx = gcdExtended(dx, p)[0];
        LL m = mod((dy * dx), p);
        LL x3 = mod((m * m - this->x - other.x) ,
p);
        LL y3 = mod((m * (this->x - x3) -
this->y), p);

        return Point(x3, y3);
}
```

Cek apakah kedua titik tersebut sama. Jika iya, lakukan operasi *double* pada titik tersebut. Jika berbeda, hitung perbedaan nilai x dan y dari kedua titik untuk menarik garis lurus antara kedua titik. Setelah itu, kebalikan dari modulo dicari menggunakan algoritma *extended Euclidean algorithm*. Kemudian, titik hasil perpotongan garis dengan kurva dihitung dan dikembalikan oleh fungsi.

Operasi selanjutnya adalah *double*. Operasi ini memanfaatkan koefisien dari x pada persamaan kurva untuk menghitung titik pada kurva. Dengan begitu, titik hasil pencerminan dapat ditemukan. Berikut adalah kode implementasi operasi tersebut.

```
Point double_point(LL a, LL p) {
        LL m = mod(((3 * this->x * this->x + a) *
gcdExtended(2 * this->y, p)[0]), p);
        LL x3 = mod((m * m - 2 * this->x), p);
        LL y3 = mod((m * (this->x - x3) -
this->y), p);

        return Point(x3, y3);
}
```

Setelah penjumlahan dan *double*, operasi lain adalah perkalian. Perkalian skalar titik pada kurva eliptik dapat dilakukan dengan cara menambahkan titik sebanyak pengali. Namun, hal tersebut tidak efisien apabila angka bernilai besar. Oleh karena itu, implementasi perkalian memanfaatkan sifat bilangan biner dengan *least significant bit*. Pengali n akan dikonversi ke dalam bentuk biner. Setelah itu, iterasi dilakukan dari LSB. Simpan sebuah variabel sementara yang menghitung hasil perkalian. Apabila nilai iterasi pada bilangan biner saat ini bernilai 1, lakukan penambahan pada hasil dengan variabel sementara. Kemudian, lakukan operasi *double* pada variabel sementara. Detail implementasi dalam C++ adalah sebagai berikut.

```
Point multiply(const LL& scalar, LL a, LL p){
        Point temp = Point(this->x, this->y);
        Point result = Point(0, 0);

        // convert scalar to binary
        std::string binary =
std::bitset<64>(scalar).to_string();
        for (int i = binary.length() - 1; i >= 0;
i--) {
        if (binary[i] == '1') {
                if (result.isInfinite()) {
                  result = temp;
                } else {
                  result = result.add(temp, p);
```

```
            }
        }
          temp = temp.double_point(a, p);
        }

        return result;
}
```

Operasi selanjutnya yang dapat dilakukan adalah *subtract*. Untuk melakukan *subtract*, cukup lakukan penambahan titik terhadap dirinya sendiri yang sudah dicerminkan pada sumbu x. Detail implementasinya adalah sebagai berikut.

```
Point substract(const Point& other, LL p) {
        Point temp = Point(other.x, -1 *
other.y);
        Point temp1 = Point(this->x, this->y);
        Point result = temp1.add(temp, p);
        return result;
}
```

B. Pustaka Math

Pustaka ini dibuat untuk mendukung operasi perhitungan yang dilakukan agar efisien secara memori dan waktu. Pustaka ini berisi fungsi untuk melakukan *extended euclidean algorithm*, tonelli shanks, powMod, mod, dan pembangkit nilai acak untuk tipe data long long.

Fungsi *extended euclidean algorithm* digunakan untuk menghitung invers modulo. Fungsi ini akan mengambilkan sebuah larik sepanjang dua yang berisi balikan dari invers modulo dan faktor persekutuan terbesar. Selain itu, fungsi ini memanfaatkan sifat angka yang relatif prima. Dalam perhitungannya, *extended euclidean algorithm* hampir sama dengan algoritma mencari faktor persekutuan terbesar, tetapi secara terbalik. Detail implementasi kode adalah sebagai berikut.

```
LL* gcdExtended(LL a, LL b) {
        LL* result = new LL[2];
        if (a == 0) {
          result[0] = b;
          result[1] = 0;
          return result;
        }

        LL x = 0, y = 1, lastx = 1, lasty = 0;
        LL temp = 0, temp2 = a, temp1 = b, q =
0;
        while (temp1 != 0) {
          q = temp2 / temp1;
          temp = temp2;
          temp2 = temp1;
          temp1 = temp - q * temp1;
          temp = x;
          x = lastx - q * x;
          lastx = temp;
          temp = y;
          y = lasty - q * y;
          lasty = temp;
        }
        result[0] = lastx;
        result[1] = lasty;
        return result;
}
```

Untuk melakukan pengkodean pesan ke dalam titik pada kurva eliptik, diperlukan perhitungan mencari akar dari bilangan r yang memenuhi persamaan

$$r^2 \equiv n \, (mod \, p)$$

Algoritma ini digunakan untuk mencari titik y berdasarkan x yang diketahui pada medan tertentu. Fungsi ini mengembalikan larik berisi dua angka, dengan angka pertama adalah hasil akar pada medan p. Detail implementasi dari algoritma Tonelli Shanks adalah sebagai berikut.

```
LL* tonelliShanks(LL a, LL m) {
        LL* result = new LL[2];
        if (powMod(a, (m - 1) / 2, m) != 1) {
          return result;
        }
        LL q = m - 1;
        LL s = 0;
        while ((q & 1) == 0) {
          q /= 2;
          s+=1;
        }
        if (s == 1) {
          LL r = powMod(a, (m + 1) / 4, m);
          if (powMod(r, 2, m) == a) {
               result[0] = r;
               result[1] = m - r;
               return result;
          }
        }
        LL z = 1;
        while (powMod(z, (m - 1) / 2, m) != m
- 1) {
          z+=1;
        }
        LL c = powMod(z, q, m);
        LL r = powMod(a, (q + 1) / 2, m);
        LL t = powMod(a, q, m);
        LL m1 = s;
        while (t != 1) {
          LL temp = t;
          LL i = 0;
          while (temp != 1) {
               temp = powMod(temp, 2, m);
               i+=1;
          }
          LL b = powMod(c, powMod(2, m1 - i -
1, m - 1), m);
          LL b2 = powMod(b, 2, m);
          r = mod((r * b), m);
          t = mod((t * b2) , m);
          c = b2;
          m1 = i;
        }
        result[0] = r;
        result[1] = m - r;
        return result;
}
```

Dalam perhitungan, operasi pangkat banyak digunakan. Namun, perlu diperhatikan bahwa pada ECC, operasi perhitungan dilakukan pada medan tertentu. Dengan begitu, fungsi powMod diimplementasikan untuk memenuhi syarat tersebut. perpangkatan dilakukan dengan melakukan *bit shift* ke kanan. Kemudian, hasil dari perpangkatan dimodulo dengan p. Berikut adalah detail implementasi dari fungsi powMod.

```
LL powMod(LL base, LL exp, LL m) {
       LL result = 1;
       base = mod(base, m);
       while (exp > 0) {
         if (mod(exp, 2) == 1) {
              result = mod((result * base),
m);
         }
         exp = exp >> 1;
         base = mod((base * base), m);
       }
       return result;
}
```

Selain fungsi-fungsi tersebut, diperlukan sebuah fungsi yang dapat membangkitkan angka bertipe long long. Dalam implementasinya, uniform random distribution digunakan untuk membangkitkan integer. Setelah itu, dibangkitkan integer kedua dan dilakukan operasi OR sehingga kedua integer tersebut bergabung menjadi tipe data long long. Detail implementasinya adalah sebagai berikut.

```
LL llrand()
{
       std::random_device dev;
       std::mt19937 rng(dev());

std::uniform_int_distribution<std::mt19937::
result_type> randDis(0, RAND_MAX);
       if (sizeof(int) < sizeof(LL)){

         return
(static_cast<LL>(randDis(dev)) <<
(sizeof(int) * 8)) |
              randDis(dev);
       }

       return randDis(dev);
}
```

C. *Pustaka ECC*

Pustaka ini mengimplementasikan fungsi enkripsi, dekripsi, *encode, decode*, dan pembangkitan kunci privat dan publik. Dalam implementasinya, ECC adalah sebuah kelas dengan atribut

- a, koefisien dari x
- b, konstanta pada persamaan kurva eliptik
- p, medan angka pada ecc
- G, titik yang menjadi *generator* subgrup pada kurva eliptik
- n, orde dari kurva eliptik

Dalam membangkitkan pasangan kunci, fungsi akan membangkitkan kunci privat dengan nilai kurang dari n. Setelah itu, kunci publik akan dibangkitkan dengan mengalikan titik G terhadap kunci privat. Detail implementasinya adalah sebagai berikut.

```
void generateKeyPair(LL* privateKeyRes,
Point* publicKeyRes) {
       *privateKeyRes = mod(llrand(),
(this->n - 1)) ;

       *publicKeyRes =
this->G.multiply(*privateKeyRes, this->a,
this->p);
}
```

Untuk melakukan *encode*, ditentukan sebuah bilangan k untuk memastikan pesan dapat diubah menjadi titik. Kemudian, *decode* dapat dilakukan dengan cara mengekstrak koordinat x pada titik hasil *encode* dan dibagi dengan k. Detail implementasi kedua fungsi tersebut adalah sebagai berikut.

```
Point ECC::encodeMessage(LL message) {
       int k = K;
       LL ctr = message + 1;
       if (ctr * k < this->p) {
         int j = 0;
         while (j < k) {
                  LL x = message * k + j;
                  LL y2 = mod(((x * x * x)+
(this->a * x) + this->b) , this->p);
                  LL y = tonelliShanks(y2,
this->p)[0];
                  if (y != 0) {
                    return Point(x, y);
                  }
                  j++;
         }
       } else {
         return Point(0, 0);
       }
       return Point(0, 0);
}

LL ECC::decodeMessage(Point message) {
       int k = K;
       return (message.getX()) / k;
}
```

Untuk melakukan enkripsi, metode *encrypt* menerima argumen pesan dalam bentuk titik dan kunci publik yang digunakan untuk mengenkripsi pesan. Sebuah bilangan k akan dibangkitkan secara acak untuk melakukan enkripsi. Fungsi akan mengembalikan larik yang berisi dua buah elemen dengan elemen pertama adalah hasil perkalian k dengan titik G dan elemen kedua adalah cipherteks dalam bentuk titik. Implementasi enkripsi pada C++ adalah sebagai berikut.

```
Point* ECC::encryptMessage(Point message,
Point publicKey) {
	Point* result = new Point[2];
	LL k = mod(llrand(), this->n);
	Point c1 = this->G.multiply(k,
this->a, this->p);
	Point publicKeyK =
publicKey.multiply(k, this->a, this->p);
	Point c2 = message.add(publicKeyK,
this->p);
	// C1 is the random point
	result[0] = c1;
	// C2 is the encrypted message
	result[1] = c2;
	return result;
}
```

Untuk melakukan dekripsi, fungsi menerima argumen berupa cipherteks dalam bentuk larik titik dan kunci privat untuk melakukan dekripsi. Dekripsi dilakukan dengan cara mengurangi nilai dari k dikali titik G pada cipherteks. Detail implementasinya adalah sebagai berikut.

```
Point ECC::decryptMessage(Point* ciphertext,
LL privateKey) {
	Point c1 = ciphertext[0];
	Point kGPrivate =
c1.multiply(privateKey, this->a, this->p);
	Point c2 = ciphertext[1];
	Point decrypted =
c2.substract(kGPrivate, this->p);
	return decrypted;
}
```

IV. Analisis Kinerja ECC pada ESP32

Untuk menguji kinerja ecc pada ESP32, beberapa variasi ukuran dari domain parameter ecc digunakan. Variasi tersebut adalah parameter dengan ukuran 8, 16, 20, 24, dan 32 bit. Dengan begitu, ukuran pesan yang dapat dienkripsi bervariasi. Pengujian yang dilakukan adalah waktu yang dibutuhkan ecc untuk membangkitkan kunci publik dan privat serta waktu yang dibutuhkan untuk melakukan enkripsi dan dekripsi. ECC akan melakukan *encoding* pada enkripsi dan *decoding* pada dekripsi. Kedua operasi tersebut akan diukur bersamaan.

A. Ukuran 8 Bit

Untuk ukuran 8 bit, rentang integer yang mungkin adalah [-128, 127]. Detail hasil pengujian disajikan pada tabel berikut.

TABLE I. Hasil Pengujian ECC Ukuran 8 Bit

Pesan	Waktu (ms)		
	Enkripsi dan Encode	***Dekripsi dan Decode***	***Pembangkitan Kunci***
7	2	5	1
37	2	5	
100	2	5	

B. Ukuran 16 Bit

Untuk ukuran 16 bit, rentang integer yang mungkin adalah [−32768, 32767]. Detail hasil pengujian disajikan pada tabel berikut.

TABLE II. Hasil Pengujian ECC Ukuran 16 Bit

Pesan	Waktu (ms)		
	Enkripsi dan Encode	***Dekripsi dan Decode***	***Pembangkitan Kunci***
5645	3	5	1
14576	3	5	

C. Ukuran 20 Bit

Untuk ukuran 20 bit, rentang integer yang mungkin adalah [-524288, 524287]. Detail hasil pengujian disajikan pada tabel berikut.

TABLE III. Hasil Pengujian ECC Ukuran 20 Bit

Pesan	Waktu (ms)		
	Enkripsi dan Encode	***Dekripsi dan Decode***	***Pembangkitan Kunci***
14576	4	6	2
145746	4	6	

D. Ukuran 24 Bit

Untuk ukuran 24 bit, rentang integer yang mungkin adalah [-8388608, 8388607]. Detail hasil pengujian disajikan pada tabel berikut.

TABLE IV. Hasil Pengujian ECC Ukuran 24 Bit

Pesan	Waktu (ms)		
	Enkripsi dan Encode	***Dekripsi dan Decode***	***Pembangkitan Kunci***
145746	3	6	2
509768	3	6	
1000000	4	6	

E. Ukuran 32 Bit

Untuk ukuran 32 bit, rentang integer yang mungkin adalah [-2147483648, 2147483647]. Detail hasil pengujian disajikan pada tabel berikut.

TABLE V. Hasil Pengujian ECC Ukuran 32 Bit

Pesan	Waktu (ms)		
	Enkripsi dan Encode	***Dekripsi dan Decode***	***Pembangkitan Kunci***
1000000	5	6	2
10987654	6	7	
109872654	5	7	

V. Kesimpulan dan Saran

Mikrokontroler masih dapat melakukan enkripsi dan dekripsi secara relatif cepat hingga ukuran angka 32 bit. Dari hasil pengujian kinerja ECC, ukuran pesan memengaruhi waktu yang diperlukan untuk melakukan enkripsi dan dekripsi. Meskipun begitu, hal ini membuktikan bahwa ECC dapat digunakan pada sistem yang memiliki sumber daya terbatas. Akan tetapi, perlu diperhatikan bahwa limitasi memori tetap ada sehingga pustaka yang digunakan harus seminimal mungkin.

Percobaan selanjutnya dapat mengimplementasikan pustaka Big Number sehingga ukuran angka yang digunakan dapat melebihi keterbatasan C++. Hal tersebut dilakukan agar tingkat keamanan kriptografi meningkat. Selain itu, pustaka ecc dapat ditambahkan melakukan *encode* dari sebuah string menjadi titik pada kurva eliptik.

Pranala Github

Seluruh kode yang diimplementasi dapat dilihat pada pranala berikut.

Ucapan Terima Kasih

Penulis berterima kasih kepada Allah SWT yang telah memberkahi rahmat-Nya sehingga penulis mampu menyelesaikan makalah IF4020 Kriptografi. Penulis mengucapkan terima kasih kepada Pak Rinaldi Munir yang membimbing penulis selama di kelas. Penulis mengucapkan terima kasih terhadap teman-teman penulis yang membantu dalam memahami cara kerja ECC dan ESP32.

Referensi

[1] W. Diffie, “The first ten years of public-key cryptography,” *Proc. IEEE*, vol. 76, no. 5, hlm. 560–577, Mei 1988, doi: 10.1109/5.4442.

[2] N. Koblitz, “Elliptic Curve Cryptosystems”.

[3] R. Munir, “ECC-Bagian 1.” 5 April 2024. Diakses: 7 Juni 2024. [Daring]. Tersedia pada: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2023-2024/22-ECC-Bagian1-2024.pdf

[4] G. Gridling dan B. Weiss, *Introduction to Microcontrollers*, 1 ed. Vienna University of Technology: Embedded Computing Systems Group, 2007.

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, 12 Juni 2024

Aditya Prawira Nugroho