LIGHTWEIGHT PUBLIC KEY CRYPTOGRAPHY DENGAN MENGGUNAKAN ALGORITMA HILL-RSA

SKRIPSI

SUKIYA RIZKIYANI

201401068

PROGRAM STUDI S-1 ILMU KOMPUTER

FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI

UNIVERSITAS SUMATERA UTARA

MEDAN

2024

LIGHTWEIGHT PUBLIC KEY CRYPTOGRAPHY DENGAN MENGGUNAKAN ALGORITMA HILL-RSA

SKRIPSI

Diajukan untuk melengkapi tugas dan memenuhi syarat memperoleh ijazah Sarjana Ilmu Komputer

SUKIYA RIZKIYANI
201401068

PROGRAM STUDI S-1 ILMU KOMPUTER
FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS SUMATERA UTARA
MEDAN
2024

PERSETUJUAN

Judul	: *LIGHTWEIGHT PUBLIC KEY CRYPTOGRAPHY* DENGAN MENGGUNAKAN ALGORITMA HILL-RSA
Kategori	: SKRIPSI
Nama	: SUKIYA RIZKIYANI
Nomor Induk Mahasiswa	: 201401068
Program Studi	: SARJANA (S-1) ILMU KOMPUTER
Fakultas	: ILMU KOMPUTER DAN TEKNOLOGI INFORMASI UNIVERSITAS SUMATERA UTARA

Komisi Pembimbing :

Pembimbing 2

Dian Rachmawati S.Si., M.Kom.
NIP. 198307232009122004

Pembimbing 1

Dr. Mohammad Andri Budiman S.T., M.Comp.Sc., M.E.M.
NIP. 197510082008011011

Diketahui/Disetujui Oleh
Program Studi S-1 Ilmu Komputer
Ketua

Dr. Amalia ST., M.T.
NIP. 197812212014042001

PERNYATAAN

LIGHTWEIGHT PUBLIC KEY CRYPTOGRAPHY DENGAN MENGGUNAKAN ALGORITMA HILL-RSA

SKRIPSI

Saya mengakui bahwa skripsi ini adalah hasil karya saya sendiri, kecuali beberapa kutipan dan ringkasan yang masing-masing telah disebutkan sumbernya.

Medan, 22 Oktober 2023

Sukiya Rizkiyani

201401068

PENGHARGAAN

Bismillahirrahmanirrahim, segala puji syukur dipanjatkan kepada Allah *Subhanahu Wa Ta'ala* atas segala limpahan rahmat dan hidayah-Nya sehingga penulis dapat berada di tahap penyusunan skripsi ini sebagai syarat untuk mendapatkan gelar Sarjana Komputer di Program Studi S-1 Ilmu Komputer, Universitas Sumatera Utara. Tidak lupa shalawat serta salam tetap tercurahkan kepada Rasulullah *Shalallaahu 'Alayhi Wasallam* yang telah mengeluarkan umat manusia dari kegelapan menuju zaman terang benderang saat ini.

Dengan penuh rasa hormat pada kesempatan ini penulis mengucapkan terima kasih kepada Bunda, Hj. Rufaidah Nurhadi, AMK. atas segala bentuk perjuangan, kasih sayang, dan perlindungan dengan doa-doa yang dipanjatkan untuk penulis. Dan terima kasih kepada Papa, R. Abimanyu Burhan atas dukungan dan kasih sayang yang membersamai di setiap langkah penulis. Terima kasih untuk setiap dukungan yang telah diberikan hingga penulis dapat berada di titik ini.

Penyusunan skripsi ini tidak terlepas dari bantuan, dukungan, dan bimbingan dari banyak pihak. Oleh karena itu, penulis mengucapkan banyak terima kasih kepada:

1. Bapak Prof. Dr. Muryanto Amin S.Sos., M.Si. selaku Rektor Universitas Sumatera Utara.
2. Ibu Dr. Maya Silvi Lydia B.Sc., M.Sc. selaku Dekan Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara dan Dosen Pembimbing Akademik yang telah memberi banyak dukungan dan motivasi kepada penulis.
3. Ibu Dr. Amalia, S.T., M.T. selaku Ketua Program Studi S-1 Ilmu Komputer Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara.
4. Bapak Dr. Mohammad Andri Budiman S.T., M.Comp.Sc., M.E.M. selaku Wakil Dekan I Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara dan Dosen Pembimbing I yang telah memberi banyak dukungan, motivasi, masukan, dan bimbingan spiritual yang membangun kepada penulis selama penyusunan skripsi ini.
5. Ibu Dian Rachmawati S.Si., M.Kom. selaku Dosen Pembimbing II yang telah memberi banyak bimbingan, masukan, dan bantuan yang berharga kepada penulis selama penyusunan skripsi ini.

6. Bapak Ivan Jaya S.Si., M.Kom. selaku Dosen Pembanding I yang telah memberi saran dan kritik yang membangun kepada penulis terhadap penyusunan skripsi ini.
7. Bapak Amer Sharif S.Si., M.Kom. selaku Dosen Pembanding II yang telah memberi saran dan kritik yang membangun kepada penulis terhadap penyusunan skripsi ini.
8. Seluruh bapak dan ibu dosen Fasilkom-TI USU, khususnya dosen Program Studi S-1 Ilmu Komputer yang telah mendidik dan memberi wawasan serta moral yang berharga, baik di bangku perkuliahan maupun setelah lulus.
9. Kakak serta sahabat dalam menjalani '24/7' kehidupan, Maulida Zikriyati S.Farm yang telah mendoakan, memberi tawa dan kegundahan, kehangatan jiwa raga, senantiasa menemani suka dan duka, serta sumber motivasi dan panutan bagi penulis.
10. Keluarga besar Bunda Ai dan Papa Abi, yaitu Atok, Nenek, Bude, Pakde, dan sepupu yang telah memberi kasih sayang, doa, dan dukungan yang berharga yang tidak terlupakan.
11. Seluruh pegawai dan staf Fasilkom-TI USU yang telah memberi bantuan selama masa perkuliahan.
12. Sahabat menjalani dunia per-S.Kom-an, 'Olakisat', 'Ciwi-ciwi Kom B', dan seluruh teman stambuk 2020 yang ramai, meriah, dan akan selalu terkenang, terima kasih untuk seluruh pembelajaran barunya, baik dari aspek akademis maupun bekal dalam menjalani kehidupan dunia dan akhirat.
13. Sahabat bermain sejak putih abu-abu, 'Limaorang' yang senantiasa memberi memori hangat, semangat yang membara, dan terima kasih untuk pelukan di saat suka dan duka.
14. Teman seper-bimbingan 'YOK BISA YOK' yang senantiasa memberi semangat, saling mendoakan, meriah bersama menuju hari yang lebih baik dan pastinya S.Kom tepat waktu.
15. Teman berproses '237056005' yang senantiasa memberi dukungan, semangat, dan telah berkontribusi banyak, baik dalam tenaga maupun pikiran.
16. Pengurus IMILKOM USU periode 2022/2023 khususnya Biro Administrasi dan Kesekretariatan yang telah memberi pengalaman dengan aktivitas yang beragam dan bekerja sama dengan baik dalam menjalankan satu periode kepengurusan.

17. Abang-kakak senior terkhusus stambuk 2018 dan 2019 yang telah memberi banyak masukan, arahan, motivasi, dan doa baiknya kepada penulis selama masa perkuliahan dan penulisan skripsi ini.
18. Adik-adik stambuk 2021 dan 2022 yang telah meluangkan tenaga untuk bekerja sama selama masa kepanitiaan serta memberi doa dan dukungan berharga kepada penulis.

Dan seluruh pihak yang telah memberi dukungan serta doa baik yang tidak dapat penulis sebutkan satu per-satu. Semoga Allah *Subhanahu Wa Ta'ala* senantiasa melimpahkan keberkahan serta kebaikan atas semua dukungan yang telah diberikan kepada penulis dan hasil penelitian ini dapat memberi manfaat maupun inspirasi untuk kedepannya.

Medan, 22 Oktober 2023

Penulis,

Sukiya Rizkiyani

ABSTRAK

Perangkat IoT (*Internet of Things*) merupakan perangkat elektronik dengan kondisi dan sumber daya yang terbatas dan saling terhubung yang memungkinkan data dapat dikirim secara *real-time*. Namun, peningkatan jumlah perangkat IoT akan selalu berhubungan dengan meningkatnya pelanggaran keamanan, sehingga dibutuhkannya ilmu dan seni untuk mengamankan data pada perangkat IoT, yaitu dengan *lightweight cryptography*. *Lightweight cryptography* dirancang khusus untuk memberikan keamanan yang efisien tanpa beban tambahan yang besar pada perangkat dengan sumber daya terbatas. Kriptografi Hill-RSA memanfaatkan bilangan prima dan kunci matriks *involutory*, sehingga perhitungan yang dilakukan tidak terlalu kompleks. Selain itu, ukuran matriks yang digunakan ditetapkan dengan ordo 2×2 dengan proses enkripsi dilakukan per-dua karakter dengan panjang $n = 6869$ berdasarkan tabel ASCII dari karakter 'space' hingga 'Z', yaitu 59 yang disesuaikan dengan seluruh karakter sehingga menjadi 6869. Penelitian ini juga melakukan pengujian terhadap emulator mikrokontroler ESP32 berbasis MicroPython dan berdasarkan waktu eksekusi program didapat bahwa algoritma Hill-RSA tidak cukup efisien untuk diterapkan pada IoT, karena dalam rentang prima 10000 emulator mengalami performa kinerja yang memburuk tidak sampai 50% dan selama 1 menit belum memberikan *output*.

Kata Kunci: *Kriptografi, Lightweight Cryptography, Internet of Things, Hill-RSA, Matriks Involutori.*

ABSTRACT

IoT (Internet of Things) devices are electronic devices with limited and interconnected conditions and resources that allow data to be sent in real-time. However, an increase in the number of IoT devices will always be associated with an increase in security breaches, so science and art are needed to secure data on IoT devices, namely with lightweight cryptography. Lightweight cryptography is specifically designed to provide efficient security without a large additional burden on resource-limited devices. Hill-RSA cryptography utilizes prime numbers and involutory matrix keys, so that the calculations performed are not too complex. Apart from that, the size of the matrix used is determined with the order 2×2 with the encryption process carried out per two characters with a length of $n = 6869$ based on the ASCII table from the characters 'space' to 'Z', namely 59 which is adjusted to all the characters to become 6869. This research also tested the ESP32 microcontroller emulator based on MicroPython and based on the program execution time it was found that the Hill-RSA algorithm was not efficient enough to be applied to IoT, because in the prime range of 10000 the emulator experienced performance that deteriorated by less than 50% and for 1 minute did not provide output.

Keywords: *Cryptography, Lightweight Cryptography, Internet of Things, Hill-RSA, Involutory Matrix.*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Kriptografi merupakan praktik dan teknik studi yang melibatkan pembuatan dan analisis protokol untuk mencegah pesan dibaca dan dimengerti oleh pihak ketiga. Konsep kriptografi terbagi menjadi dua, yaitu kriptografi kunci privat (*single-key*) dan kriptografi kunci publik (*two-keys*). Kriptografi kunci privat disebut sebagai kriptografi simetris yang bergantung pada penggunaan satu kunci rahasia untuk proses enkripsi dan dekripsi pesan. Konsep kriptografi kunci privat sendiri terbagi lagi menjadi tiga kategori, salah satunya *block cipher* (Basri, 2016). Dan Hill Cipher merupakan algoritma kriptografi yang merahasiakan pesan dengan membaginya menjadi beberapa blok (sekumpulan bit). Sedangkan kriptografi kunci publik disebut sebagai kriptografi asimetris yang bergantung pada penggunaan dua kunci (*private* dan *public key*) (Stallings, 2022). Konsep kriptografi kunci publik memiliki sejarah yang luas dimulai dengan skema yang diperkenalkan oleh Diffie dan Hellman di tahun 1976 (Karatas, Luy, & Gonen, 2019). Lalu, pada tahun 1978 diterbitkan sebuah algoritma kriptografi kunci publik kedua, yaitu RSA. *Hybrid cryptography* merupakan penerapan gabungan dari dua konsep kriptografi, yaitu simetris dan asimetris yang memanfaatkan kelebihan masing-masing kriptografi.

Dalam proses skema kriptografi, algoritma *hybrid* menggunakan kunci ganda, di mana kunci publik dapat didistribusikan secara terbuka kepada siapa pun yang ingin mengirim pesan, sedangkan kunci privat harus dijaga kerahasiaannya oleh pemiliknya. Kriptografi *hybrid* memiliki kelebihan, terutama dalam konteks keamanan yang lebih baik. Namun, kriptografi *hybrid* memiliki waktu komputasi yang lebih lama dan ini merupakan tantangan dalam mengembangkan kriptografi *hybrid* agar waktu komputasi relatif rendah (Basri, 2016) dan dapat diterapkan untuk keamanan pada perangkat IoT (*Internet of Things*).

Lightweight cryptography adalah sub-bidang kriptografi yang berkaitan dengan teknologi yang dikembangkan untuk mengamankan perangkat yang dibatasi sumber daya, seperti perangkat IoT (Stallings, 2022). *Internet of Things* sudah menjadi dominan era penelitian karena aplikasinya di berbagai ranah seperti transportasi dan logistik pintar, perawatan kesehatan pintar, lingkungan pintar, infrastruktur pintar (kota pintar, rumah pintar, kantor pintar, mal pintar, industri 4.0), pertanian pintar dan masih banyak lagi (Thakor, Razzaque, & Khandaker, 2021). Perkembangan IoT telah membuat berjuta perangkat elektronik saling terhubung. Hal ini merupakan suatu keuntungan karena data yang terdapat pada suatu perangkat dapat dikirim ke perangkat lainnya secara *real-time*. Data yang dikirimkan ada yang bersifat umum dan rahasia. Data yang bersifat umum dapat dikirim begitu saja. Namun, data yang bersifat rahasia perlu dijaga keamanannya dan tidak boleh sampai diketahui oleh pihak luar. Terjaganya keamanan dan kerahasiaan data yang dikirimkan merupakan faktor yang penting pada era IoT. Salah satu solusi untuk menjamin keamanan dan kerahasian data adalah dengan menggunakan kriptografi.

Perangkat IoT hanya dapat memproses data yang relatif kecil serta kapasitas komputasi yang kecil, kecepatan pemrosesan data rendah, dan pemrosesan data yang ringan sehingga perangkat IoT tidak dapat mengalokasikan memori yang cukup besar dan memproses energi yang besar hanya untuk fungsi keamanan. Sehingga *lightweight cryptography* merupakan hal yang penting dalam melakukan pengamanan pada perangkat IoT karena tidak memerlukan sumber daya yang besar untuk memberikan keamanan pada perangkat IoT (Sugondo & Budi, 2021). *Lightweight cryptography* merupakan upaya untuk mengembangkan implementasi yang efisien dari algoritma *lightweight* baru (Stallings, 2022). *Lightweight cryptography* memiliki tujuan untuk mengembangkan algoritma *lightweight* baru dan ketika diimplementasikan memiliki sumber daya yang efisien dan hemat, sehingga perangkat IoT memiliki keamanan tanpa beban tambahan yang besar.

Algoritma Hill-RSA merupakan algoritma *hybrid* modifikasi dari penerapan algoritma RSA pada Hill Cipher dengan tujuan untuk meningkatkan keamanan dan kinerja algoritma. Pada modifikasi ini, algoritma RSA berperan dalam membantu mengatasi kelemahan dan meningkatkan keamanan serta efisiensi skema kriptografi konvensional, yaitu Hill Cipher dengan menghasilkan kunci

publik dan privat yang digunakan untuk proses enkripsi dan dekripsi yang berbentuk blok. Hill Cipher merupakan algoritma dengan kemampuan proses enkripsi dan dekripsi yang relatif cepat. Sedangkan algoritma RSA memanfaatkan bilangan prima yang besar sebagai kunci sehingga tingkat keamanannya relatif tinggi (Santoso, 2021).

Panjang kunci yang besar dari hasil enkripsi dan dekripsi akan membuat komputasi menjadi lambat dan perangkat dengan sumber daya terbatas tidak cocok menggunakan komputasi kriptografi yang kompleks (Rana, Mamun, & Islam, 2021). Oleh karena itu, pada penelitian ini dilakukan penyeimbangan antara tingkat keamanan dan keterbatasan sumber daya yang dimiliki oleh perangkat IoT. Kompleksitas komputasi pada algoritma Hill-RSA dikurangi dengan menetapkan ukuran matriks dan meminimalkan ukuran kunci yang sesuai untuk digunakan perangkat IoT. Penelitian ini menguji program algoritma Hill-RSA dengan menggunakan emulator mikrokontroler ESP32 berbasis MicroPython untuk melihat kemampuan komputasi algoritma Hill-RSA pada perangkat virtual yang memiliki sumber daya terbatas, sehingga pengembangan algoritma lebih terkontrol sebelum diimplementasikan langsung pada perangkat fisik.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dibuat, peningkatan jumlah perangkat IoT akan selalu berhubungan dengan meningkatnya pelanggaran keamanan, sehingga dibutuhkannya sebuah solusi kriptografi untuk menjamin keamanan dan kerahasian data. Namun, pada perangkat IoT dibutuhkan keamanan tanpa memberi beban tambahan yang besar, salah satunya dengan mengimplementasikan konsep *lightweight cryptography* yang memanfaatkan algoritma *public key* dengan komputasi yang tidak terlalu kompleks agar dapat berjalan lancar di perangkat IoT yang memiliki kapasitas terbatas.

1.3 Batasan Masalah

Penelitian ini memiliki ruang lingkup yang dibatasi oleh beberapa hal sebagai berikut

1. Perhitungan kriptografi dengan algoritma gabungan Hill Cipher dan RSA.

2. Proses *key generation*, enkripsi, dan dekripsi menggunakan ukuran matriks 2×2.
3. Tabel *encoding* berdasarkan ASCII mulai dari "*space*" hingga karakter "Z".
4. Mikrokontroler yang digunakan adalah emulator ESP32 berbasis MicroPython.

1.4 Tujuan Penelitian

Penelitian ini bertujuan untuk mengembangkan algoritma kriptografi kunci publik, yaitu algoritma Hill-RSA agar efisien dan sesuai untuk diimplementasikan pada perangkat yang memiliki keterbatasan kondisi dan sumber daya, seperti perangkat IoT (*Internet of Things*) dan melakukan pengujian dengan memanfaatkan emulator mikrokontroler ESP32 berbasis MicroPython.

1.5 Manfaat Penelitian

Manfaat yang diharapkan dari penelitian ini adalah mengamankan data dengan menggunakan algoritma kriptografi yang ringan dan efisien dalam penggunaan sumber daya. Algoritma kriptografi yang ringan dapat membantu mengamankan data pada perangkat IoT dengan menggunakan sumber daya yang terbatas, sehingga dapat meningkatkan efisiensi dan kinerja perangkat IoT. Selain itu, dapat memberikan pengalaman pengguna yang lebih baik dan memberikan rasa aman dan privasi bagi pengguna perangkat IoT.

1.6 Metodologi Penelitian

Beberapa metode yang diterapkan dalam penelitian ini adalah sebagai berikut:

1. Studi Pustaka

   Pada tahap ini, peneliti memulai penelitian dengan mengumpulkan referensi yang diperoleh dari sumber tertulis, seperti jurnal, *e-book*, dan *proceeding*. Pencarian referensi dilakukan untuk memperoleh data dan informasi yang berkaitan dengan permasalahan yang ada, seperti *lightweight public key cryptography* dan algoritma Hill-RSA.

2. Analisis dan Perancangan Sistem

   Pada tahap ini, peneliti melakukan analisis algoritma Hill-RSA sebagai algoritma kunci publik yang sesuai untuk digunakan pada perangkat kecil dengan sumber daya terbatas, seperti IoT dan efisien dalam membantu meningkatkan keamanan dan privasi data yang ditransmisikan antara perangkat IoT dan jaringan serta melakukan analisis terhadap apa saja yang akan dibutuhkan dalam penelitian untuk segera dilakukan perancangan sistem dalam sebuah diagram alir (*flowchart*).

3. Implementasi Sistem

   Pada tahap ini, peneliti melakukan proses pembuatan program berdasarkan diagram alir (*flowchart*) yang telah dirancang dan menganalisis *lightweight public key cryptography* dalam melakukan pengamanan pada perangkat IoT, seperti *confidentiality* (privasi), *integrity*, dan autentikasi ketika mengirimkan sebuah data dengan mengimplementasikan algoritma Hill-RSA serta mengurangi kompleksitas komputasi dari algoritma, sehingga sesuai untuk perangkat IoT yang hanya dapat memproses data relatif kecil ke dalam bahasa pemrograman Python.

4. Pengujian Sistem

   Pada tahap ini, peneliti melakukan proses uji coba apakah penggunaan algoritma Hill-RSA efisien dan memiliki kompleksitas komputasi yang relatif rendah untuk diterapkan pada perangkat kecil IoT yang memiliki sumber daya terbatas. Pengujian ini dilakukan dengan mengurangi kompleksitas komputasi dan mengembangkan algoritma Hill-RSA dengan menjalankan program di dalam emulator mikrokontroler ESP32 berbasis MicroPython.

5. Dokumentasi Sistem

   Pada tahap ini, peneliti melakukan dokumentasi pada setiap tahap yang terjadi dalam penelitian dan membuat kesimpulan akhir dalam bentuk laporan penelitian atau skripsi.

1.7 Penelitian Relevan

Beberapa penelitian terdahulu yang relevan dengan penelitian yang dilakukan dalam penelitian ini, antara lain:

1. Berdasarkan buku (Stallings, 2022) dengan judul "Cryptography and Network Security Principles and Practice", pada *chapter* 14 dinyatakan bahwa *lightweight cryptography* tidak hanya berfokus pada pengembangan algoritma yang aman tetapi juga dapat meminimalkan waktu eksekusi, penggunaan memori, dan konsumsi daya. Algoritma ini sesuai dengan sistem tertanam kecil, seperti IoT. *Lightweight cryptography* mencakup upaya untuk mengembangkan implementasi yang efisien dari konvensional algoritma kriptografi serta desain algoritma *lightweight* baru. Kerja *lightweight cryptography* sangat bersangkutan dengan efisiensi dan kekompakan algoritma enkripsi simetris (kunci rahasia) dan fungsi *hash* kriptografi.
2. Berdasarkan penelitian (Rana, Mamun, & Islam, 2021) yang membahas pentingnya pengembangan algoritma *lightweight cryptography* dalam mengamankan jaringan IoT yang dibatasi sumber daya. Penelitian ini juga membahas berbagai algoritma *lightweight cryptography* yang cocok untuk perangkat IoT, seperti algoritma One Round Cipher (ORC), Generalised Triangle Based Security Algorithm (G-TBSA), Modified PRESENT, Key-Dependent and Flexible (KDF), Lightweight CA (LCC), Modified QARMA, Modified Block Cipher Technique (MBCT), dan lain-lain. Penelitian ini menyimpulkan dengan membahas arah penelitian masa depan dalam *lightweight cryptography* untuk keamanan perangkat IoT dengan berfokus pada pengurangan ukuran kunci, mengurangi ukuran blok, memperkenalkan putaran yang lebih mudah.
3. Berdasarkan penelitian (Thakor, Razzaque, & Khandaker, 2021) keamanan dianggap sebagai tantangan nomor satu dalam penerapan IoT karena sebagian besar perangkat IoT dapat diakses secara fisik di dunia nyata dan banyak diantaranya memiliki sumber daya yang terbatas, seperti energi, memori, daya pemrosesan, dan bahkan perangkat fisik) yang membuat perangkat IoT menjadi target menarik bagi *attackers*. *Lightweight cryptography* merupakan solusi untuk mengatasi tantangan dalam mengamankan komunikasi pada perangkat IoT yang dibatasi sumber daya. Algoritma *lightweight cryptography* dirancang dan dioptimalkan untuk memberikan keamanan dan secara bersamaan meminimalkan penggunaan sumber daya. Algoritma ini memberikan

kerahasiaan, integritas, serta autentikasi dan otorisasi data yang melintas melalui perangkat IoT.

4. Berdasarkan penelitian (Lee & Lee, 2020) yang membahas mengenai tantangan keamanan dalam lingkungan IoT dan mengusulkan dua jenis skema autentikasi serta kesepakatan kunci yang ringan untuk perangkat IoT dengan sumber daya terbatas. Sistem kriptografi kunci publik sebagai pemberi keamanan akan sulit diterapkan pada perangkat IoT. Oleh karena itu, skema atau protokol komunikasi yang berpartisipasi harus diringankan sesuai dengan perangkat yang dibatasi sumber daya.
5. Berdasarkan penelitian (Hasoun, Khlebus, & Tayyeh, 2021) yang membahas mengenai peningkatan keamanan dan efisiensi dari Hill Cipher klasik dalam kriptografi kunci publik. Penelitian ini mengusulkan sebuah pendekatan baru, yaitu menggabungkan Hill Cipher dan RSA dengan memanfaatkan matriks *involutory* untuk membuat kunci. Penelitian ini juga melakukan evaluasi terhadap modifikasi Hill Cipher dan terbukti bahwa versi modifikasi lebih aman dan efisien waktu dalam memproses enkripsi dan dekripsi dibandingkan dengan versi klasik dan literatur lain.

1.8 Sistematika Penulisan

Sistematika penulisan skripsi yang digunakan dalam penelitian ini adalah sebagai berikut:

BAB 1 PENDAHULUAN

Bab ini mencakup penjelasan mengenai latar belakang pemilihan judul, rumusan dan batasan masalah, tujuan, manfaat, dan metodologi penelitian, penelitian relevan, dan sistematika penulisan skripsi.

BAB 2 LANDASAN TEORI

Bab ini menjelaskan beberapa teori yang berkaitan dengan penelitian, seperti pendahuluan kriptografi, *lightweight public key cryptography*, *Internet of Things* (IoT), landasan perhitungan matematika, seperti matriks dan algoritma yang digunakan, yaitu Hill-RSA.

BAB 3 ANALISIS DAN PERANCANGAN

Bab ini menjelaskan mengenai analisis pada algoritma dan dilakukan perancangan diagram yang diperlukan, seperti diagram alir (*flowchart*).

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Bab ini berisi penjelasan mengenai implementasi algoritma dalam sebuah program yang kemudian diuji pada emulator mikrokontroler serta pembahasan hasil dari pengujian yang telah dilakukan.

BAB 5 KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan yang dapat diperoleh berdasarkan pemaparan pada setiap bab serta saran yang diberikan peneliti sebagai masukan untuk penelitian selanjutnya.

BAB 2

LANDASAN TEORI

2.1 Kriptografi

2.1.1 Definisi kriptografi

Kriptografi adalah studi tentang metode komunikasi yang aman antara dua pihak. Biasanya, ada dua pihak yang ingin saling berkirim pesan, tetapi mereka ingin menghindari kemungkinan pihak ketiga memahami pesan-pesan ini. Kriptografi sering digunakan untuk meningkatkan keamanan suatu sistem informasi agar tidak terjadi kebobolan data maupun penyadapan data di saat pertukaran informasi (Lie & Alamsyah, n.d., 2023). Dengan menggunakan metode kriptografi maka akan terbentuk keamanan yang baik pada komputer sehingga akses dari luar tidak bisa langsung memasuki komputer, akan tetapi harus melewati metode kriptografi (Tristianto & Anunwembun, 2022).

Ilmu kriptografi dilakukan dengan cara substitusi (penggantian huruf) dan transposisi (perpindahan posisi) yang menghasilkan kode-kode rahasia dengan makna yang sulit terpecahkan oleh orang lain (Romindo & Ferawaty, 2021). Teknik enkripsi adalah proses awal pengacakkan data "naskah asli" (*plaintext*) hasil dari teknik enkripsi adalah sistem substitusi atau data asli sudah menjadi "naskah acak" (*ciphertext*) menjadi data yang tidak mudah dipahami orang lain, yang dapat membuka data asli (*plaintext*) hanya orang yang memiliki kunci dekripsi (Saragi, *et al*., 2020). Teknik dekripsi yang merupakan teknik kebalikkan dari proses enkripsi atau pengembalian naskah asli, dimana proses pengembalian naskah acak harus di dekripsi dengan kunci yang sesuai dengan kunci enkripsi untuk mendapatkan pesan asli. Kriptografi tidak hanya bermanfaat untuk penyembunyian data tetapi merupakan kumpulan teknik yang menyediakan keamanan pada data tersebut (Romindo & Ferawaty, 2021).

Dengan menggunakan kriptografi dalam *lightweight cryptography*, data pada perangkat IoT dapat diamankan dengan efisien dan efektif, sehingga dapat mencegah serangan dan penyadapan oleh pihak yang tidak berwenang.

Kriptografi digunakan untuk mengamankan autentikasi, kerahasiaan, integritas data, dan akses kontrol jaringan. Namun, karena banyaknya kendala perangkat IoT, kriptografi tradisional protokol tidak lagi cocok untuk semua lingkungan IoT, seperti kota pintar. Akibatnya, peneliti telah mengusulkan berbagai algoritma dan protokol kriptografi ringan untuk mengamankan data di perangkat IoT.

2.1.2 Fungsi kriptografi

Kriptografi berfungsi untuk menyediakan layanan keamanan, sebagai berikut (Munir, 2019):

1. *Confidentiality* (kerahasiaan), merupakan layanan dalam menjaga kerahasiaan pesan agar tidak dapat dibaca oleh pihak yang tidak sah dan hanya pihak yang sah yang dapat membaca serta mengembalikan kode rahasia (*ciphertext*) ke pesan asli (*plaintext*) karena hanya mereka yang memiliki kuncinya. Hal ini melibatkan proses enkripsi dalam menyembunyikan pesan menjadi kode rahasia.
2. *Data integrity* (integritas data), merupakan layanan yang memastikan bahwa pesan asli tetap utuh, tidak berubah, dan termanipulasi selama proses pengiriman pesan. Layanan ini dapat mengimplementasikan fungsi *hash* atau tanda tangan digital (*digital signature*) untuk memverifikasi bahwa data tidak berubah selama transmisi.
3. *Authentication* (otentikasi), merupakan layanan yang membantu proses verifikasi identitas satu sama lain merupakan pihak sah dan dapat memastikan keaslian pesan berasal dari pengirim (*sender*) dan penerima (*recipient*) yang sesungguhnya. *Digital signature* dan *digital certificate* dapat digunakan untuk memastikan identitas pihak yang sah.
4. *Non-repudiation* (anti-penyangkalan), merupakan layanan yang mencegah, mengatasi, dan menjamin seluruh pihak yang terlibat dalam komunikasi tidak bisa menyangkal akan keterlibatannya pada proses mengirim dan menerima pesan. Layanan ini membutuhkan infrastruktur, yaitu PKI (*Public Key Infrastructure*) yang dapat memberikan fondasi kuat dan mendukung konsep *non-repudiation* terutama dalam hal digital. *Cookies*, *cache*, dan *log file* merupakan elemen yang memberikan bukti dan catatan informasi berupa jejak histori.

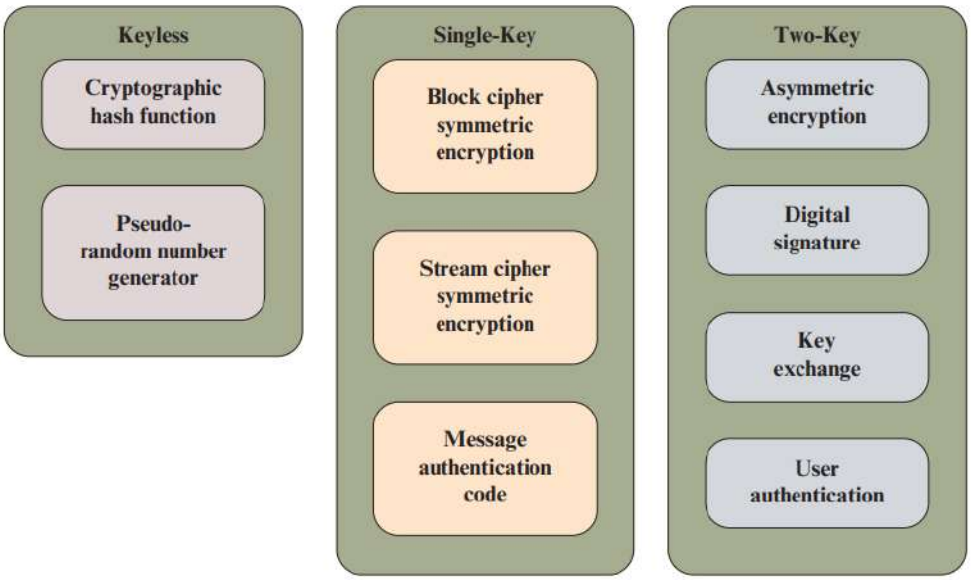
2.1.3 Komponen kriptografi

Kriptografi memiliki beberapa komponen untuk mencapai dan menciptakan sistem kriptografi yang aman dan dapat diandalkan. Berikut merupakan 7 komponen utama kriptografi (Ariyus, 2018):

1. *Plaintext*, bisa juga disebut sebagai *cleartext* dimana teks asli (pesan) ini memiliki makna yang dapat dimengerti dan dibaca oleh manusia. *Plaintext* akan diproses menjadi *ciphertext* dengan mengimplementasikan algoritma kriptografi. *Plaintext* dapat berbentuk gambar, audio, dan video.
2. *Ciphertext*, merupakan teks-kode atau pesan yang dihasilkan melalui proses enkripsi yang berbentuk karakter acak dan tidak memiliki makna, sehingga tidak dapat dibaca.
3. Enkripsi, merupakan sebuah cara dalam mengamankan kerahasiaan data yang dikirim. Pesan asli tanpa proses enkripsi disebut *plaintext* yang diubah menjadi kode rahasia (tidak dimengerti), hal ini dilakukan dengan menggunakan algoritma enkripsi dan kunci yang sesuai.
4. Dekripsi, merupakan proses kebalikan dari enkripsi di mana dekripsi adalah proses pengembalian kode rahasia (*ciphertext*) ke bentuk asalnya (*plaintext*). Algoritma yang digunakan dalam proses ini berbeda dengan algoritma yang digunakan untuk enkripsi.
5. Kunci, merupakan parameter yang diterapkan untuk mengontrol proses enkripsi dan dekripsi. Kunci terbagi menjadi dua jenis, yaitu kunci rahasia (*private key*) dan kunci umum (*public key*).
6. Pesan, merupakan data atau informasi yang dapat dikirim dan disimpan. Pesan dapat dikirim melalui pos, kurir, saluran telekomunikasi dan pesan dapat disimpan pada media perekam, seperti *storage* atau kertas.
7. Kriptanalisis, merupakan kegiatan analisis kode untuk mendapatkan pesan asli atau *plaintext* dan mencoba menembus kerahasiaan suatu kode tanpa terlebih dahulu mengetahui kunci sah yang digunakan. *Breaking code* merupakan istilah ketika pesan asli berhasil diketahui tanpa menggunakan kunci yang sah dan ini dilakukan oleh kriptanalis. Prinsip *Kerckhoff* (1883) menyatakan bahwa: Algoritma kriptografi, tabel *encoding*, dan *ciphertext* harus dianggap sudah diketahui publik; hanya kunci dan *plaintext* yang boleh dianggap belum diketahui publik (rahasia).

2.1.4 Jenis kriptografi

Sistem algoritma kriptografi berdasarkan jenis kunci yang digunakan dapat dibedakan menjadi tiga kategori, yaitu *keyless*, *single-key*, dan *two- keys*.



Gambar 2.1 Jenis Algoritma Kriptografi (Stallings, 2022)

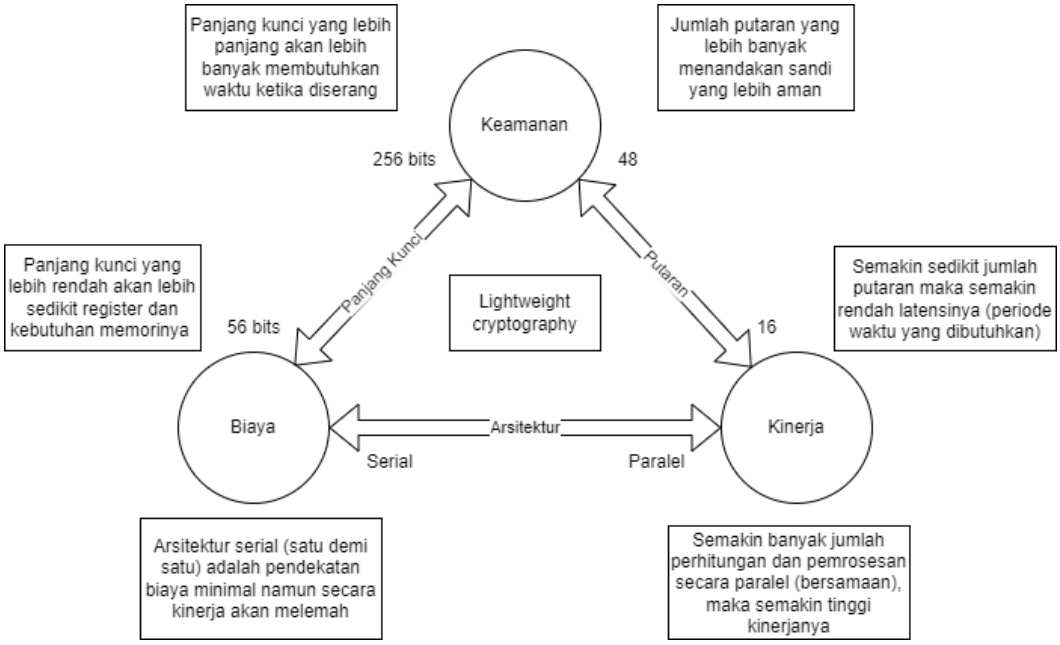
Algoritma *keyless* tidak menggunakan kunci apapun selama transformasi kriptografi. Algoritma *keyless* merupakan fungsi deterministik yang mempunyai sifat berguna tertentu untuk kriptografi, yaitu fungsi *hash* kriptografi dan pembangkit *pseudorandom number*. Algoritma kriptografi *single-key* bergantung pada penggunaan kunci rahasia (*secret key*). Algoritma enkripsi yang menggunakan satu kunci disebut algoritma simetris. Dengan enkripsi simetris, diperlukan algoritma enkripsi sebagai masukan beberapa data yang akan dilindungi dan kunci rahasia dan menghasilkan transformasi yang tidak dapat dipahami pada data tersebut. Algoritma dekripsi yang sesuai akan mengubah data dan kunci rahasia yang sama dan memulihkan data asli. Algoritma kriptografi *two-keys* bergantung pada penggunaan dua kunci, yaitu *private key* dan *public key*. Algoritma enkripsi yang menggunakan dua kunci disebut sebagai algoritma asimetris (Stallings, 2022).

Kriptografi kunci simetris aman dan relatif cepat, satu-satunya kelemahan enkripsi simetris adalah pembagian kunci antara pihak-pihak yang berkomunikasi tanpa mengorbankannya. Kriptografi asimetris menggunakan dua kunci, yaitu *private* dan *public*. Ini memastikan kerahasiaan dan integritas dengan

memanfaatkan kunci publik *recipient* dan selanjutnya memastikan autentikasi dengan menggunakan kunci pribadi *sender* untuk mengenkripsi. Dan *recipient* melakukan proses dekripsi menggunakan kunci publik terlebih dahulu, lalu menggunakan kunci pribadi yang dimiliki. Satu-satunya kelemahan enkripsi asimetris adalah kuncinya yang besar yang meningkatkan kompleksitas dan memperlambat proses (Thakor, Razzaque, & Khandaker, 2021).

2.2 *Lightweight Public Key Cryptography*

Lightweight cryptography adalah sub-bidang kriptografi yang berkaitan dengan teknologi yang dikembangkan untuk mengamankan perangkat yang dibatasi sumber daya, seperti perangkat IoT (*Internet of Things*) (Stallings, 2022). *Lightweight cryptography* fokus pada penggunaan sumber daya yang efisien untuk mengenkripsi data, sehingga dapat digunakan pada perangkat IoT yang memiliki keterbatasan sumber daya. *Lightweight cryptography* merupakan solusi untuk mengatasi tantangan dalam mengamankan komunikasi pada perangkat IoT yang dibatasi sumber daya (Thakor, Razzaque, & Khandaker, 2021). Dengan menggunakan *lightweight cryptography*, data yang dikirimkan melalui jaringan IoT dapat dienkripsi dan hanya dapat dibaca oleh pihak yang memiliki kunci dekripsi yang sesuai. Selain itu, penggunaan *lightweight cryptography* pada perangkat IoT dapat memberikan beberapa keuntungan, seperti efisiensi sumber daya, keamanan data, efisiensi energi, dan optimalisasi *bandwidth*. Tujuan utama pengembangan *lightweight* adalah mengurangi ukuran blok dan kunci, memperkenalkan putaran yang lebih mudah (Rana, Mamun, & Islam, 2021).



Gambar 2.2 *Trade-off Lightweight Cryptography* (Rana, Mamun, & Islam, 2021)

Berdasarkan Gambar 2.2 terdapat *trade-off* atau hubungan yang signifikan antara biaya, keamanan, dan kinerja pada *lightweight cryptography*. Dibandingkan dengan arsitektur serial, pengaturan arsitektur paralel akan meningkatkan kinerja dan mengurangi latensi. Akibatnya, struktur paralel dan jumlah putaran kriptografi yang lebih sedikit meningkatkan kinerja. Panjang kunci berhubungan langsung dengan keamanan jaringan *lightweight cryptography*. Semakin panjang kunci maka semakin aman. Namun, kunci yang panjang akan memerlukan lebih banyak memori dan CPU dan cenderung akan meningkatkan biaya. Biaya mengacu pada sejumlah sumber daya yang diperlukan untuk mengimplementasikan atau menjalankan algoritma kriptografi, dapat mencakup biaya perangkat keras, perangkat lunak, energi, dan biaya produksi. Algoritma kriptografi yang lebih ringan biasanya memiliki kinerja yang lebih baik karena memerlukan lebih sedikit komputasi tetapi hal ini akan mengurangi tingkat keamanan.

2.3 *Internet of Things* (IoT)

Internet of Things (IoT) adalah sebuah konsep yang mengacu pada penggunaan perangkat dan sistem cerdas yang terhubung untuk memanfaatkan data yang dikumpulkan oleh sensor dan aktuator dalam mesin dan objek fisik lainnya. IoT bekerja dengan memanfaatkan sebuah argumentasi pemrograman dengan tiap

perintah argumennya untuk menghasilkan sebuah interaksi antara sesama mesin yang terhubung secara otomatis tanpa campur tangan manusia dan dalam jarak berapapun. Algoritma kriptografi yang ringan dapat membantu mengamankan data pada perangkat IoT dengan menggunakan sumber daya yang terbatas, sehingga dapat meningkatkan efisiensi dan kinerja perangkat IoT. Keamanan adalah elemen penting dalam lingkungan IoT, jadi untuk keamanan komunikasi perlu dilakukan autentikasi antara objek komunikasi (Lee & Lee, 2020).

2.4 Mikrokontroler

Mikrokontroler adalah komputer kecil yang dirancang dengan tugas atau operasi tertentu dalam suatu *chip* IC (*Integrated Circuit*). Sebuah *chip* IC terdiri dari satu atau lebih inti prosesor, memori, dan perangkat I/O (input dan *output*). Mikrokontroler memiliki kaitan yang cukup kuat dalam pengembangan sistem cerdas yang terhubung melalui jaringan, seperti IoT. Mikrokontroler berperan sebagai otak yang mengontrol fungsi-fungsi khusus di dalam perangkat IoT, sehingga IoT dapat memberikan layanan otomatisasi yang cerdas.

2.4.1 ESP32

ESP32 merupakan mikrokontroler yang dikembangkan oleh perusahaan China di Shanghai, yaitu Espressif Systems. ESP32 sangat populer dikarenakan kemampuannya dalam mengintegrasikan konektivitas WiFi dan Bluetooth, memiliki pemrosesan yang kuat, dan berbiaya rendah. Oleh karena itu, mikrokontroler ESP32 sering digunakan dalam berbagai proyek, salah satunya proyek IoT yang membutuhkan koneksi jaringan. *Prototype* mikrokontroler ESP32 banyak tersedia di berbagai *software application* dan emulator elektronik online (*website*), seperti Wokwi. Wokwi merupakan emulator *online* yang memungkinkan pengujian terhadap sistem di berbagai macam lingkungan mikrokontroler populer, seperti ESP32 (Wahyudi, 2022) tanpa harus terlebih dahulu memiliki perangkat fisiknya.

2.4.2 MicroPython

MicroPython merupakan penerapan ulang bahasa pemrograman Python 3. Bahasa Python sendiri merupakan salah satu bahasa pemrograman yang paling banyak digunakan. MicroPython memiliki cakupan sebagian dari *library* standar Python yang khusus dirancang untuk dapat beroperasi pada mikrokontroler dengan sumber daya dan kondisi yang terbatas, seperti ESP32 serta dapat memprogram mikrokontroler dengan bahasa tingkat tinggi. Pemanfaatan MicroPython cukup sederhana dan mudah dipelajari, karena emulasinya menggunakan terminal. MicroPython sering digunakan untuk tujuan pendidikan karena memiliki REPL (*Read-Eval-Print-Loop*) yang memungkinkan untuk berinteraksi dengan mikrokontroler secara *interface* dan *real-time*, karena hal tersebut MicroPython dapat digunakan sebagai alat *prototyping* dari perangkat IoT (Gaspar, et al., 2020).

2.5 Matriks

Arthur Cayley berhasil menemukan matriks, saat Cayley berusia 17 tahun tepatnya pada tahun 1859 dalam sebuah studi sistem persamaan dan transformasi linear. Matriks adalah kumpulan bilangan yang tersusun dalam baris dan kolom dengan notasi $[\dots]$ atau $(\dots)$ (Side & Syahrana, 2015). Bilangan-bilangan yang ada di dalam suatu matriks disebut dengan elemen atau anggota matriks.

Biasanya nama suatu matriks ditulis dalam huruf kapital, seperti A dan elemen atau anggota matriks ditulis dalam huruf kecil dengan indeks ganda a_{ij}, di mana i merupakan indeks pertama yang menyatakan letak baris anggota matriks dan j merupakan indeks kedua yang menyatakan letak kolom anggota matriks. Matriks memiliki ukuran yang disebut dengan ordo dan matriks dikatakan memiliki ordo jika memiliki baris dan kolom. Bentuk umum suatu matriks A dengan ordo $n \times n$, sebagai berikut:

$$A_{n\times n} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Contoh: Matriks $A_{2\times 2} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$

2.5.1 Matriks invertible

Matriks *invertible* merupakan matriks yang memiliki *multiplicative inverse* (Dewi, Sembiring, Ginting, & Ginting, 2022). Matriks *A* adalah matriks *invertible* jika dan hanya jika determinannya tidak sama dengan nol ($|A| \neq 0$) (Strang, 2006). Determinan matriks *A* dapat ditulis dengan $\det A$ atau $|A|$,

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \text{ dengan } |A| = ad - bc \neq 0$$

2.5.2 Matriks inverse

Matriks *A inverse*, dinotasikan dengan A^{-1}. Matriks *A* yang mempunyai *inverse* disebut matriks *non-singular* dan matriks yang tidak mempunyai *inverse* disebut matriks singular. Matriks *A* merupakan matriks *inverse* (*non-singular*) jika $\det A \neq 0$. Untuk mencari *inverse* matriks dengan ordo 2×2, sebagai berikut:

$$A^{-1} = \frac{1}{|A|}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Contoh: Diketahui sebuah matriks $A = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix}$

a. Hitung determinan dan
b. Hitung *inverse*

Penyelesaian:

a. $|A| = ad - bc$
$= 2(4) - 1(3)$
$= 8 - 3 = 5$

b. $A^{-1} = \frac{1}{|A|}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$
$= \frac{1}{5}\begin{bmatrix} 4 & -1 \\ -3 & 2 \end{bmatrix}$
$= \begin{bmatrix} {}^{4}/_{5} & {}^{-1}/_{5} \\ {}^{-3}/_{5} & {}^{2}/_{5} \end{bmatrix}$

2.5.3 Matriks modulo

Matriks *modulo* adalah bilangan dari anggota matriks yang dihitung dalam *modulo* tertentu. *Modulo* dapat diartikan dalam dua makna, yaitu sisa bagi antar

dua bilangan yang dinotasikan dengan (mod) dan bermakna *interchange* yang menyatakan kesamaan nilai antar dua bilangan atau lebih dalam suatu *modulo* dengan notasi Gauss ($\equiv$). Notasi Gauss digunakan untuk reduksi atau mengubah suatu bilangan, baik dalam bentuk matriks menjadi lebih sederhana.

Jika terdapat suatu matriks *modulo* A yang ditulis dengan $A\ (mod\ n)$ dan matriks A memiliki anggota a_{ij} maka setiap anggota matriks diterapkan operasi *modulo* (Side & Syahrana, 2015).

Contoh: Lakukan konversi matriks A dan $A^{-1}\ (mod\ 2)$, di mana

$$A = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix}$$

Penyelesaian:

a. $A = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} (mod\ 2)$

$= \begin{bmatrix} 2\ mod\ 2 & 1\ mod\ 2 \\ 3\ mod\ 2 & 4\ mod\ 2 \end{bmatrix}$

$= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} (mod\ 2)$

$\therefore 2 \equiv 0\ (mod\ 2), 1 \equiv 1\ (mod\ 2), 3 \equiv 1\ (mod\ 2), 4 \equiv 0\ (mod\ 2)$

b. $A^{-1} = \frac{1}{|A|}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$

$|A| = ad - bc = 8 - 3 = 5$

$A^{-1} = 5^{-1}\begin{bmatrix} 4 & -1 \\ -3 & 2 \end{bmatrix} (mod\ 2)$ $\quad \left[\frac{1}{|A|} = |A|^{-1} = 5^{-1}\right]$

$= 5\begin{bmatrix} 4 & -1 \\ -3 & 2 \end{bmatrix}$ $\quad [5^{-1} \equiv 5$ (generalisasi teorema)]

$= \begin{bmatrix} 20 & -5 \\ -15 & 10 \end{bmatrix} (mod\ 2)$

$= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

$\therefore\ 20 \equiv 0\ (mod\ 2), -5 \equiv 1\ (mod\ 2),$

$-15 \equiv 1\ (mod\ 2), 10 \equiv 0\ (mod\ 2)$

Menghitung $5^{-1}\ (mod\ 2)$ dengan generalisasi teorema euler:

$a^{\Phi(n)} \equiv 1\ (mod\ n)$

$5^{\Phi(2)} \equiv 5^1 \equiv 1\ (mod\ 2)$ $\quad [\Phi(2) = 2 - 1 = 1]$

$5^{-1}5^2 \equiv 5^1 \equiv 5\ (mod\ 2)$

2.5.4 Matriks involutory

Matriks *involutory* merupakan matriks yang mengeliminasi matriks *inverse* (Acharya, Patra, & Panda, 2009). Suatu matriks persegi A dikatakan *involutory* jika *inverse* dari matriks tersebut A^{-1} adalah matriks asal itu sendiri:

$$A = A^{-1}$$

Matriks *involutory* harus merupakan matriks persegi dan *invertible*, di mana ketika dikali dengan dirinya sendiri akan menghasilkan matriks identitas (I) dengan ukuran matriks yang sama. Berikut merupakan bentuk matriks *involutory*:

$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, maka

$a^2 + bc = 1$ dan $d = -a$

Contoh: Buktikan bahwa matriks A merupakan matriks *involutory*

$$A = \begin{bmatrix} 3 & 2 \\ -4 & -3 \end{bmatrix}$$

Penyelesaian:

Buktikan dengan syarat matriks *involutory*:

$a^2 + bc = 1$

$3^2 + 2(-4) = 9 + (-8) = 1$ dan

$d = -a = -3$

$\therefore$ Terbukti bahwa matriks A merupakan matriks *involutory*.

2.6 *Fermat's Little Theorem*

Fermat's Little Theorem adalah teorema teori bilangan yang dinyatakan pertama kali oleh matematikawan Prancis, Pierre de Fermat kepada temannya yang merupakan seorang koresponden matematika, yaitu Frénicle de Bessy tahun 1640 dalam sebuah surat. Namun, saat itu Fermat tidak menyertakan bukti dikarenakan terlalu panjang dan Euler berhasil memberikan bukti serta mempublikasikan teorema ini pada tahun 1736. Penamaan "*little*" pada teorema ini digunakan untuk membedakannya dengan *Fermat's Last Theorem*.

Teorema ini cukup membantu dalam penerapan teori bilangan seperti relasi kongruen *modulo* n dan kriptografi kunci publik. Selain itu, teorema ini merupakan teorema yang sangat sederhana tetapi memiliki keterlibatan yang luas terhadap

kriptografi (Samandari, et al., 2023). *Fermat's Little Theorem* menyatakan bahwa: Jika p adalah bilangan prima dan a bilangan bulat positif yang tidak habis dibagi p dengan $GCD(a, p) = 1$, maka berlaku:

$a^{p-1} \equiv 1 \ (mod\ p)$.

Contoh: Buktikan $a^{p-1} \equiv 1 \ (mod\ p)$, jika $a = 3$ dan $p = 7$

Penyelesaian:

$a^{p-1} \equiv 1 \ (mod\ p)$

$3^{7-1} \equiv 1 \ (mod\ 7)$

$3^6 \equiv (3^2)^3 \ (mod\ 7)$

$\equiv 9^3 \ (mod\ 7)$

$\equiv 2^3 \ (mod\ 7)$

$\equiv 8 \ (mod\ 7)$

$\equiv 1$

∴ Dari perhitungan tersebut terbukti bahwa $a^{p-1} \equiv 1 \ (mod\ p)$.

2.7 *Euler's Totient Function*

Pada tahun 1763, Leonhard Euler memperkenalkan suatu fungsi aritmatika yang menghitung semua bilangan bulat positif tertentu (n) dan relatif prima terhadap n, yaitu *euler's totient function*. Selain dalam teori bilangan, fungsi totient euler digunakan dalam proses enkripsi pada algoritma RSA dengan tujuan keamanan dan juga banyak digunakan pada teori grup (Sonea & Cristea, 2023).

Fungsi totient euler dari n dapat ditulis dengan $\Phi(\mathrm{n})$, adalah banyaknya bilangan dalam rentang $[1, n-1]$ yang relatif prima dengan n. Totient euler menggunakan *brackets inclusive* dalam menyatakan batas suatu rentang. Jika terdapat suatu interval $1 \leq x \leq 3$ dan ditulis dengan $[1, 3]$ maka x merujuk pada inklusivitas nilai mulai dari 1 sampai 3. Terdapat dua fungsi totient euler:

1. $\Phi(\mathrm{n})$ adalah banyaknya bilangan dalam rentang $[1, n-1]$ yang relatif prima dengan n.
2. Bila n merupakan bilangan prima maka $\Phi(n) = n - 1$.

Contoh: Hitung totient euler dari 10 dan 5

Penyelesaian:

$\Phi(10) = [1, n-1]$

$= [1, 9]$

$= \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

$= 4$

$\Phi(5) = 5 - 1 = 4$

∴ Didapatkan 4 bilangan yang relatif prima dengan 10, yaitu $\{1, 3, 7, 9\}$ sehingga $\Phi(10) = 4$. Dan 5 merupakan bilangan prima sehingga $\Phi(5) = 4$.

2.8 Generalisasi Teorema Euler

Teorema euler adalah generalisasi dari *Fermat's Little Theorem*, di mana untuk $mod\ n$ dan bilangan bulat a apa pun relatif prima (*coprime*) dengan n maka (Munir, 2019):

$$a^{\Phi(n)} \equiv 1\ (mod\ n)$$

di mana $\Phi(n)$ merupakan fungsi totient euler pada n dan generalisasi teorema ini memiliki relevansi dengan *Fermat's Little Theorem*, yaitu jika $GCD(a, p) = 1$ maka $a^{p-1} \equiv 1\ (mod\ p)$ (Samandari, et al., 2023).

Contoh: Hitung $5^{-1}\ (mod\ 7)$

Penyelesaian:

$a^{\Phi(n)} \equiv 1\ (mod\ n)$ $\quad [a = 5, n = 7]$

$\Phi(7) = 6$ $\quad [\Phi(7) = 7 - 1 = 6]$

$5^{\Phi(7)} \equiv 5^6 \equiv 1\ (mod\ 7)$ $\quad [a^{\Phi(n)} \equiv 1\ (mod\ n)]$

$5^{-1}5^6 \equiv 5^5 \equiv 3\ (mod\ 7)$ $\quad [a^m \cdot a^n = a^{m+n}]$

$5^{-1} \equiv 3$

2.9 *Inverse modulo*

Inverse merupakan balikan dan *modulo* merupakan hasil sisa operasi pembagian satu bilangan dengan bilangan lainnya. $m^{-1}\ (mod\ n)$ adalah *inverse* m dalam $modulo\ n$. Syarat $m\ (mod\ n)$ memiliki *inverse* adalah:

$$GCD\ (m, n) = 1 \text{ dan } m > 1$$

Inverse dari $m\ (mod\ n)$ adalah bilangan bulat dinotasikan sebagai m^{-1}, sehingga (Munir, 2019):

$$m^{-1} \cdot m\ (mod\ n) = 1$$

Perkalian *inverse modulo* merupakan solusi menghitung *inverse* dalam *modulo* selain menggunakan generalisasi teorema Euler. Dan perhitungan *inverse modulo* banyak dimanfaatkan pada beberapa algoritma kriptografi, salah satunya algoritma RSA (Bufalo, Bufalo, & Orlando, 2021) dan Hill Cipher.

Contoh: Berapa *inverse* dari 5 $(mod\ 7)$

Penyelesaian:

$m = 5; n = 7$

Tabel 2.1 Penyelesaian *Inverse Modulo*

$\boldsymbol{m^{-1}}$	$\boldsymbol{m^{-1} \cdot 5\ (mod\ 7)}$
1	5
2	3
3	1

∴ Berdasarkan Tabel 2.1 didapatkan bahwa bilangan bulat m^{-1} yang memenuhi $m^{-1} \cdot m\ (mod\ n) = 1$ adalah 3, maka *inverse* dari 5 $(mod\ 7)$ adalah 3.

2.10 Hill Cipher (1929)

Hill Cipher merupakan *polyalphabetic substitution cipher* yang menggunakan alfabet substitusi berbeda untuk setiap huruf pada *plaintext*, *cipher* ini ditemukan oleh Lester S. Hill pada tahun 1929 (Siahaan & Siahaan, 2018). Hill Cipher menggunakan konsep substitusi polialfabet yang membuat *cipher* ini lebih sulit dipecahkan jika dibandingkan dengan *cipher* yang menggunakan substitusi monoalfabet di mana setiap huruf *plaintext* disimbolkan dengan satu alfabet yang sama.

Hill Cipher adalah algoritma kriptografi simetris yang menggunakan aritmatika *modulo* terhadap perkalian matriks dan matriks *inverse*. Matriks yang digunakan merupakan matriks *invertible* persegi dengan ukuran $n \times n$ dalam

$(mod\ p)$ (Hidayat & Alawiyah, 2013). Hill Cipher dikategorikan sebagai *block cipher* karena proses enkripsi dan dekripsi teks dibagi menjadi blok-blok dengan ukuran tertentu (Siahaan & Siahaan, 2018).

***Key Generation* (Pembangkitan Kunci)**

Kunci Hill Cipher merupakan matriks dengan ukuran $n \times n$, di mana n adalah ukuran blok. Misalkan K adalah matriks kunci dan matriks K harus merupakan matriks *invertible* yang mempunyai *inverse* (K^{-1}) karena kunci ini akan digunakan pada proses dekripsi. Proses *key generation* yang dilakukan oleh *sender* adalah sebagai berikut:

1. Tentukan ukuran matriks $K_{n\times n}$, di mana jumlah baris = kolom.
2. Bangkitkan elemen matriks $K = \begin{bmatrix} k_{11} & k_{12} & \dots & k_{1n} \\ k_{21} & k_{22} & \dots & k_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ k_{n1} & k_{n2} & \dots & k_{nn} \end{bmatrix}$.
3. Hitung determinan matriks, di mana $\det\ K \neq 0$.

Contoh:

1. $n = 2$, maka $K_{2\times 2}$
2. Bangkitkan elemen matriks, $K = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix}$
3. $\det\ K = 5 \neq 0$

***Encryption* (Enkripsi)**

Enkripsi Hill Cipher dimulai dengan mengubah *plaintext* menjadi bilangan yang sesuai dengan tabel *encoding* yang telah disepakati dan membaginya ke bentuk blok-blok. Proses enkripsi juga dilakukan oleh *sender* berikut tahapannya:

1. Tentukan tabel *encoding* untuk menyusun abjad dan angka dapat dilakukan acak atau sesuai tabel ASCII.
2. Bangkitkan suatu *plaintext* P, dengan panjang $P = 0\ (mod\ n)$.
3. Konversi P menjadi angka sesuai dengan tabel *encoding*.
4. Bagi P dalam bentuk blok yang memiliki ukuran sama dengan matriks K, misal $P = \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix}$.

5. Enkripsi *plaintext* dengan $C = K \cdot P \ (mod\ N)$.
6. Kirim C dalam bentuk angka kepada *recipient*.

Contoh:

1. Tabel *encoding*

Char	A	B	C	D	E	…	Z
Code	1	2	3	4	5	…	26

   Panjang tabel *encoding* $(N = 26)$

2. $P =$ "$\boldsymbol{SUKI}$",

   Panjang $P = 4 \ (mod\ 2) = 0$

3. Konversi *plaintext* menjadi angka

$P =$ "$SUKI$"

$P = 19, 21, 11, 9$

4. Membagi P menjadi blok matriks, yaitu:

   $P_{1,2} = \begin{bmatrix} 19 \\ 21 \end{bmatrix}$ dan $P_{3,4} = \begin{bmatrix} 11 \\ 9 \end{bmatrix}$

5. $C = K \cdot P \ (mod\ N)$

   $C_{1,2} = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 19 \\ 21 \end{bmatrix} = \begin{bmatrix} 59 \\ 141 \end{bmatrix} (mod\ 26) = \begin{bmatrix} 7 \\ 11 \end{bmatrix}$ dan

   $C_{3,4} = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 11 \\ 9 \end{bmatrix} = \begin{bmatrix} 31 \\ 69 \end{bmatrix} (mod\ 26) = \begin{bmatrix} 5 \\ 17 \end{bmatrix}$

6. Kirim C ke *recipient*

   $C = [7, 11, 5, 17]$

***Decryption* (Dekripsi)**

Tahapan proses dekripsi sama dengan proses enkripsi, tetapi proses dekripsi memerlukan perhitungan *inverse* matriks kunci (K^{-1}). Berikut merupakan proses dekripsi Hill Cipher:

1. Terima *ciphertext* atau C dari *sender*.
2. Hitung *inverse* matriks kunci K^{-1}.
3. Bagi C dalam bentuk blok yang memiliki ukuran sama dengan matriks K.
4. Dekripsi *ciphertext* dengan $P = K^{-1} \cdot C \ (mod\ N)$.
5. Gabungkan hasil dekripsi blok C dan konversi angka menjadi karakter dengan tabel *encoding* yang telah disepakati.

Contoh:

1. *Recipient* menerima *ciphertext*, yaitu $C = [7, 11, 5, 17]$
2. $K = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix}$

   $|K| = ad - bc = 8 - 3 = 5$

   $K^{-1} = \frac{1}{|K|}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$ $\quad \left[\frac{1}{|K|} = |K|^{-1} = 5^{-1}\right]$

   $= 5^{-1}\begin{bmatrix} 4 & -1 \\ -3 & 2 \end{bmatrix} (mod\ 26)$ $\quad [5^{-1} \equiv 21\ (mod\ 26)]$

   $= 21\begin{bmatrix} 4 & -1 \\ -3 & 2 \end{bmatrix} (mod\ 26)$

   $= \begin{bmatrix} 84 & -21 \\ -63 & 42 \end{bmatrix} (mod\ 26)$

   $= \begin{bmatrix} 6 & 5 \\ 15 & 16 \end{bmatrix}$

   Menghitung 5^{-1} $(mod\ 26)$ dengan perkalian *inverse modulo*:

$\boldsymbol{m^{-1}}$	$\boldsymbol{m^{-1} \cdot 5\ (mod\ 26)}$
1	5
2	10
3	15
⋮	⋮
21	1

3. Membagi C menjadi blok matriks, yaitu:

   $C_{1,2} = \begin{bmatrix} 7 \\ 11 \end{bmatrix}$ dan $C_{3,4} = \begin{bmatrix} 5 \\ 17 \end{bmatrix}$

4. $P = K^{-1} \cdot C\ (mod\ N)$

   $P_{1,2} = \begin{bmatrix} 6 & 5 \\ 15 & 16 \end{bmatrix} \cdot \begin{bmatrix} 7 \\ 11 \end{bmatrix} = \begin{bmatrix} 97 \\ 281 \end{bmatrix} (mod\ 26) = \begin{bmatrix} 19 \\ 21 \end{bmatrix}$ dan

   $P_{3,4} = \begin{bmatrix} 6 & 5 \\ 15 & 16 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 17 \end{bmatrix} = \begin{bmatrix} 115 \\ 347 \end{bmatrix} (mod\ 26) = \begin{bmatrix} 11 \\ 9 \end{bmatrix}$

5. Gabungkan hasil dekripsi dan lakukan konversi dengan tabel *encoding*:

   $P = [19, 21, 11, 9] =$ "$\boldsymbol{SUKI}$"

2.11 Algoritma RSA (1978)

Algoritma RSA (Rivest-Shamir-Adleman) merupakan algoritma terpopuler dari sekian banyak algoritma kriptografi kunci publik yang pernah ada. Algoritma ini

dibuat pada tahun 1976 oleh peneliti dari MIT (*Massachussets Institute of Technology*), yaitu Ron **R**ivest, Adi **S**hamir, dan Leonard **A**dleman yang diterbitkan pada tahun 1978 (Rachmawati & Budiman, 2020).

Keamanan algoritma RSA bergantung pada sulitnya memecahkan pemfaktoran bilangan bulat yang besar menjadi faktor-faktor prima, semakin besar suatu bilangan bulat maka keamanan kriptosistem juga semakin meningkat. Pemfaktoran bilangan ini dilakukan untuk mendapatkan kunci pribadi. Kriptografi kunci publik berkaitan dengan komunikasi melalui jalur aman (rahasia) dan tidak aman (tidak rahasia), di mana RSA memiliki (Munir, 2019):

1) p dan q (rahasia)
2) $n = p \cdot q$ (tidak rahasia)
3) $\Phi(\mathrm{n}) = (p-1)(q-1)$ (rahasia)
4) e (kunci enkripsi) (tidak rahasia)
5) d (kunci dekripsi) (rahasia)
6) m (*plaintext*) (rahasia)
7) c (*ciphertext*) (tidak rahasia)

***Key Generation* (Pembangkitan Kunci)**

Proses pembangkitan kunci dilakukan oleh *recipient*, berikut tahapannya:

1. Bangkitkan 2 buah bilangan prima, yaitu p dan q di mana $p \neq q$.
2. Hitung $n = p \times q$.
3. Hitung $\Phi(\mathrm{n}) = (p-1)(q-1)$.
4. Bangkitkan nilai e, dengan syarat:
   - e harus bilangan bulat ganjil,
   - $1 < e < \Phi(\mathrm{n})$, dan
   - $GCD\big(e, \Phi(\mathrm{n})\big) = 1$.
5. Hitung $d \equiv e^{-1}\big(mod\ \Phi(\mathrm{n})\big)$.
6. Rahasiakan nilai $(p, q, \Phi(\mathrm{n}),$ dan $d)$ sebagai kunci pribadi.
7. Publikasikan nilai e dan n sebagai kunci publik.

Contoh:

1. Bangkitkan bilangan prima $p \neq q$, yaitu $p = 3$ dan $q = 7$

2. $n = p \times q$

   $= 3 \times 7 = 21$

3. $\Phi(\text{n}) = (p-1)(q-1)$

   $= (3-1)(7-1)$

   $= (2)(6) = 12$

4. Bangkitkan nilai e, yaitu $e = 5$

5. $d \equiv e^{-1}\big(mod\ \Phi(\text{n})\big)$

   $d \equiv 5^{-1}(mod\ 12)$

   Menghitung nilai d dengan perkalian *inverse modulo*:

$\boldsymbol{m^{-1}}$	$\boldsymbol{m^{-1} \cdot 5\ (mod\ 12)}$
1	5
2	10
3	3
4	8
5	1

6. Kunci privat $p = 3,\ q = 7,\ \Phi(n) = 12$, dan $d = 5$
7. Kunci publik $e = 5$ dan $n = 21$

***Encryption* (Enkripsi)**

Proses merahasiakan pesan atau *plaintext* dilakukan oleh *sender*, berikut tahapannya:

1. Sepakati bersama tabel *encoding* dengan penerima kunci.
2. Dapatkan kunci publik milik *recipient*.
3. *Encode* pesan (m) yang akan diamankan menjadi angka dengan melihat tabel *encoding*.
4. Enkripsi pesan (m), dengan rumus:

$$c = m^e\ mod\ n$$

5. Kirim c dalam bentuk angka kepada *recipient*.

Contoh:

1. Tabel *encoding*

Char	A	B	C	D	M	P	S
Code	1	2	3	4	13	16	19

2. Kunci publik $e = 5$ dan $n = 21$
3. *Encode* pesan dengan melihat tabel *encoding*, yaitu $m = \boldsymbol{B} = 2$
4. $c = m^e \ mod \ n$

   $= 2^5 \ mod \ 21$

   $= 32 \ mod \ 21$

   $= 11$ [nilai c dikirim ke *recipient*]

***Decryption* (Dekripsi)**

Proses membalikkan *ciphertext* menjadi *plaintext* kembali dilakukan oleh *recipient*, berikut tahapannya:

1. *Recipient* menerima *ciphertext* yang dikirim oleh *sender*, yaitu c.
2. Dekripsi c menjadi m, dengan rumus:

$$m = c^d \ mod \ n$$

3. *Recipient* dapat mengerti pesan (m) dari *sender*.

Contoh:

1. $c = 11$
2. $m = c^d \ mod \ n$

   $= 11^5 \ mod \ 21$

   $= 161051 \ mod \ 21$

   $= 2$
3. Lihat tabel *encoding* $m = 2 = \boldsymbol{B}$

2.12 Algoritma Hill-RSA (2021)

Hill Cipher menggunakan kunci tetap yang diterapkan pada semua *plaintext*, hal ini mengakibatkan Hill Cipher rentan terhadap serangan. Jika kriptanalis berhasil mendapatkan dan mengetahui satu pesan asli, maka informasi *plaintext* tersebut

dapat digunakan untuk mendekripsi pesan lain, karena Hill Cipher menggunakan kunci yang sama dan tetap. Alternatif Hill Cipher adalah dengan menggunakan algoritma enkripsi yang kuat dengan kunci yang berbeda, sehingga Hill Cipher lebih aman dari serangan *plaintext*.

Algoritma Hill-RSA merupakan algoritma *hybrid* Hill Cipher yang memanfaatkan sistem enkripsi kunci publik RSA. Keamanan Hill-RSA bergantung pada kerahasiaan matriks kunci *involutory* dan pemfaktoran bilangan bulat N untuk menghitung kunci privat d. Algoritma modifikasi ini tidak memerlukan tambahan operasi dalam menentukan *inverse* dari matriks kunci, karena algoritma Hill-RSA menggunakan matriks *involutory* di mana matriks ini dapat mengeliminasi matriks *inverse* dan akan menghemat banyak waktu komputasi sehingga lebih efisien. Berikut merupakan skema utama dari algoritma Hill-RSA:

Key Generation (Pembangkitan Kunci)

Algoritma modifikasi ini memanfaatkan sebuah matriks kunci *involutory*, di mana proses *key generation* ini dimulai dengan membangkitkan kunci *involutory* terlebih dahulu oleh *sender* dengan tahapan sebagai berikut:

1. Bangkitkan matriks kunci *involutory* ($K = K^{-1}$):
   - Matriks K harus berukuran persegi dan merupakan matriks *invertible*.
   - Bangkitkan K atas kesepakatan bersama dan elemen matriks $K_{ij} \in \mathbb{Z}_n$, dengan syarat:

     $$a^2 + bc = 1 \text{ dan}$$
     $$d = -a$$

     apabila matriks $K = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$.
2. *Sender* mengirim matriks kunci *involutory* K kepada *recipient* secara rahasia.

Recipient akan melanjutkan proses pembangkitan kunci dengan algoritma RSA:

1. *Recipient* menerima matriks kunci rahasia yang dikirim oleh *sender*, yaitu K.
2. Bangkitkan 2 buah bilangan prima, yaitu p dan q di mana $p \neq q$.
3. Hitung $N = p \times q$.
4. Hitung $\Phi(\mathrm{N}) = (p-1)(q-1)$.

5. Bangkitkan nilai e, dengan syarat:
   - $1 < e < \Phi(\mathrm{N})$,
   - e harus bilangan bulat ganjil, dan
   - $\mathrm{GCD}\big(\mathrm{e}, \Phi(\mathrm{N})\big) = 1$.
6. Hitung $d \equiv e^{-1}\big(mod\ \Phi(\mathrm{N})\big)$.
7. Rahasiakan nilai d sebagai kunci privat.
8. Publikasikan nilai e dan N sebagai kunci publik.

Contoh:

Sender membangkitkan matriks kunci *involutory*.

1. Bangkitkan matriks persegi $K = K^{-1}$ dan $K_{ij} \in \mathbb{Z}_{11}$, dengan

$$K = \begin{bmatrix} a & b \\ c & d \end{bmatrix},\ a^2 + bc = 1 \text{ dan } d = -a$$

$K = \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix},\ 1^2 + 1(0) = 1$ dan $d = -1$

Pastikan bahwa $K = K^{-1}$

$K = \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix} (mod\ 5) \equiv \begin{bmatrix} 1 & 1 \\ 0 & 4 \end{bmatrix}$

$|K| = ad - bc = 4 - 0 = 4$

$K^{-1} = |K|^{-1} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$

$K^{-1} = 4^{-1} \begin{bmatrix} 4 & -1 \\ 0 & 1 \end{bmatrix} (mod\ 5)$ $\quad [4^{-1} \equiv 4\ (mod\ 5)]$

$= 4 \begin{bmatrix} 4 & -1 \\ 0 & 1 \end{bmatrix} (mod\ 5)$

$= \begin{bmatrix} 16 & -4 \\ 0 & 4 \end{bmatrix} (mod\ 5)$

$= \begin{bmatrix} 1 & 1 \\ 0 & 4 \end{bmatrix}$

Dari perhitungan tersebut didapat bahwa matriks $K = K^{-1}$, yaitu $\begin{bmatrix} 1 & 1 \\ 0 & 4 \end{bmatrix} (mod\ 5)$

2. Kirim matriks kunci *involutory* $K = \begin{bmatrix} 1 & 1 \\ 0 & 4 \end{bmatrix} (mod\ 5)$ ke *recipient* secara rahasia

Recipient melanjutkan proses pembangkitan kunci dengan RSA.

1. *Recipient* menerima K yang dikirim oleh *sender*, yaitu $\begin{bmatrix}1 & 1\\ 0 & 4\end{bmatrix} (mod\ 5)$
2. Bangkitkan bilangan prima $p \neq q$, yaitu $p = 3$ dan $q = 7$
3. $N = p \times q$
   $= 3 \times 7 = 21$
4. $\Phi(\mathrm{N}) = (p-1)(q-1)$
   $= (3-1)(7-1)$
   $= (2)(6) = 12$
5. Bangkitkan nilai e, yaitu $e = 5$
6. $d \equiv e^{-1}\big(mod\ \Phi(\mathrm{N})\big)$
   $d \equiv 5^{-1}(mod\ 12)$
   Menghitung nilai d dengan perkalian *inverse modulo*:

$\boldsymbol{m^{-1}}$	$\boldsymbol{m^{-1} \cdot 5\ (mod\ 12)}$
1	5
2	10
3	3
4	8
5	1

7. Kunci privat $d = 5$
8. Kunci publik $e = 5$ dan $N = 21$

***Encryption* (Enkripsi)**

Algoritma modifikasi ini memanfaatkan dua algoritma dalam proses enkripsi, teknik pertama menggunakan algoritma enkripsi Hill Cipher terhadap blok *plaintext* m dan teknik kedua menggunakan algoritma enkripsi RSA terhadap hasil enkripsi dari algoritma pertama, yaitu Hill Cipher. Langkah-langkah proses enkripsi Hill-RSA adalah sebagai berikut:

1. Nyatakan suatu *plaintext* dan lakukan konversi sesuai dengan tabel *encoding* yang telah disepakati.

2. Bagi konversi *plaintext* dalam blok matriks M, di mana $m_i \in \mathbb{Z}_n$, misal $M = \begin{bmatrix} m_1 \\ \vdots \\ m_n \end{bmatrix}$.
3. Hitung blok dengan enkripsi Hill Cipher berikut:

$$C_{H_i} = K \cdot M \ (mod\ n)$$

4. Lanjutkan dengan menghitung hasil C_H dengan enkripsi RSA berikut:

$$C_{R_i} = (C_{H_i})^e \ (mod\ N)$$

5. Kirim hasil enkripsi $C_{R_i} = \begin{bmatrix} c_{r1} \\ \vdots \\ c_{rn} \end{bmatrix}$ kepada *recipient*.

Contoh:

1. Tabel *encoding*

Char	A	I	K	S	U
Code	1	9	11	19	21

Plaintext $=$ "$\boldsymbol{SUKI}$" $= [19, 21, 11, 9]$

2. Membagi *plaintext* menjadi blok matriks M dan $m_i \in \mathbb{Z}_5$:
   - $\boldsymbol{S} = 19\ (mod\ 5) \equiv 4$
   - $\boldsymbol{U} = 21\ (mod\ 5) \equiv 1$
   - $\boldsymbol{K} = 11\ (mod\ 5) \equiv 1$
   - $\boldsymbol{I} = 9\ (mod\ 5) \equiv 4$

   , maka $M_{1,2} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$ dan $M_{3,4} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$
3. Enkripsi dengan Hill Cipher (C_H):

$$C_{H_i} = K \cdot M \ (mod\ n)$$

$$C_{H_{1,2}} = \begin{bmatrix} 1 & 1 \\ 0 & 4 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix} (mod\ 5) = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$$

$$C_{H_{3,4}} = \begin{bmatrix} 1 & 1 \\ 0 & 4 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 16 \end{bmatrix} (mod\ 5) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

4. Lanjut dengan enkripsi RSA (C_R):

$$C_{R_i} = (C_{H_i})^e \ (mod\ N)$$

$$C_{R_{1,2}} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}^5 = \begin{bmatrix} 3125 \\ 1024 \end{bmatrix} (mod\ 21) = \begin{bmatrix} 17 \\ 16 \end{bmatrix}$$

$$C_{R_{3,4}} = \begin{bmatrix} 5 \\ 16 \end{bmatrix}^5 = \begin{bmatrix} 3125 \\ 1048576 \end{bmatrix} (mod\ 21) = \begin{bmatrix} 17 \\ 4 \end{bmatrix}$$

5. *Sender* mengirim *ciphertext* ke *recipient*, yaitu $C_{R_{1,2}} = \begin{bmatrix}17\\16\end{bmatrix}$ dan $C_{R_{3,4}} = \begin{bmatrix}17\\4\end{bmatrix}$

***Decryption* (Dekripsi)**

Setiap blok *ciphertext* yang dikirim oleh sender harus didekripsi. Algoritma Hill-RSA membutuhkan kunci rahasia d dan matriks kunci rahasia K, tahapan prosesnya sebagai berikut:

1. *Recipient* menerima *ciphertext* yang dikirim oleh *sender*, yaitu C_{R_i}.
2. Hitung blok C_{R_i} dengan dekripsi RSA berikut:
$$D(C_{R_i}) = (C_{R_i})^d \ (mod\ N)$$
3. Lanjutkan dengan dekripsi $D(C_{R_i})$ menjadi M dengan dekripsi Hill Cipher berikut:
$$M = K \cdot D(C_{R_i}) \ (mod\ n)$$
4. Gabungkan hasil dekripsi blok C_{R_i} dan konversi dengan tabel *encoding*.

Contoh:

1. Terima *ciphertext* $C_{R_{1,2}} = \begin{bmatrix}17\\16\end{bmatrix}$ dan $C_{R_{3,4}} = \begin{bmatrix}17\\4\end{bmatrix}$
2. Dekripsi dengan RSA $D(C_{R_i})$:
$$D(C_{R_i}) = (C_{R_i})^d \ (mod\ N)$$
$$D\left(C_{R_{1,2}}\right) = \begin{bmatrix}17\\16\end{bmatrix}^5 = \begin{bmatrix}1419857\\1048576\end{bmatrix} (mod\ 21) = \begin{bmatrix}5\\4\end{bmatrix}$$
$$D\left(C_{R_{3,4}}\right) = \begin{bmatrix}17\\4\end{bmatrix}^5 = \begin{bmatrix}1419857\\1024\end{bmatrix} (mod\ 21) = \begin{bmatrix}5\\16\end{bmatrix}$$
3. Lanjut dengan dekripsi Hill Cipher yang memanfaatkan $K = K^{-1}$:
$$M = K \cdot D(C_{R_i}) \ (mod\ n)$$
$$M_{1,2} = \begin{bmatrix}1 & 1\\0 & 4\end{bmatrix} \cdot \begin{bmatrix}5\\4\end{bmatrix} = \begin{bmatrix}9\\16\end{bmatrix} (mod\ 5) = \begin{bmatrix}4\\1\end{bmatrix}$$
$$M_{3,4} = \begin{bmatrix}1 & 1\\0 & 4\end{bmatrix} \cdot \begin{bmatrix}5\\16\end{bmatrix} = \begin{bmatrix}21\\64\end{bmatrix} (mod\ 5) = \begin{bmatrix}1\\4\end{bmatrix}$$
4. Gabungkan blok hasil dekripsi dan konversi dengan tabel *encoding*:
$M = [4, 1, 1, 4]$
$M =$ "$\boldsymbol{SUKI}$"

BAB 3

ANALISIS DAN PERANCANGAN

3.1 Analisis

Analisis adalah proses memecahkan dan menguraikan suatu informasi menjadi komponen yang lebih kecil sehingga mudah dipahami (Mulyanto, Herfandi, & Kirana, 2022). Proses analisis merupakan landasan awal sebelum melakukan perancangan dan pengembangan suatu sistem, agar sesuai dengan kebutuhan dan lebih terstruktur dalam mencapai tujuan akhir. Analisis sistem terdiri dari analisis masalah, analisis kebutuhan, dan analisis proses.

3.1.1 Analisis masalah

Analisis masalah merupakan tahapan mengidentifikasi sebab dan akibat dari suatu permasalahan. Penelitian ini mengidentifikasi permasalahan pada perangkat IoT, terutama dalam hal keamanan. Peningkatan jumlah perangkat IoT akan selalu berbanding lurus dengan meningkatnya pelanggaran keamanan pada IoT itu sendiri. IoT merupakan perangkat dengan sumber daya terbatas dan hanya dapat memproses data yang relatif kecil.

Pada tahap mengidentifikasi masalah, penelitian ini menggunakan metode 5-*Whys* untuk mempermudah proses analisis. Metode 5-*Whys* merupakan metode tanya-jawab sederhana yang cukup efektif ketika fokus utamanya adalah mengidentifikasi sebab dan akibat dari suatu masalah. Metode ini dilakukan dengan bertanya "mengapa" secara berulang sebanyak 5 kali atau lebih, diantaranya sebagai berikut:

1. Mengapa ada kebutuhan sistem kriptografi pada perangkat IoT?

   IoT (*Internet of Things*) merupakan perangkat elektronik yang saling terhubung dengan kemampuan mengirim data melalui jaringan. IoT dapat mengirim data privasi maupun tidak privasi secara *real-time* serta menjaga keamanan dan kerahasiaan data privasi dari pihak yang tidak terkait. Solusi dari keamanan pengiriman data privasi adalah dengan menggunakan kriptografi, tetapi penggunaan kriptografi biasa tidak dapat diterapkan karena

IoT memiliki sumber daya terbatas yang tidak dapat mengalokasikan memori dan energi yang besar hanya untuk keamanan saja. Oleh karena itu, dibutuhkannya perancangan algoritma kriptografi yang ringan.

2. Mengapa perangkat IoT memerlukan *lightweight cryptography*?
   Perangkat IoT merupakan perangkat dengan sumber daya terbatas yang hanya dapat memproses data serta memiliki kapasitas komputasi yang relatif kecil, sehingga perangkat IoT tidak dapat mengalokasikan memori yang besar hanya untuk fungsi keamanan. Oleh karena itu, metode kriptografi yang digunakan untuk keamanan IoT harus ringan dan efisien dalam menggunakan sumber daya. Dan jawaban dari permasalah tersebut adalah *lightweight cryptography* yang dirancang khusus untuk menghadapi keterbatasan sumber daya IoT dan memberikan keamanan dengan tingkat yang cukup serta dengan minimal *overhead*. *Lightweight cryptography* merupakan hal yang sangat penting dalam melakukan pengamanan yang efisien pada perangkat IoT karena tidak memerlukan sumber daya besar.

3. Mengapa efisiensi komputasional menjadi pertimbangan utama dalam memilih algoritma *lightweight public key cryptography*?
   Algoritma *lightweight public key cryptography* yang dipilih harus efisien secara komputasi, di mana proses kriptografi, seperti *key generation*, enkripsi, dan dekripsi dilakukan dengan memastikan keamanan perangkat IoT yang tercapai tanpa memiliki beban tambahan besar dan tanpa mengorbankan kinerja perangkat IoT serta daya tahan baterainya.

4. Mengapa penting untuk mempertimbangkan keamanan informasi bahkan dalam lingkungan dengan sumber daya terbatas?
   Keamanan suatu informasi tidak bergantung pada tingkat sumber daya yang disediakan. Jika terdapat serangan informasi pada suatu perangkat maka akan menyebabkan risiko yang cukup krusial, terutama dalam hal keselamatan dan keamanan individu yang dirugikan. Walaupun sumber daya suatu perangkat terbatas, keamanan informasi dari perangkat tersebut tetap menjadi prioritas utama. Bahkan terkadang dikarenakan sumber daya terbatas tersebut,

informasi yang dikumpul dan ditransmisikan oleh perangkat lebih bersifat sensitif.

5. Mengapa Hill-RSA dipilih khusus di antara berbagai algoritma kriptografi untuk implementasi *lightweight public key cryptography*?

   Pemilihan kombinasi Hill Cipher dan RSA dipilih dengan tujuan untuk memberikan keamanan yang cukup dengan minimal *overhead* dalam mengimplementasikan *lightweight public key cryptography*. Hill Cipher merupakan algoritma simetris dengan proses enkripsi pesan yang cepat dan RSA merupakan algoritma asimetris (*public key*) yang populer digunakan karena memiliki keamanan yang cukup tinggi dengan memanfaatkan bilangan prima dan distribusi kunci secara publik. Selain itu, terbukti pada poin 2.12 bahwa modifikasi Hill Cipher dan RSA memiliki kalkulasi waktu enkripsi dan keamanan yang lebih baik.

3.1.2 Analisis kebutuhan

Analisis kebutuhan merupakan tahapan penting yang berfokus pada pengenalan dan pemahaman kebutuhan yang diperlukan untuk merancang sistem dalam memenuhi tujuan. Analisis kebutuhan dibagi dalam dua bagian utama, yaitu fungsional dan non-fungsional.

1. Kebutuhan fungsional

   Kebutuhan fungsional adalah spesifikasi fungsionalitas yang dapat dilakukan dan harus ada pada suatu sistem dalam mencapai tujuan utama. Penelitian ini memiliki kebutuhan fungsional utama, yaitu:

   a. Membangkitkan matriks kunci *involutory*, kunci privat, dan kunci publik yang akan digunakan pada proses enkripsi dan dekripsi dengan algoritma Hill-RSA.
   b. Batasan matriks yang digunakan adalah matriks persegi dan matriks kunci *involutory* dengan ordo 2×2 agar kompleksitas komputasi tidak memberi beban kinerja yang besar pada perangkat IoT.
   c. Menerima masukan *plaintext* dan mengubahnya dalam bentuk angka sesuai dengan panjang tabel *encoding* yang telah disepakati, yaitu tabel ASCII dari karakter ‘space’ hingga karakter ‘Z’.

d. *Plaintext* yang telah diubah menjadi angka dibagi dalam beberapa blok mengikuti ukuran matriks kunci.
e. Melakukan proses enkripsi, yaitu merahasiakan *plaintext* matriks menjadi *ciphertext* dengan dua tahap enkripsi (Hill Cipher dan RSA) yang memanfaatkan kunci yang telah dibangkitkan.
f. Proses enkripsi *plaintext* dilakukan per-dua karakter dengan panjang tabel *encoding* dari karakter 'space' hingga 'Z', yaitu 59 berdasarkan tabel ASCII. Proses ini dapat meningkatkan keamanan *plaintext* dan mendukung pemanfaatan sumber daya yang lebih efisien.
g. Pemanfaatan panjang tabel *encoding* ditingkatkan dari 59 menjadi 6869, dikarenakan proses enkripsi dilakukan per-dua karakter. Berikut merupakan pemaparan detail dari $((90 - 32 + 10 \times 100) + (90 - 32 + 10)) + 1$:
   - $(90 - 32)$: pengurangan seluruh karakter 'space' hingga 'Z' dengan 32 dilakukan untuk membuat kode karakter dimulai dari 0 hingga 58.
   - $(58 + 10)$: penambahan 10 dilakukan untuk menyeimbangkan seluruh karakter sehingga satuannya menjadi puluhan.
   - (68×100): hasil penambahan dikali dengan 100 yang akan menghasilkan 6800 karena proses enkripsi dilakukan per-dua karakter yang setiap karakternya memiliki satuan angka puluhan.
   - $(90 - 32)$: pengurangan dengan 32 dilakukan kembali untuk mengindentifikasi karakter kedua yang akan digabung.
   - $(58 + 10)$: penambahan 10 dilakukan kembali untuk menyeimbangkan seluruh karakter sehingga satuannya menjadi puluhan.
   - $(6800) + 68 = 6868 + 1$: penambahan 1 dilakukan untuk menghindari karakter 'ZZ' dengan hasil *modulo* 0 yang merupakan karakter 'space' setelah dikurang dengan 32.
   - Sehingga panjang tabel *encoding* menjadi 6869.
h. Menghasilkan *ciphertext* matriks angka yang tidak terbaca dan bermakna dari proses enkripsi yang telah dilakukan.
i. Melakukan proses dekripsi, yaitu mengubah *ciphertext* matriks menjadi *plaintext* kembali dengan dua tahap dekripsi (RSA dan Hill Cipher) yang memanfaatkan kunci yang telah dibangkitkan.

j. Mengubah matriks angka hasil dari proses dekripsi menjadi *plaintext* agar dapat dibaca dan dimengerti dengan tabel *encoding* yang telah disepakati sebelumnya.
k. Menghasilkan *plaintext* yang terbaca dan bermakna yang merupakan hasil dari konversi matriks angka.
l. Memindahkan program algoritma yang telah dirancang ke atas emulator mikrokontroler ESP32 berbasis MicroPython.
m. Melihat waktu yang dibutuhkan mikrokontroler ESP32 dalam mengeksekusi program algoritma Hill-RSA dalam satuan waktu *millisecond* (*ms*).

2. Kebutuhan non-fungsional

   Kebutuhan non-fungsional adalah spesifikasi tambahan yang mendukung sistem, dapat berupa kinerja, keamanan, serta batasan dari sistem. Penelitian ini memiliki beberapa kebutuhan non-fungsional, yaitu:

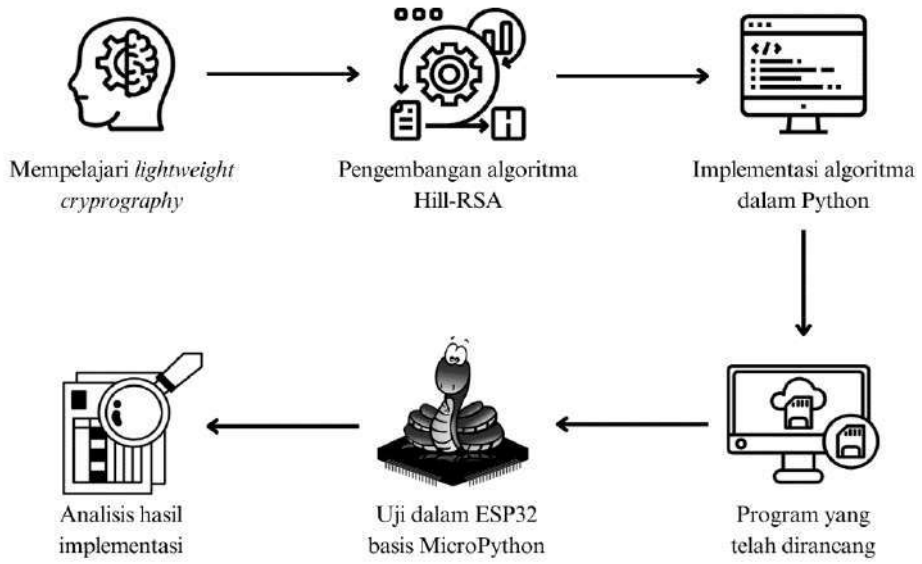
   a. Memiliki kontrol yang akan terus meminta inputan benar jika inputan yang diberikan tidak dalam batasan karakter yang telah disepakati, yaitu ‘space’ hingga karakter ‘Z’.
   b. Menampilkan hasil pembangkitan kunci, proses enkripsi *plaintext*, dan dapat mengembalikan *ciphertext* menjadi *plaintext* kembali melalui proses dekripsi.
   c. Menampilkan waktu yang dibutuhkan mikrokontroler ESP32 dalam mengeksekusi program algoritma Hill-RSA dalam satuan *millisecond* (*ms*).

3.2 Perancangan Sistem

Perancangan sistem dibangun berdasarkan analisis yang telah dilakukan terhadap penelitian. Fokus utama dari perancangan ini adalah meningkatkan efisiensi dan efektivitas sistem yang melibatkan spesifikasi detail mengenai sistem yang dirancang dengan memanfaatkan bentuk diagram, termasuk bagaimana proses alur dari perancangan sistem program penelitian ini.

3.2.1 Diagram umum penelitian

Diagram umum penelitian membantu memahami bagaimana suatu penelitian dilaksanakan dengan memberi gambaran visual mengenai langkah-langkah yang dilakukan, termasuk langkah perumusan masalah, pengumpulan dan analisis data, hingga penyusunan laporan.

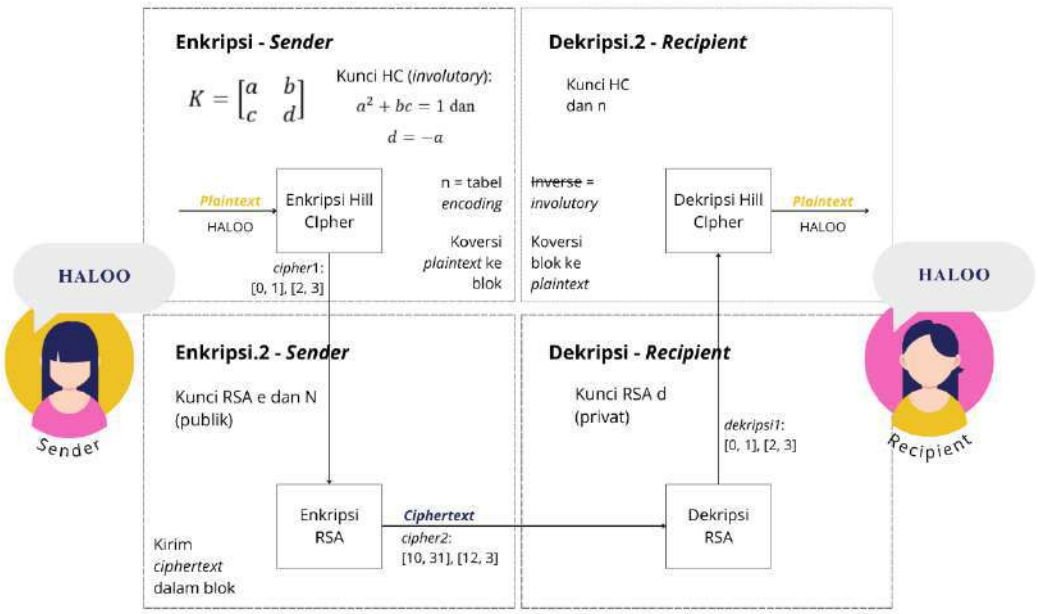


Gambar 3.1 Alur Penelitian

Berdasarkan Gambar 3.1 alur penelitian ini digambarkan dalam bentuk diagram visual, di mana penelitian ini dimulai dengan mengumpulkan informasi dari beberapa referensi dan mempelajari konsep *lightweight cryptography*, yang kemudian dilakukan pengembangan algoritma Hill-RSA dan mengimplementasikannya dalam pemrograman Python. Program Python dikonversi menjadi MicroPython agar sesuai dengan konsep *lightweight cryptography* dan dilakukan pengujian beserta analisis terhadap mikrokontroler ESP32.

3.2.2 Diagram umum Hill-RSA

Penelitian ini menggunakan algoritma kriptografi *hybrid* yang merupakan modifikasi dari penerapan algoritma RSA terhadap Hill Cipher dengan tujuan untuk meningkatkan keamanan dan kinerja algoritma Hill Cipher. Berikut merupakan representasi skema kriptografi *hybrid* yang lebih spesifik, yaitu Hill Cipher dan RSA.



Gambar 3.2 Alur Algoritma Hill-RSA

Pada Gambar 3.2 diperlihatkan bahwa algoritma gabungan ini memiliki skema tambahan di mana algoritma simetris, yaitu Hill Cipher yang terlebih dahulu merahasiakan *plaintext* lalu hasil enkripsi Hill Cipher dilanjutkan dengan metode enkripsi RSA yang memanfaatkan kunci publik, yaitu e dan N. Hasil akhir dari kedua metode enkripsi akan dikirim ke penerima (*recipient*) dalam bentuk blok-blok matriks.

Tahap pertama dekripsi dilakukan dengan algoritma RSA yang memanfaatkan kunci privat, yaitu d dan dalam *modulo* N, hasil dari dekripsi pertama ini akan sama dengan '*cipher1*' yang dilanjutkan ke tahap kedua dekripsi, yaitu Hill Cipher yang seharusnya memanfaatkan fungsi matriks *inverse*, tetapi pada algoritma ini dilakukan dengan membuat matriks kunci *involutory* sehingga tidak perlu operasi tambahan untuk matriks *inverse*. Setelah proses dekripsi berhasil dilakukan, *plaintext* akan dihasilkan dan dapat dibaca serta dimengerti oleh penerima pesan.

3.2.3 Perancangan program pada emulator ESP32

Perancangan program algoritma Hill-RSA pada penelitian ini pertama kali menggunakan bahasa pemrograman Python. Dan bahasa yang disediakan oleh emulator *online* ESP32, yaitu Wokwi adalah MicroPython.

MicroPython merupakan bahasa pemrograman yang dirancang khusus untuk bekerja dalam kondisi dan sumber daya yang terbatas. Oleh karena itu, MicroPython tidak memiliki perpustakaan standar yang lengkap dan hanya mencakup sebagian kecil dari perpustakaan Python. Salah satu *library* yang tidak tersedia di MicroPython adalah *library* 'time' yang digunakan untuk mengukur waktu eksekusi program. Alternatif dari *library* tersebut adalah dengan mengimpor *library* 'utime'. Berikut merupakan modul 'time':

```
import time             # modul python

start_time = time.time()       # catat waktu mulai

# input code

end_time = time.time()       # catat waktu selesai
execution_time =  (end_time - start_time) * 1000

print("Selesai.")
print("Waktu eksekusi:", execution_time, "ms")
```

Dan berikut merupakan modul 'utime':

```
import utime             # modul micropython

start_time = utime.ticks_ms()     # catat waktu mulai

# input code

end_time = utime.ticks_ms()       # catat waktu selesai
execution_time = utime.ticks_diff(end_time, start_time)

print("Selesai.")
print("Waktu eksekusi:", execution_time, "ms\n")
```

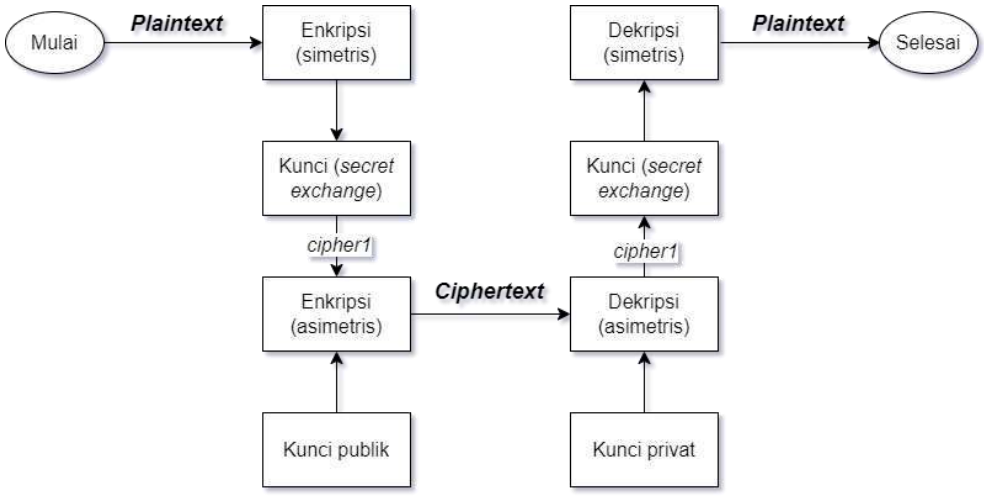
3.3 *Flowchart* (Diagram Alir)

Flowchart merupakan penggambaran visual dari serangkaian proses, keputusan, dan aliran logika dari suatu sistem yang direpresentasikan dalam bentuk diagram alir dengan beberapa simbol yang ditetapkan oleh American National Standards Institute (ANSI), seperti panah, persegi panjang, hexagon, dan lainnya. Penelitian ini memiliki beberapa *flowchart* yang berkaitan dengan algoritma Hill-RSA, sebagai berikut.

3.3.1 Flowchart hybrid cryptography

Flowchart kriptografi gabungan ditunjukkan pada Gambar 3.3, di mana proses enkripsi dimulai dengan mengenkripsi *plaintext* dengan metode simetris yang menggunakan pertukaran satu kunci bersifat rahasia. Lalu, dari proses enkripsi simetris tersebut dihasilkan kode *cipher* pertama untuk dilanjutkan proses enkripsi kedua dengan metode enkripsi asimetris menggunakan kunci publik yang telah dibangkitkan sebelumnya. Dari kedua tahap enkripsi tersebut dihasilkan *end-ciphertext* yang dikirim ke *recipient* untuk dilakukan tahap dekripsi.

Recipient telah menerima *ciphertext* yang dikirim oleh *sender* dan akan dilakukan proses dekripsi untuk dapat membaca dan mengerti pesan yang dikirim. Proses dekripsi kriptografi *hybrid* juga terbagi menjadi dua tahap, yaitu dekripsi pertama menggunakan metode dekripsi asimetris yang memanfaatkan kunci privat milik *recipient*. Dari proses dekripsi pertama dihasilkan kode *cipher* yang memiliki nilai sama dengan kode *cipher* pertama saat proses enkripsi. Lalu, hasil kode *cipher* dari dekripsi pertama dilanjutkan dengan metode dekripsi simetris yang memanfaatkan satu kunci yang telah ditukarkan secara rahasia sebelumnya. Jika perhitungan yang dilakukan benar, maka *plaintext* yang dihasilkan dapat terbaca dan memiliki makna.

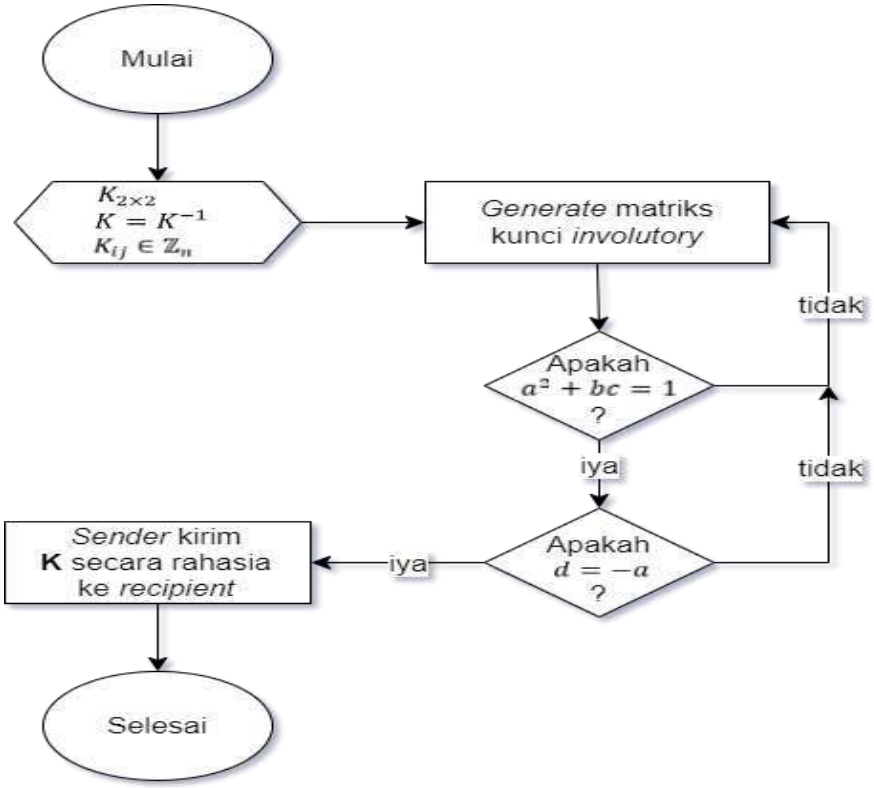


***Gambar 3.3** Flowchart Hybrid Cryptography*

3.3.2 Flowchart key generation by sender

Flowchart key generation yang dilakukan oleh *sender* (pengirim pesan) ditunjukkan pada Gambar 3.4. Proses pembangkitan kunci dimulai dengan syarat

bahwa kunci yang akan dibangkitkan merupakan matriks *involutory* ($K = K^{-1}$) yang memiliki elemen matriks berupa bilangan bulat dalam $mod\ n$ dengan ordo 2×2. Lalu, bangkitkan K atas kesepakatan bersama, di mana elemen matriks $a^2 + bc = 1$ dan $d = -a$. Dan kirimkan matriks kunci *involutory* K yang sudah dibangkitkan kepada *recipient* secara rahasia.

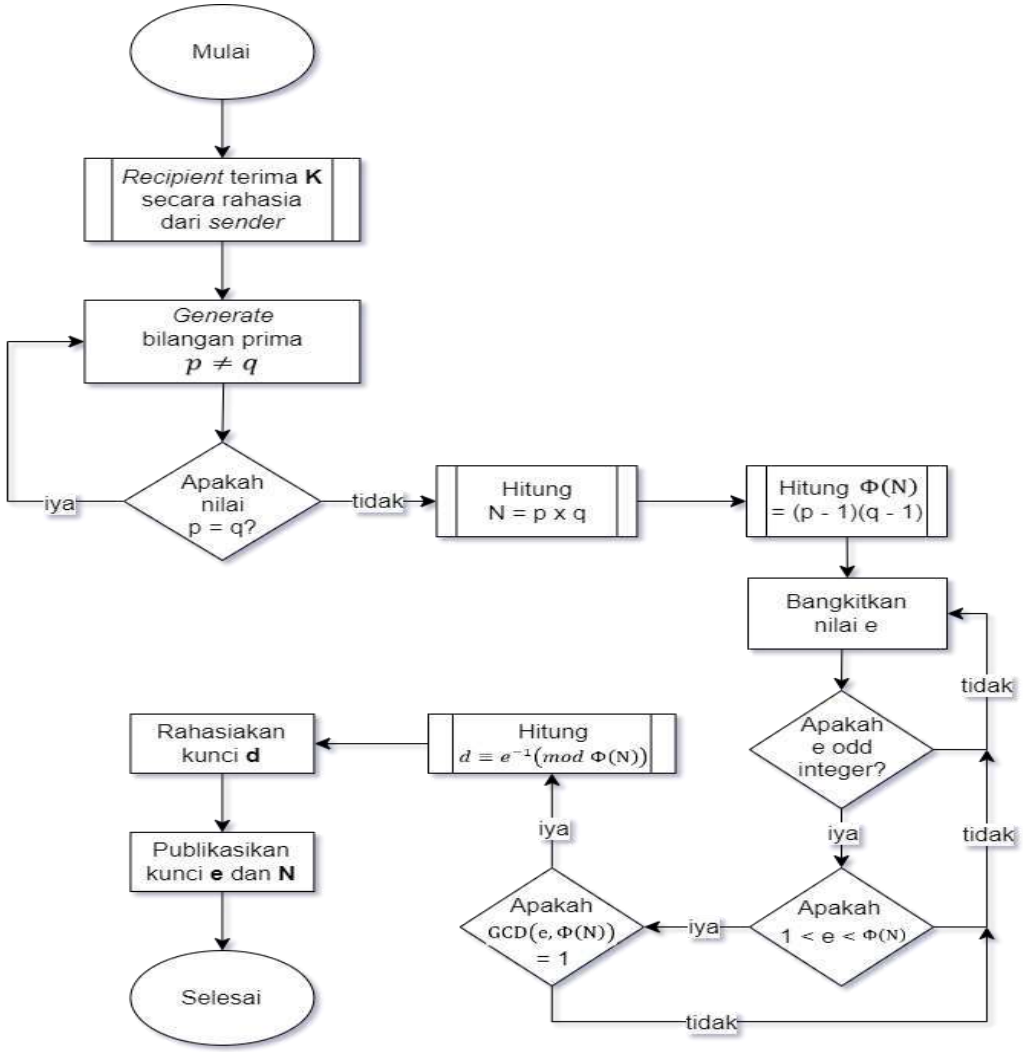


Gambar 3.4 *Flowchart Key Generation* (*Sender*)

3.3.3 Flowchart key generation by recipient

Flowchart key generation yang dilakukan oleh *recipient* (penerima pesan) ditunjukkan pada Gambar 3.5. Bangkitkan dua buah bilangan prima yang berbeda $p \neq q$, setelah *recipient* menerima matriks kunci K. Lalu, dari nilai p dan q yang telah dibangkitkan, hitung N dan $\phi(N)$. Setelah menghitung kedua nilai tersebut, bangkitkan nilai e yang memiliki syarat e harus bilangan bulat ganjil, $1 < e < \Phi(\mathrm{n})$, dan $GCD\big(e, \Phi(\mathrm{n})\big) = 1$. Setelah berhasil membangkitkan nilai e dengan syarat tersebut, hitung nilai d.

Dari perhitungan yang dilakukan maka didapatkan nilai d sebagai kunci privat dan nilai e dan nilai N sebagai kunci publik.



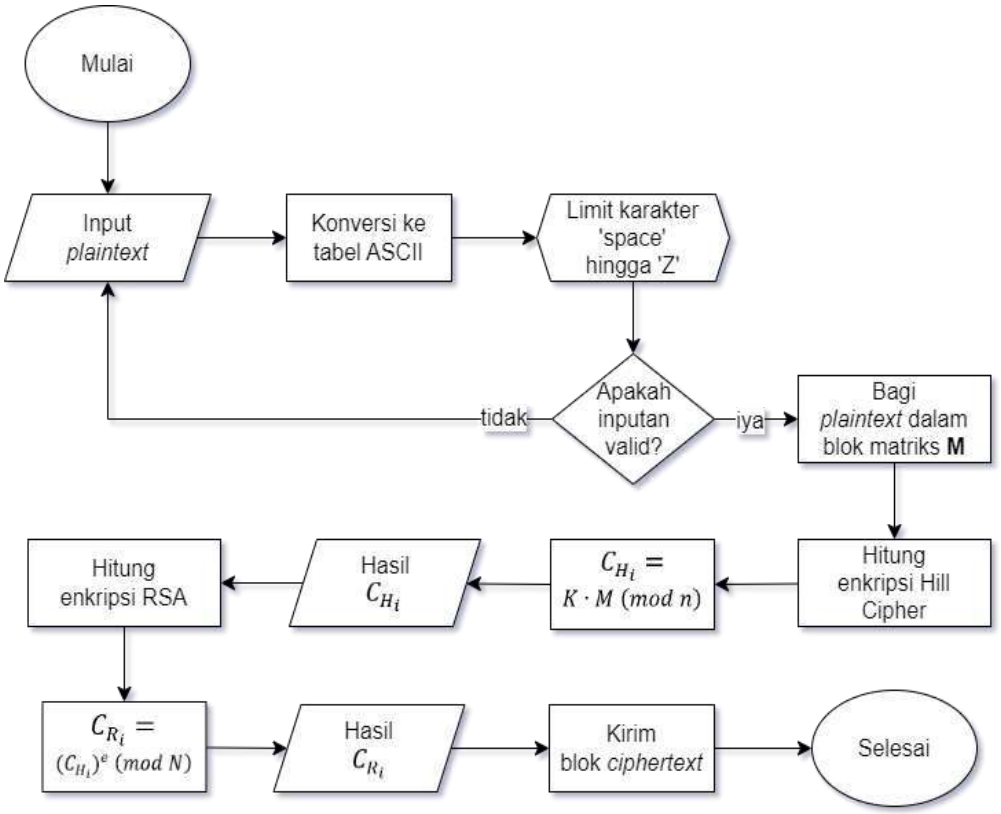
Gambar 3.5 *Flowchart Key Generation* (*Recipient*)

3.3.4 Flowchart enkripsi by sender

Flowchart enkripsi yang dilakukan oleh *sender* (pengirim pesan) ditunjukkan pada Gambar 3.6 dengan menyatakan *plaintext* yang ingin dikirim, melakukan konversi *plaintext* berdasarkan tabel *encoding* dengan rentang yang telah disepakati. Mengubah *plaintext* ke dalam beberapa blok matriks M.

Proses enkripsi dilakukan dengan dua tahap, di mana yang pertama dengan melakukan enkripsi Hill Cipher pada setiap blok matriks M. Proses enkripsi Hill Cipher memanfaatkan matriks K dan n yang sebelumnya telah dibangkitkan dan disepakati. Hasil dari enkripsi Hill Cipher ini akan dienkripsi lagi dengan RSA yang memanfaatkan *modulo* eksponensial dari kunci publik, yaitu e dan N. Hasil

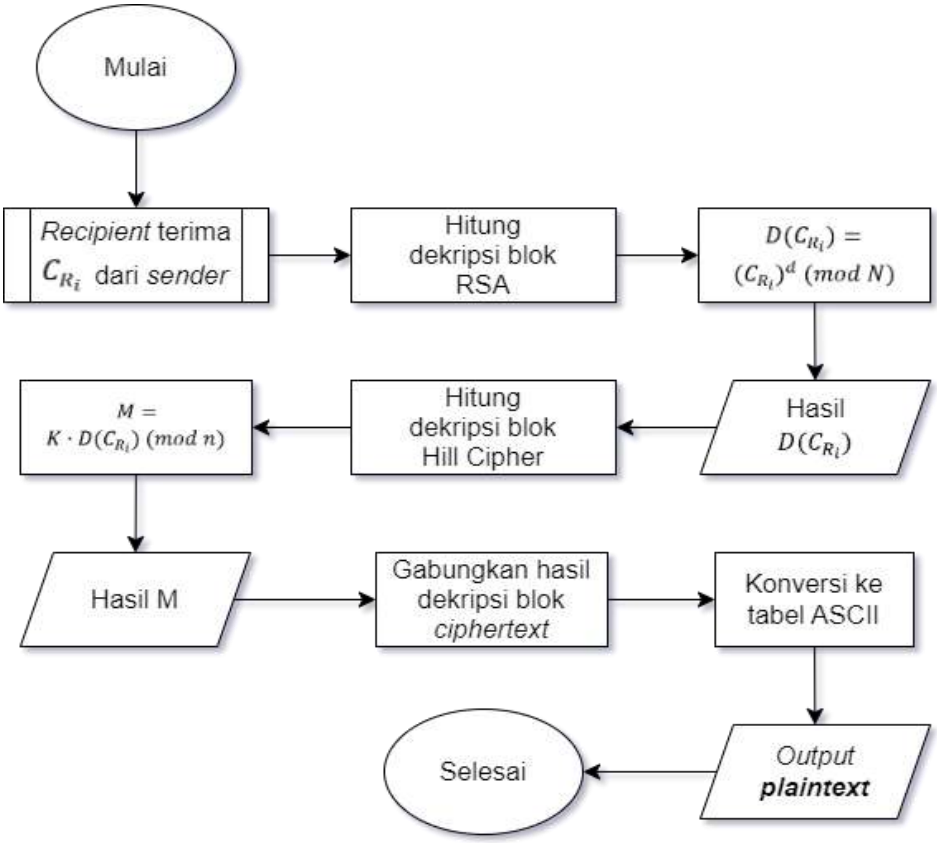
dari tahap enkripsi yang kedua, yaitu RSA merupakan *end-ciphertext* yang dikirim untuk dilakukan tahap dekripsi oleh *recipient*.



Gambar 3.6 *Flowchart* Enkripsi (*Sender*)

3.3.5 Flowchart dekripsi by recipient

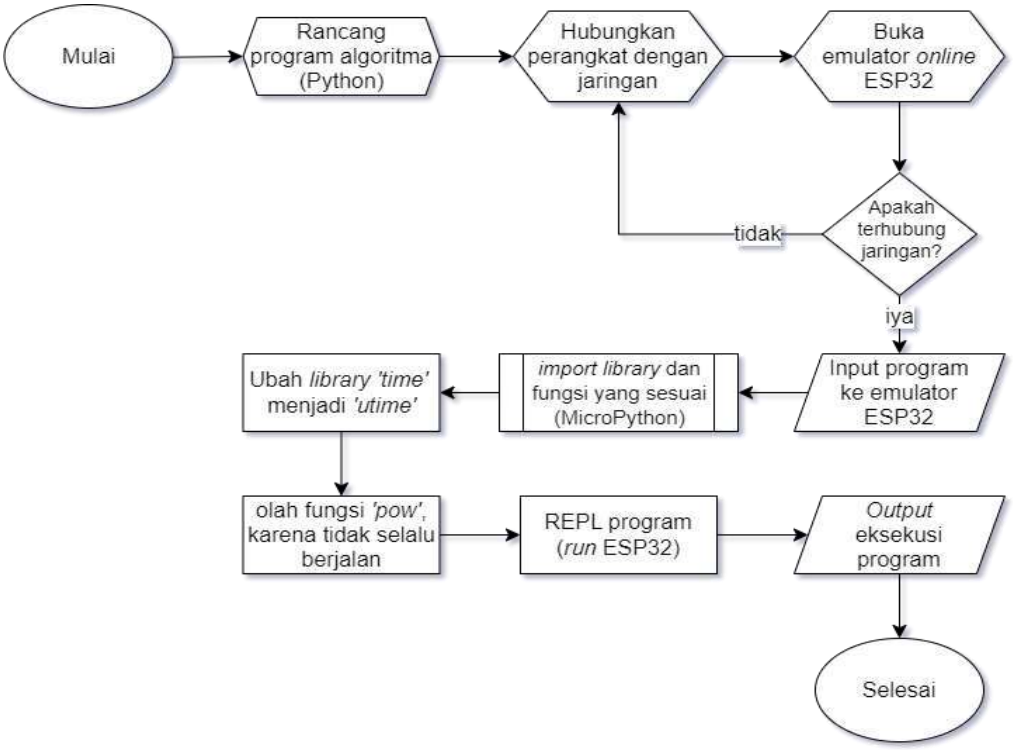
Flowchart dekripsi yang dilakukan oleh *recipient* (penerima pesan) ditunjukkan pada Gambar 3.7, di mana setelah *recipient* menerima *end-ciphertext* dari *sender*, dilakukan perhitungan dekripsi pertama dengan menggunakan RSA yang memanfaatkan kunci privat *d* untuk mendekripsi blok *ciphertext*. Kemudian hasil dekripsi tersebut dilanjutkan ke tahap dekripsi kedua dengan menggunakan Hill Cipher yang memanfaatkan matriks kunci *involutory*. Setelah melakukan dua tahap dekripsi tersebut, gabungkanlah hasil dari dekripsi terakhir yang dilakukan. Lalu, konversi sesuai dengan tabel *encoding* yang sebelumnya telah disepakati. Jika perhitungan yang dilakukan benar maka *plaintext* yang dihasilkan dapat terbaca dan memiliki makna.



Gambar 3.7 *Flowchart* Dekripsi (*Recipient*)

3.3.6 Flowchart emulator ESP32 berbasis MicroPython

Flowchart peralihan program yang telah dirancang ke dalam emulator ESP32 ditunjukkan pada Gambar 3.8. Jika emulator ESP32 sudah terkoneksi jaringan maka pindahkan program ke dalam emulator tersebut. Sesuaikan *library* yang diimpor dengan bahasa yang digunakan emulator, pada penelitian ini emulator ESP32 menggunakan MicroPython. *Library* 'time' tidak tersedia di MicroPython sehingga diganti menjadi 'utime' dan fungsi *pow()* tidak selalu berjalan, sehingga untuk beberapa *code* diubah agar emulator dapat berjalan.



Gambar 3.8 *Flowchart Emulator ESP32*

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

Setelah tahap analisis dan perancangan, tahap selanjutnya adalah implementasi dan pengujian, di mana dalam tahap ini penelitian melibatkan sebuah *software*, yaitu Visual Studio Code sebagai *code editor* dan melakukan pengujian dengan menggunakan emulator *online* berupa *prototype* mikrokontroler, yaitu ESP32. Bahasa pemrograman yang digunakan penelitian ini adalah Python dan MicroPython untuk menjalankan emulator ESP32.

4.1.1 Spesifikasi laptop dan emulator yang digunakan

Pada tahap implementasi digunakan beberapa perangkat yang mendukung proses pengerjaan penelitian ini, diantaranya adalah perangkat laptop dan perangkat emulator mikrokontroler ESP32 yang diakses secara *online* melalui situs web Wokwi.

Laptop yang digunakan untuk merancang program algoritma Hill-RSA dan melakukan pengujian kompleksitas algoritma dengan menjalankan emulator ESP32 adalah Acer dengan spesifikasi sebagai berikut:

Tabel 4.1 Spesifikasi Laptop yang Digunakan

Seri	Prosesor (CPU)	Inti Prosesor	RAM	Memori	Grafis (GPU)
Acer (Aspire A514-54G)	2.40 GHz (8 CPUs)	Dual-core	4096 MB (4 GB)	512 GB	2 GB

Berdasarkan Tabel 4.1 laptop Acer dengan seri Aspire A514-54G memiliki:

- 8 inti pemroresan yang independen dan memiliki kecepatan dari setiap inti prosesor dalam menjalankan instruksi sebesar 2.40 GHz.
- 2 inti pemrosesan yang independen yang memungkinkan prosesor menangani tugas secara bersamaan (*multitasking*).

- Penyimpanan utama dengan sifat penyimpanan sementara sebesar 4096 MB yang setara dengan 4 GB.
- Penyimpanan dengan jangka panjang yang bersifat permanen sebesar 512 GB.
- *Graphics Processing Unit* atau kartu grafis NVIDIA sebesar 2 GB yang bertujuan untuk meningkatkan kinerja grafis pada perangkat.

ESP32 menyediakan *datasheet* yang berisi dokumen resmi yang diproduksi oleh Espressif Systems untuk memberikan pemahaman mendalam mengenai spesfikasi, fitur, dan karakteristik dari mikrokontroler ESP32. Berikut merupakan spesifikasi teknis dari ESP32 (Espressif, 2023):

Tabel 4.2 Spesifikasi Mikrokontroler ESP32

Seri	**Prosesor (CPU)**	**Inti Prosesor**	**RAM**	**Memori**	**Bluetooth**	**WiFi**
ESP32	160 *up to* 240 MHz	*Dual-core*	RAM: 520 KiB	ROM: 448 KiB	Bluetooth v4.2 BR/EDR	802.11n (2.4 GHz)
				SRAM: 520 KB	Bluetooth LE	150 Mbps (transfer data)

Berdasarkan Tabel 4.2 berikut merupakan penjelasan mengenai spesifikasi dari mikrokontroler ESP32 yang memiliki:

- Kecepatan prosesor sebesar 160 hingga 240 MHz.
- *Microprocessor* Tensilica Xtensa 32-bit LX6 yang memiliki 2 inti. Semua chip dalam seri ESP32 adalah *dual-core*, kecuali ESP32-S0WD merupakan *single-core*.
- Kapasitas *Random Access Memory* sebesar 520 KiB (KibiByte) yang setara dengan 0.53248 MB untuk menyimpan data yang aktif digunakan oleh CPU atau program saat ini.
- Kapasitas *Read-Only Memory* sebesar 448 KiB (KibiByte) yang setara dengan 0.458752 MB untuk menyimpan program atau kode yang tidak

dapat diubah, seperti instruksi yang diperlukan untuk menjalankan mikrokontroler.

- Kapasitas *Static Random-Access Memory* sebesar 520 KB (KiloByte) yang setara dengan 0.52 MB untuk menyimpan data yang dapat diubah, seperti nilai variabel dan memanipulasi serta menyimpan data sementara.
- Bluetooth v4.2 BR/EDR (Bluetooth Basic Rate/Enhanced Data Rate) merupakan versi perbaikan dan penambahan fitur yang lebih baik dari sisi kinerja, konektivitas, keamanan, dan efisiensi energi.
- Bluetooth LE (Bluetooth Low Energy) disebut Bluetooth *smart* yang dirancang untuk konsumsi daya rendah tetapi tetap mendukung proses koneksi yang cepat antara perangkat.
- Standar WiFi yang dikeluarkan oleh IEEE (Institute of Electrical and Electronics Engineers) dengan kecepatan frekuensi operasi WiFi sebesar 2.4 GHz dengan kecepatan dalam mentransfer data yang dicapai oleh perangkat dengan koneksi WiFi sebesar 150 Mbps (150 megabit per detik).

4.1.2 Program algoritma Hill-RSA

Modifikasi dari suatu algoritma akan membuat perhitungan algoritma tersebut menjadi lebih rumit, terutama jika nilai dan *plaintext* yang digunakan memiliki digit atau karakter yang sangat panjang. Begitu pula, untuk modifikasi algoritma Hill Cipher yang memanfaatkan algoritma kunci publik RSA untuk meningkatkan keamanan serta efisiensi algoritma. Program algoritma Hill-RSA dirancang menggunakan bahasa Python dan menggunakan *software* Visual Studio Code.

Pada Gambar 4.1 terdapat *output key generation* dari hasil *running* program. Algoritma Hill-RSA memiliki dua tahap k*ey generation*, yaitu matriks kunci *involutory* K, yang dipertukarkan secara rahasia memakai konsep kriptografi simetris. Lalu, terdapat kunci publik N yang di dapat dari perkalian dua buah bilangan prima $p \neq q$ dan kunci publik e yang dibangkitkan dengan beberapa syarat, seperti yang sudah dijelaskan pada tahap enkripsi di poin sub-bab 2.12 (Hill-RSA), begitu pula dengan kunci privat d.

```
===== KEY GENERATION =====
Matriks K (secret exchange)
K: [[375, 374], [6493, 6494]] mod 6869

p: 4673
q: 8627
N: 40313971
Kunci privat d: 28352989
Kunci publik e: 12736885 dan N: 40313971
```

Gambar 4.1 *Key Generation*

Setelah program memberi *output* berupa *key generation*, maka pada tahap enkripsi, terdapat inputan *plaintext* yang harus diisi terlebih dahulu agar program dapat berjalan yang dapat dilihat pada Gambar 4.2 dan syarat inputannya adalah berdasarkan tabel ASCII dari karakter ‘space’ hingga karakter ‘Z’.

```
===== ENCRYPTION =====
Masukkan plaintext (kapital):
```

Gambar 4.2 Input *Plaintext*

Program akan terus meminta inputan *plaintext* yang valid, seperti Gambar 4.3 apabila inputan yang diberikan tidak dalam rentang karakter ‘space’ hingga karakter ‘Z’ berdasarkan tabel ASCII.

```
===== ENCRYPTION =====
Masukkan plaintext (kapital): kita

INPUT TIDAK VALID. Coba lagi!


Masukkan plaintext (kapital): lightweight

INPUT TIDAK VALID. Coba lagi!


Masukkan plaintext (kapital): pubLIC

INPUT TIDAK VALID. Coba lagi!


Masukkan plaintext (kapital): power12

INPUT TIDAK VALID. Coba lagi!


Masukkan plaintext (kapital): POWER12
Plaintext matrix: [[5857, 6027], [6547, 2810]]
```

Gambar 4.3 Input *Plaintext* Valid

Plaintext yang berhasil diinput akan dikonversi ke dalam bentuk matriks dan setiap dua karakter akan digabung. Sehingga kode rahasia dari *plaintext* lebih aman karena dibuat per-blok dengan dua karakter, seperti Gambar 4.4.

```
Masukkan plaintext (kapital): POWER12
Plaintext matrix: [[5857, 6027], [6547, 2810]]

Enkripsi-1 (hc): [[1509, 207], [6694, 4694]] mod 6869
Enkripsi-2 (rsa): [[23843907, 17893269], [18631690, 38167555]] mod 40313971
```

Gambar 4.4 Proses Enkripsi

Gambar 4.5 merupakan tahap dekripsi di mana seharusnya tahap dekripsi Hill Cipher dilakukan dengan memanfaatkan matriks *inverse K*. Namun, pada algoritma modifikasi ini dimanfaatkan sebuah konsep matriks *involutory* yang akan mengeliminasi *inverse* sehingga program tidak memerlukan operasi tambahan untuk menghitung matriks *inverse* dan ini akan membuat algoritma menjadi lebih efisien waktu secara komputasi.

```
===== DECRYPTION =====
Dekripsi-1 (rsa): [[1509, 207], [6694, 4694]] mod 40313971
Dekripsi-2 (hc): [[5857, 6027], [6547, 2810]] mod 6869

Transpose dekripsi: [[5857, 6547], [6027, 2810]]
Plaintext: POWER12
```

Gambar 4.5 Proses Dekripsi

Dan pada Gambar 4.6 program dinyatakan berakhir dengan menghasilkan waktu eksekusi yang dilalui program dalam satuan *millisecond* (*ms*). Dimana pada Gambar 4.6 waktu eksekusi yang dihasilkan adalah 4411 *ms*.

```
Selesai.
Waktu eksekusi: 4411.671161651611 ms
```

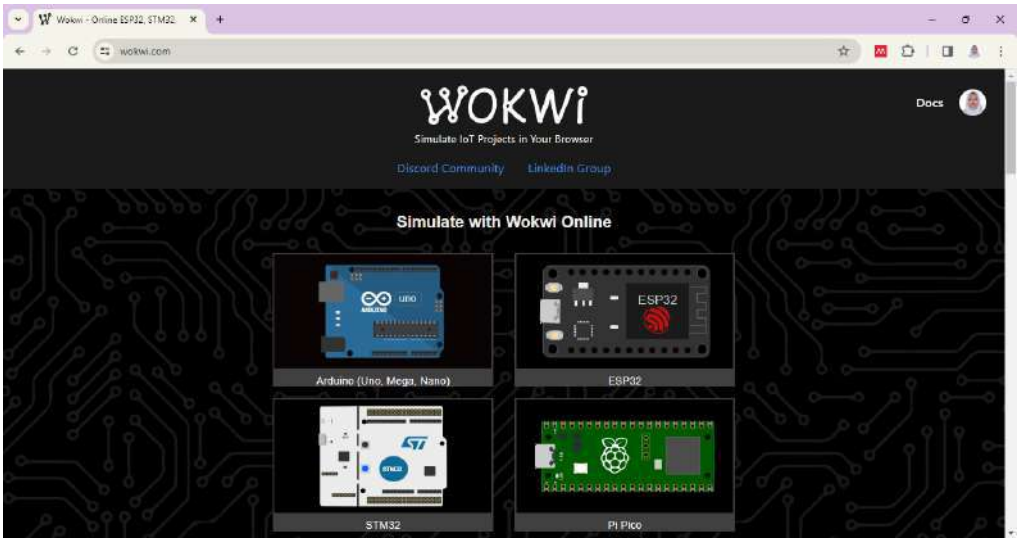
Gambar 4.6 Waktu Eksekusi Program

4.1.3 Emulator ESP32 berbasis MicroPython

Emulator mikrokontroler ESP32 adalah alat pengembangan yang memungkinkan untuk melakukan pengujian terhadap suatu program dengan MicroPython tanpa terlebih dahulu melakukannya di mikrokontroler berbentuk fisik, salah satunya ESP32. Dengan menggunakan emulator, pengembang dapat menguji dan

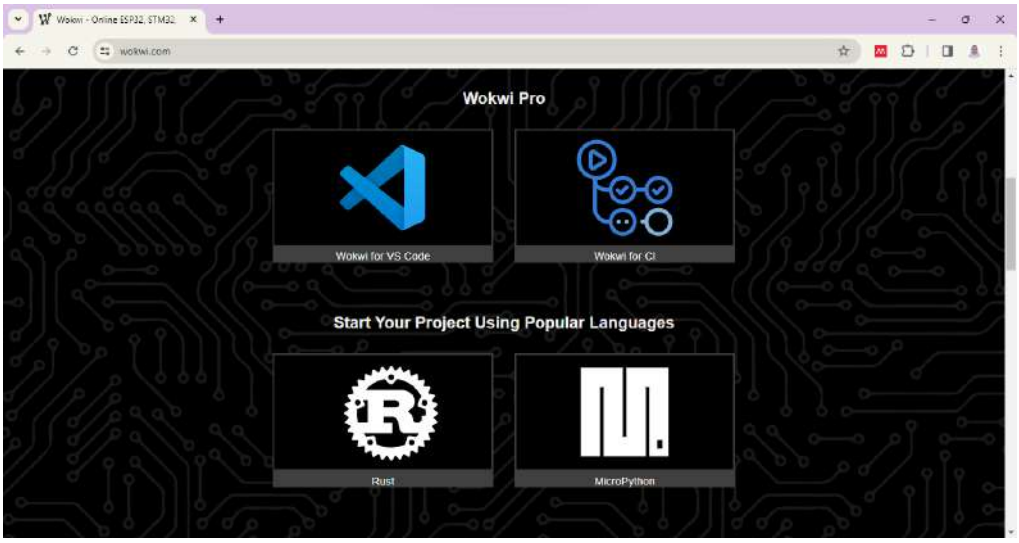
mengembangkan algoritma ataupun program secara. Hal ini, tentu saja mempermudah pengembangan dalam mengidentifikasi masalah dan penyesuaian kode sebelum diimplementasikan pada perangkat ESP32 yang sebenarnya. Emulator ESP32 berbasis MicroPython menyediakan lingkungan pengujian dengan bahasa pemrograman yang lebih tinggi dan lebih fleksibel.

Penelitian ini menggunakan emulator *virtual online* yang tersedia di *browser*, yaitu Wokwi. Pada Gambar 4.7, terlihat bahwa Wokwi menyediakan banyak mikrokontroler populer, salah satunya mikrokontroler ESP32.



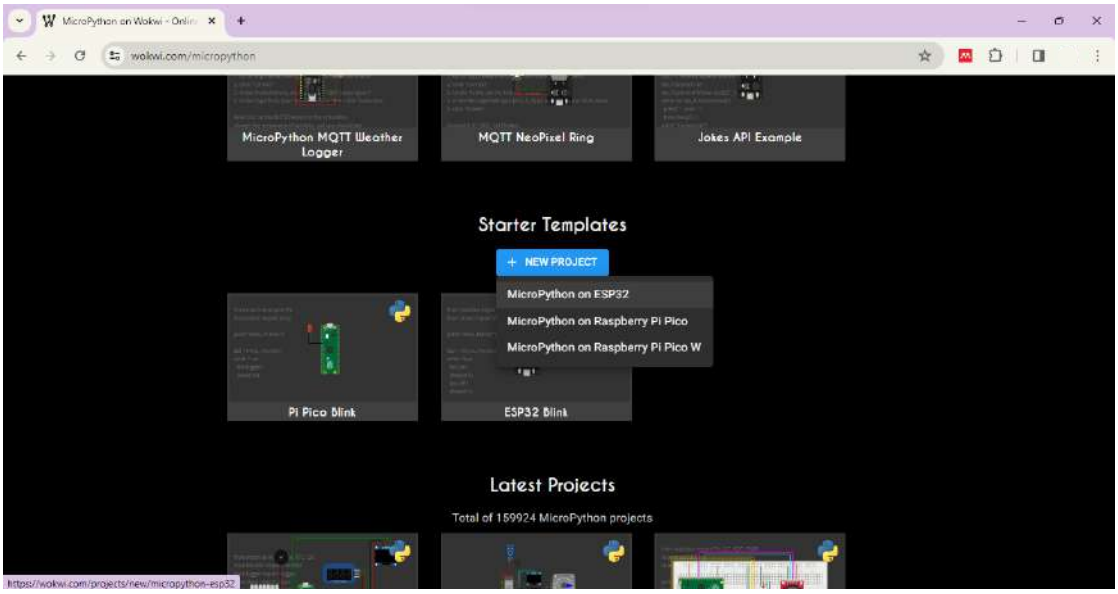
Gambar 4.7 Tampilan *Website* Wokwi

Terdapat dua bahasa pemrograman yang disediakan Wokwi, lihat pada Gambar 4.8, salah satunya adalah MicroPython yang digunakan untuk menguji algoritma Hill-RSA.

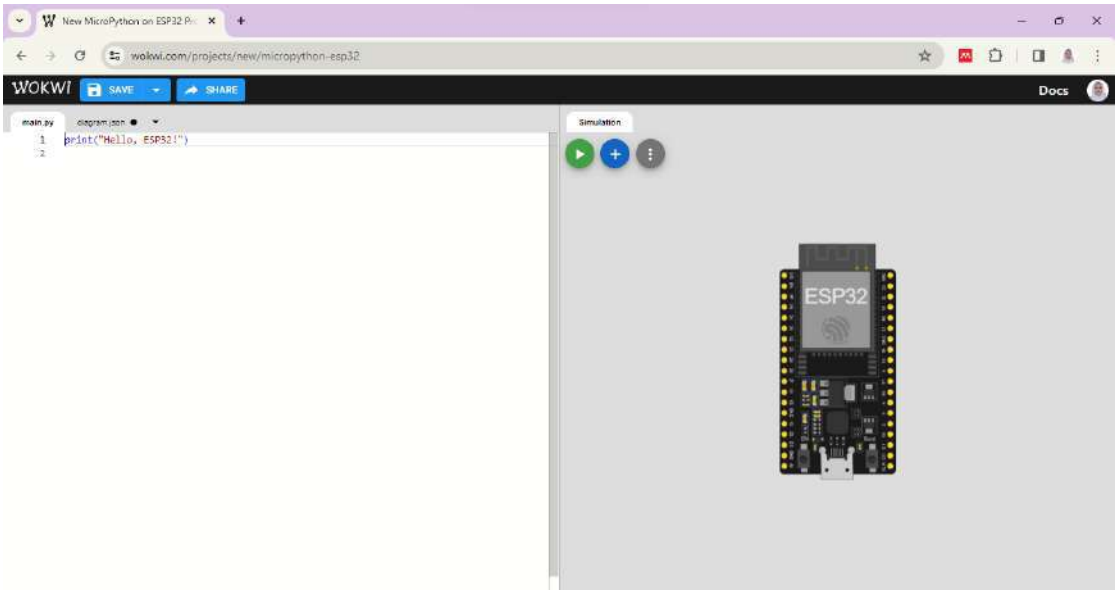


Gambar 4.8 Tampilan Bahasa Pemrograman Wokwi

Pengujian algoritma Hill-RSA pada mikrokontroler ESP32 untuk melihat seberapa efisien algoritma tersebut dalam memberi fungsi keamanan pada perangkat IoT dengan mengujinya terlebih dahulu di emulator ESP32 yang disediakan oleh Wokwi. Berikut merupakan tampilan jika memulai membuat proyek baru pada Wokwi, lihat pada Gambar 4.9 dan Gambar 4.10.

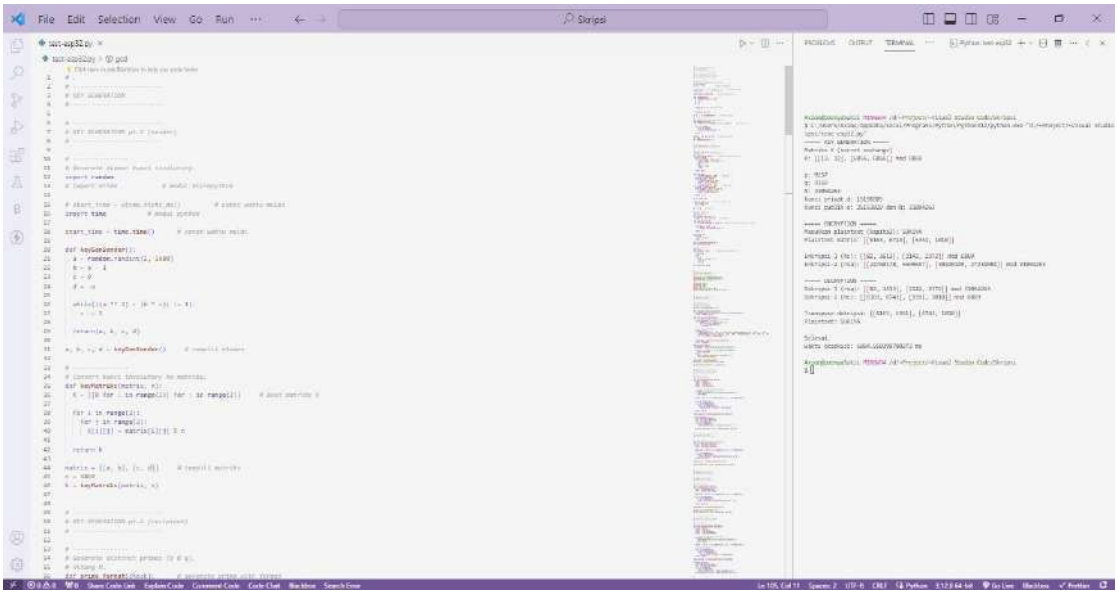


Gambar 4.9 Tampilan *New Project* Wokwi

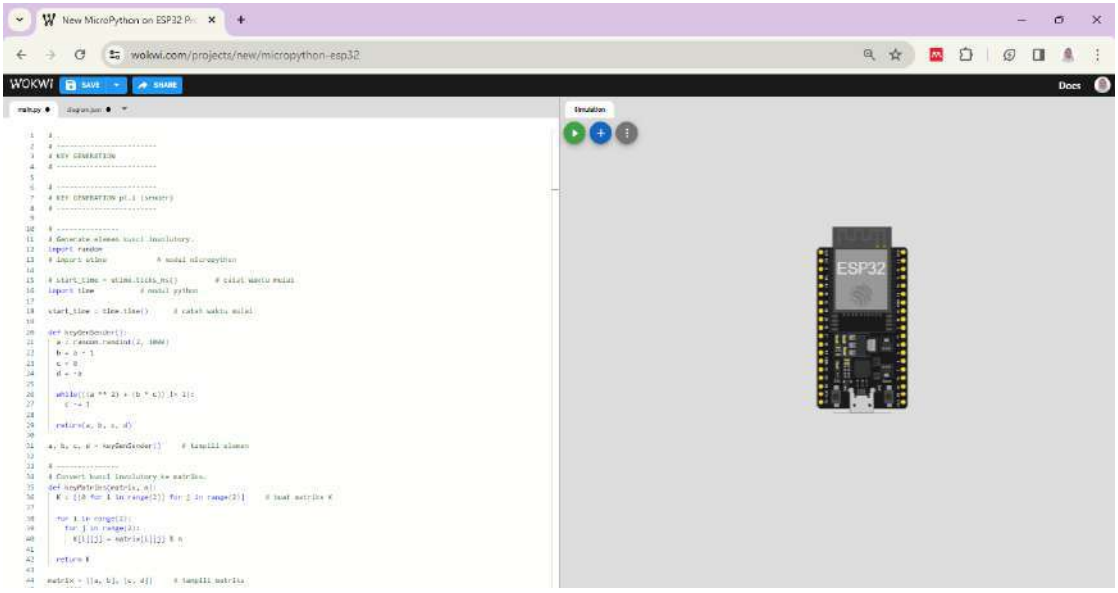


Gambar 4.10 Tampilan *Blank Project* Wokwi

Program algoritma Hill-RSA (Gambar 4.11) yang sudah dibuat, disalin dan ditempel ke *file* ‘main.py’ yang telah disediakan Wokwi, lihat pada Gambar 4.12



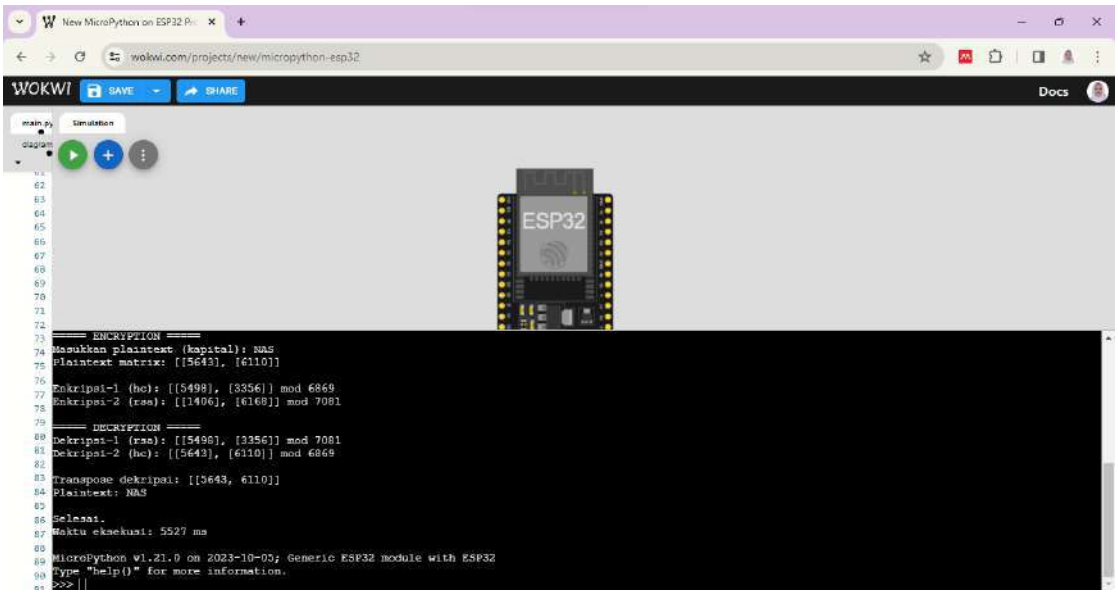
Gambar 4.11 Program Algoritma Hill-RSA



Gambar 4.12 Tampilan Program pada ESP32

Program pada penelitian ini pertama kali dirancang dengan bahasa Python. Namun, tidak semua *library* Python tersedia di MicroPython, salah satu *library* ‘time’. Agar dapat berjalan di emulator ESP32, *library* ‘time’ diganti dengan ‘utime’ untuk mengukur waktu eksekusi program. Terlihat pada *output*

Gambar 4.13 bahwa program berhasil menghitung waktu eksekusi dalam satuan *millisecond* (*ms*), yaitu 5527 *ms*.



Gambar 4.13 Tampilan Eksekusi Program pada ESP32

4.2 Pengujian

Penelitian ini melakukan proses pengujian terhadap program yang telah dirancang dan dikembangkan. Pengujian ini dilakukan dengan kriteria sebagai berikut:

1. Menganalisis waktu proses eksekusi program pada Visual Studio Code dan Emulator ESP32 dengan satuan *millisecond* (*ms*).
2. Menghitung kemungkinan kompleksitas algoritma Hill-RSA dengan Teori Cormen.

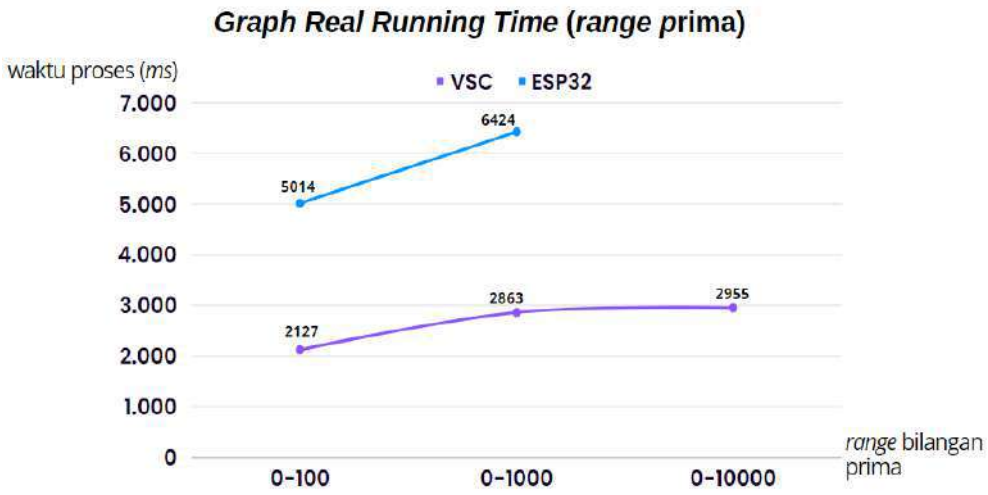
4.2.1 Pengujian real running time program

Pengujian *real running time* dilakukan untuk menganalisis waktu proses mengeksekusi program algoritma Hill-RSA dengan ketetapan ukuran matriks, yaitu 2×2.

1. Nilai kunci publik N diperoleh dari perkalian dua buah bilangan prima p dan q, di mana pada pengujian ini dilakukan percobaan dengan rentang nilai 100 hingga 10000 serta waktu rata-rata dari *running time* dalam satuan *millisecond* (*ms*).

Tabel 4.3 Waktu Eksekusi Program (*Range* Prima)

	Range *p* dan *q*	Uji 1 (*ms*)	Uji 2 (*ms*)	Uji 3 (*ms*)	Uji 4 (*ms*)	Rata-rata (*ms*)
VSC	**0 - 100**	2809	2315	1725	1662	2127
	0 - 1000	2894	2645	2780	3133	2863
	0 - 10000	2772	2273	3371	3404	2955
Emulator ESP32	**0 - 100**	3429	6916	4130	5579	5014
	0 - 1000	9155	4282	6325	5935	6424
	0 - 10000	-	-	-	-	-



Gambar 4.14 Grafik Waktu Eksekusi Program (*Range* Prima)

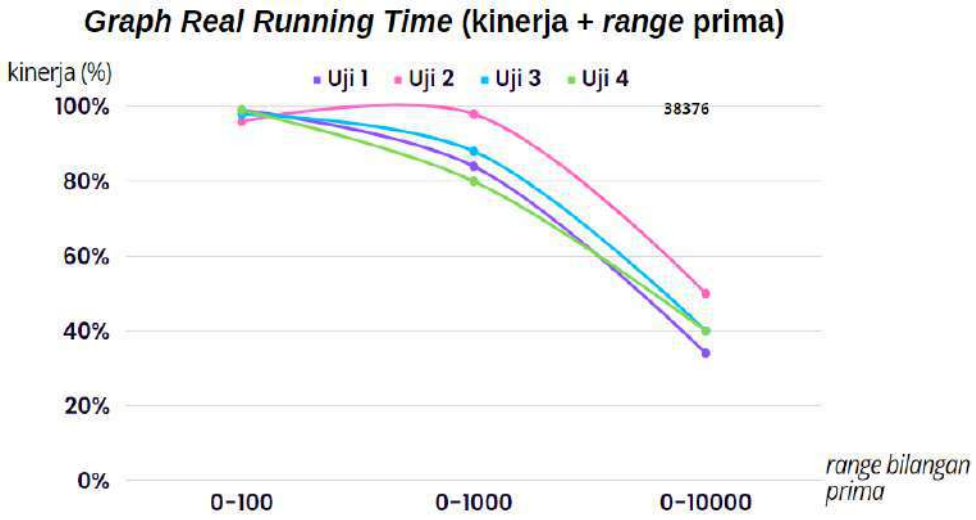
Pada Tabel 4.3 dan Gambar 4.14 diperlihatkan bahwa Visual Studio Code memiliki rata-rata waktu eksekusi program tidak sampai 3000 *ms*, sedangkan pada emulator ESP32 dalam rentang bilangan prima 0-100 sudah mencapai 5014 *ms* dan dalam rentang bilangan prima 0-10000 menunjukkan bahwa performa kinerja emulator ESP32 tidak sampai 50% dan belum memberikan *output* apapun selama 1 menit, yang ditunjukkan pada Gambar 4.15.

Gambar 4.15 Emulator ESP32 dengan *Range* Prima 10000

Pada Tabel 4.4 dan Gambar 4.16 ditunjukkan performa kinerja dari emulator ESP32 dengan menggunakan rentang bilangan prima p dan q yang sama pada Tabel 4.3.

Tabel 4.4 Performa Kinerja Emulator ESP32

Range p dan q	**Uji 1 (%)**	**Uji 2 (%)**	**Uji 3 (%)**	**Uji 4 (%)**
0 - 100	99% - 100%	96% - 100%	98% - 100%	99% - 100%
0 - 1000	84% - 99%	98% - 100%	98% - 100%	80% - 100%
0 - 10000	50% - 99%	40% - 70%	40% - 70%	40% - 60%



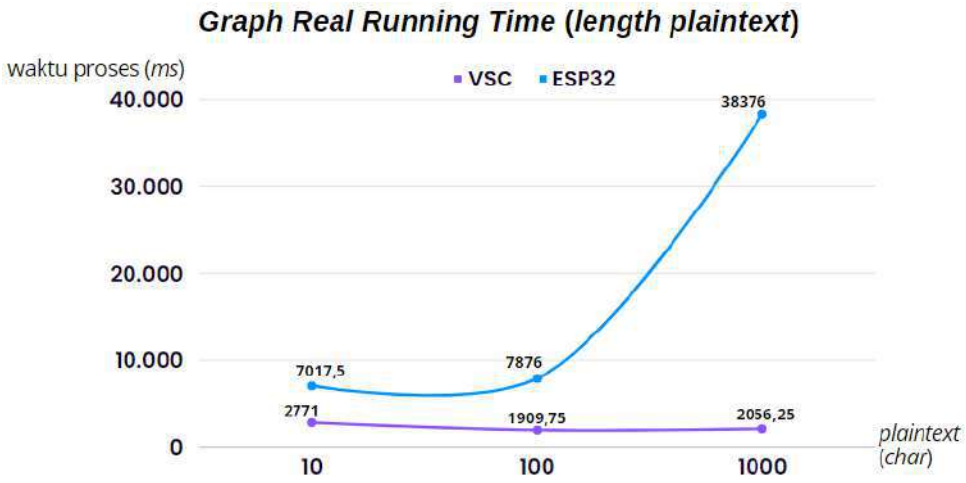
Gambar 4.16 Performa Kinerja Emulator ESP32

Dapat disimpulkan dari Tabel 4.4 dan Gambar 4.16 bahwa bilangan prima dengan rentang 0-10000 performa kinerja emulator ESP32 menurun dan berdasarkan Tabel 4.3 semua pengujian yang dilakukan tidak memberi *output* program.

2. Menganalisis *real running time* berdasarkan panjang karakter *plaintext* yang berbeda dimulai dari 10, 100, dan 1000 karakter.

Tabel 4.5 Waktu Eksekusi Program (*Length Plaintext*)

	***Plaintext* (*char*)**	**Uji 1 (*ms*)**	**Uji 2 (*ms*)**	**Uji 3 (*ms*)**	**Uji 4 (*ms*)**	**Rata-rata (*ms*)**
VSC	**10**	5010	2893	1716	1465	2771
	100	2696	1650	1488	1805	1909,75
	1000	2786	1693	2139	1607	2056,25
Emulator ESP32	**10**	9823	5624	6025	6598	7017,5
	100	8751	8091	5881	8781	7876
	1000	52731	28431	37211	35131	38376



Gambar 4.17 Grafik Waktu Eksekusi Program (*Length Plaintext*)

Pada Tabel 4.5 dan Gambar 4.17 diperlihatkan bahwa waktu proses eksekusi program lebih membutuhkan waktu seiring dengan banyaknya karakter *plaintext* yang dimasukkan. Namun, tidak terlalu ada pengaruh yang signifikan antara banyaknya *plaintext* yang dimasukkan dengan performa kinerja pada emulator ESP32, seperti yang ditunjukkan pada Gambar 4.18.

Gambar 4.18 Emulator ESP32 dengan Panjang Karakter 1000

4.2.2 *Theoretical running time dengan teori Cormen*

Teori Cormen merupakan teori yang digunakan untuk memperkirakan dan menghitung kompleksitas waktu dalam menganalisis suatu algoritma dengan memanfaatkan bentuk tabel dan notasi θ (*theta*). Tabel Cormen dibuat oleh Thomas H. Cormen yang memberikan kemungkinan terburuk atas waktu eksekusi program dan membantu mengetahui tingkat efisiensi dari suatu algoritma dengan mempertimbangkan ukuran input. Berikut merupakan penerapan *theoretical running time* dalam program algoritma Hill-RSA:

1. Kompleksitas algoritma *generate random* elemen matriks kunci

Tabel 4.6 Kompleksitas *Generate* Elemen Matriks

```
FUNCTION keyGenSender():

  SET a = random.randint(2, 1000)
  SET b = a - 1
  SET c = 0
  SET d = -a

  WHILE(((a ** 2) + (b * c)) != 1):
    c -= 1

  RETURN(a, b, c, d)
```

C	#	C#
C_1	1	C_1
C_1	1	C_1
C_1	1	C_1
C_1	1	C_1
C_2	n^2	$C_2 n^2$
C_3	n^2	$C_3 n^2$
C_4	1	C_4

$$\therefore T() = 4C_1 + (C_2 + C_3)n^2 + C_4$$
$$= \theta(n^2) \text{ (kuadratik)}$$

2. Kompleksitas algoritma *convert* elemen ke bentuk matriks

Tabel 4.7 Kompleksitas Elemen ke Matriks

```
FUNCTION keyMatriks(matrix, n):

  SET K = [[0 FOR i IN range(2)] FOR j
IN range(2)]

  FOR i IN range(2):
    FOR j IN range(2):
      SET K[i][j] = matrix[i][j] % n

  RETURN K
```

C	#	C#
C_1	1	C_1
C_2	2	$2C_2$
C_2	4	$4C_2$
C_1	4	$4C_1$
C_3	1	C_3

$$\therefore T(m,n) = C_1 + 2C_2 + 4C_2 + 4C_1 + C_3$$
$$= \theta(1) \text{ (konstan)}$$

3. Kompleksitas algoritma Fermat's Little Theorem

Tabel 4.8 Kompleksitas *Generate Prime*

```
FUNCTION prime_fermat(check):

  SET hasil = False

  FOR a IN range(2, check - 1):
    IF pow(a, check - 1, check)
EQUALS 1:
      SET hasil = True
    ELSE:
      SET hasil = False
      break

  RETURN hasil
```

C	#	C#
C_1	1	C_1
C_2	n - 1	C_2(n - 1)
C_3	n - 1	C_3(n - 1)
C_1	n - 1	C_1(n - 1)
C_4	n - 1	C_4(n - 1)
C_1	n - 1	C_1(n - 1)
C_5	n - 1	C_5(n - 1)
C_6	1	C_6

$$\therefore T(n) = C_1 + C_2n - C_2 + C_3n - C_3 + C_1n - C_1 + C_4n - C_4 + C_1n - C_1 + C_5n - C_5 + C_6$$

$$= (C_2 + C_3 + C_1 + C_4 + C_1 + C_5)n - C_2 - C_3 - C_1 - C_4 - C_1 - C_5 + C_1 + C_6$$

$$= \theta(n) \text{ (linear)}$$

4. Kompleksitas algoritma *generate random* p dan q

Tabel 4.9 Kompleksitas *Generate Random* (p dan q)

```
FUNCTION keys_rsa():

  SET p = random.randint(2, 100)
  SET p_check = prime_fermat(p)
  WHILE p_check EQUALS False:
    SET p = random.randint(2, 100)
    SET p_check = prime_fermat(p)

  SET q = random.randint(2, 100)
  SET q_check = prime_fermat(q)
  WHILE q_check EQUALS False or p
EQUALS q:
    SET q = random.randint(2, 100)
    SET q_check = prime_fermat(q)

  RETURN p, q
```

C	#	C#
C_1	1	C_1
C_1	1	C_1
C_2	n	C_2n
C_1	n	C_1n
C_1	n	C_1n
C_1	1	C_1
C_1	1	C_1
C_2	n	C_2n
C_1	n	C_1n
C_1	n	C_1n
C_3	1	C_3

$$\therefore T() = 4C_1 + (2C_2 + 4C_1)n + C_3$$

$$= \theta(n) \text{ (linear)}$$

5. Kompleksitas algoritma *generate* kunci publik e

Tabel 4.10 Kompleksitas Fungsi Totient

```
FUNCTION totient_function(p, q):

  RETURN (p - 1) * (q - 1)
```

C	#	C#
C_1	1	C_1

$\therefore T(m,n) = C_1$

$= \theta(1)$ (konstan)

Tabel 4.11 Kompleksitas *Euclidean* GCD

```
FUNCTION gcd(a, b):

  WHILE b:
    SET a, b = b, a % b

  RETURN a
```

C	#	C#
C_1	n	C_1n
C_2	n	C_3n
C_3	1	C_3

$\therefore T(m,n) = (C_1 + C_2)n + C_3$

$= \theta(n)$ (linear)

Tabel 4.12 Kompleksitas *Generate* e

```
FUNCTION generate_e(phi_N):

  SET e = random.randrange(3, phi_N,
2)
  WHILE gcd(e, phi_N) != 1:
    SET e = random.randrange(3, phi_N,
2)

  RETURN e

SET phi_N = totient_function(p, q)
SET e = generate_e(phi_N)
```

C	#	C#
C_1	1	C_1
C_2	n	C_2n
C_1	n	C_1n
C_3	1	C_3
C_1	1	C_1
C_1	n	C_1n

$\therefore T(n) = 2C_1 + (C_2 + 2C_1)n + C_3$

$= \theta(n)$ (linear)

6. Kompleksitas algoritma menghitung kunci privat d

Tabel 4.13 Kompleksitas *Inverse Modulo*

```
FUNCTION inverse(x, mod):

  IF x EQUALS 0:
```

C	#	C#
C_1	1	C_1

```
    RETURN 0

  ELSE:
    SET y = 1
    WHILE x * y % mod != 1:
      y += 1

    RETURN y

SET d = inverse(e, phi_N)
```

C_2	1	C_2
C_3	1	C_3
C_4	1	C_4
C_5	n	C_5n
C_6	n	C_6n
C_2	1	C_2
C_4	1	C_4

$$\therefore T(m,n) = C_1 + 2C_2 + C_3 + 2C_4 + (C_5 + C_6)n$$
$$= \theta(n) \text{ (linear)}$$

7. Kompleksitas algoritma *convert* teks ke matriks (ASCII)

Tabel 4.14 Kompleksitas Matriks *Transpose*

```
FUNCTION transpose_result(text_matrix):

  SET transpose_matrix = [list(row) FOR
row IN zip(*text_matrix)]

  RETURN transpose_matrix
```

C	#	C#
C_1	1	C_1
C_2	1	C_2

$$\therefore T(n) = C_1 + C_2$$
$$= \theta(1) \text{ (konstan)}$$

Tabel 4.15 Kompleksitas Teks ke Matriks

```
FUNCTION text_to_matrix(plaintext):

  IF len(plaintext) % 4 != 0:
    SET tambah_x = 4 - len(plaintext) %
4
    FOR i IN range(tambah_x):
        plaintext += ' '

  SET text_matrix = [
    [(ord(plaintext[i]) - ord(' ') +
10) * 100 + (ord(plaintext[i + 1]) -
ord(' ') + 10),
     (ord(plaintext[i + 2]) - ord(' ')
+ 10) * 100 + (ord(plaintext[i + 3]) -
ord(' ') + 10)]
     FOR i IN range(0, len(plaintext),
4)
    ]
```

C	#	C#
C_1	1	C_1
C_2	1	C_2
C_3	n	C_3n
C_4	1	C_4
C_2	1	C_2
C_3	n	C_3n

```
  SET transposed =
transpose_result(text_matrix)

  RETURN transposed
```

C_2	1	C_2
C_5	1	C_5

$\therefore T(n) = C_1 + 3C_2 + 2C_3n + C_4 + C_5$

$= \theta(n)$ (linear)

8. Kompleksitas enkripsi Hill Cipher

Tabel 4.16 Kompleksitas Enkripsi Hill Cipher

```
FUNCTION multiply_matrices(K,
matrix):

  SET rows_a = len(K)
  SET cols_a = len(K[0])
  SET rows_b = len(matrix)
  SET cols_b = len(matrix[0])

  IF cols_a != rows_b:
    PRINT("Perkalian matriks tidak
dapat dilakukan.")

    RETURN None

  SET result = [[0 FOR _ IN
range(cols_b)] FOR _ IN
range(rows_a)]

  FOR i IN range(rows_a):
    FOR j IN range(cols_b):
      FOR k IN range(cols_a):
        result[i][j] += K[i][k] *
matrix[k][j]

  RETURN result

SET result_matrix =
multiply_matrices(K, pt_matrix)

IF result_matrix:
  SET row = len(result_matrix)
  SET col = len(result_matrix[0])
  FOR i IN range(row):
    FOR j IN range(col):
      SET result_matrix[i][j] =
result_matrix[i][j] % n
```

C	#	C#
C_1	1	C_1
C_1	1	C_1
C_1	1	C_1
C_1	1	C_1
C_2	1	C_2
C_3	1	C_3
C_4	1	C_4
C_1	1	C_1
C_5	m	C_5m
C_5	m.n	C_5mn
C_5	m.n.m	C_5m^{2}n
C_6	m.n.m	C_6m^{2}n
C_4	1	C_4
C_1	1	C_1
C_2	1	C_2
C_1	1	C_1
C_1	1	C_1
C_5	r	C_5r
C_5	r.c	C_5rc
C_1	r.c	C_1rc

$\therefore T(m,n) = 8C_1 + 2C_2 + C_3 + 2C_4 + C_5m + C_5mn + (C_5 + C_6)\mathrm{m}^2n$

$+C_5r + (C_5 + C_1)rc$

$= \theta(\mathrm{m}^2n)$ (kuadratik)

9. Kompleksitas enkripsi RSA

Tabel 4.17 Kompleksitas Enkripsi RSA

```
FUNCTION rsa_matrices(result_matrix):

  SET rows = len(result_matrix)
  SET cols = len(result_matrix[0])

  SET hasil_rsa = [[0 FOR _ IN
range(cols)] FOR _ IN range(rows)]

  FOR i IN range(rows):
    FOR j IN range(cols):
      SET hasil_rsa[i][j] =
pow(result_matrix[i][j], e, N)

  RETURN hasil_rsa

SET encrypt_rsa =
rsa_matrices(result_matrix)
```

C	#	C#
C_1	1	C_1
C_1	1	C_1
C_1	1	C_1
C_2	r	C_2r
C_2	r.c	C_2rc
C_1	r.c	C_1rc
C_3	1	C_3
C_1	1	C_1

$$\therefore T(n) = 4C_1 + C_2r + (C_1 + C_2)rc + C_3$$
$$= \theta(rc) \text{ (linear)}$$

10. Kompleksitas dekripsi RSA

Tabel 4.18 Kompleksitas Dekripsi RSA

```
FUNCTION rsa_decrypt(encrypt_rsa):

  SET rows = len(encrypt_rsa)
  SET cols = len(encrypt_rsa[0])

  SET hasil_rsa = [[0 FOR _ IN
range(cols)] FOR _ IN range(rows)]

  FOR i IN range(rows):
    FOR j IN range(cols):
      SET hasil_rsa[i][j] =
pow(encrypt_rsa[i][j], d, N)

  RETURN hasil_rsa

SET decryptRSA =
rsa_decrypt(encrypt_rsa)
```

C	#	C#
C_1	1	C_1
C_1	1	C_1
C_1	1	C_1
C_2	r	C_2r
C_2	r.c	C_2rc
C_1	r.c	C_1rc
C_3	1	C_3
C_1	1	C_1

$$\therefore T(n) = 4C_1 + C_2r + (C_1 + C_2)rc + C_3$$
$$= \theta(r) \text{ (linear)}$$

11. Kompleksitas dekripsi Hill Cipher

Tabel 4.19 Kompleksitas Dekripsi Hill Cipher

```
FUNCTION multiply_matrices(K,
decryptHC):

  SET rows_a = len(K)
  SET cols_a = len(K[0])
  SET rows_b = len(decryptHC)
  SET cols_b = len(decryptHC[0])

  IF cols_a != rows_b:
    PRINT("Perkalian matriks tidak
dapat dilakukan.")

    RETURN None

  SET result = [[0 FOR _ IN
range(cols_b)] FOR _ IN
range(rows_a)]

  FOR i IN range(rows_a):
    FOR j IN range(cols_b):
      FOR k IN range(cols_a):
        result[i][j] += K[i][k] *
decryptHC[k][j]

  RETURN result

SET hasil_decrypt =
multiply_matrices(K, decryptRSA)

IF hasil_decrypt:
  SET row = len(hasil_decrypt)
  SET col = len(hasil_decrypt[0])

  FOR i IN range(row):
    FOR j IN range(col):
      SET hasil_decrypt[i][j] =
hasil_decrypt[i][j] % n
  PRINT(hasil_decrypt)
```

C	#	C#
C_1	1	C_1
C_1	1	C_1
C_1	1	C_1
C_1	1	C_1
C_2	1	C_2
C_3	1	C_3
C_4	1	C_4
C_1	1	C_1
C_5	m	C_5m
C_5	m.n	C_5mn
C_5	m.n.m	C_5m^{2}n
C_6	m.n.m	C_6m^{2}n
C_4	1	C_4
C_1	1	C_1
C_2	1	C_2
C_1	1	C_1
C_1	1	C_1
C_5	r	C_5r
C_5	r.c	C_5rc
C_1	r.c	C_1rc
C_3	1	C_3

$$\therefore T(m,n) = 8C_1 + 2C_2 + 2C_3 + 2C_4 + C_5m + C_5mn + (C_5 + C_6)\mathrm{m}^2n$$
$$+C_5r + (C_5 + C_1)rc$$
$$= \theta(\mathrm{m}^2n) \text{ (kuadratik)}$$

12. Kompleksitas algoritma *convert* matriks ke teks (ASCII)

Tabel 4.20 Kompleksitas Matriks *Transpose*

```
FUNCTION
transpose_decrypt(hasil_decrypt):

  SET transpose_matrix = [list(row) FOR
row IN zip(*hasil_decrypt)]

  RETURN transpose_matrix
```

C	#	C#
C_1	1	C_1
C_2	1	C_2

$\therefore T(n) = C_1 + C_2$

$= \theta(1)$ (konstan)

Tabel 4.21 Kompleksitas Matriks ke Teks

```
FUNCTION matrix_to_text(test_t):

  SET row = len(test_t)
  SET col = len(test_t [0])

  SET split_matrix = [[0 FOR _ IN
range(col*2)] FOR _ IN range(row)]

  SET plaintext TO ""

  FOR i IN range(row):
    SET split_matrix[i][0] =
test_t[i][0] // 100
    SET split_matrix[i][1] =
test_t[i][0] % 100
    SET split_matrix[i][2] =
test_t[i][1] // 100
    SET split_matrix[i][3] =
test_t[i][1] % 100

  SET row_split = len(split_matrix)
  SET col_split = len(split_matrix[0])
  FOR i IN range(row_split):
    FOR j IN range(col_split):
      plaintext +=
chr(split_matrix[i][j] - 10 + 32)

  RETURN plaintext
```

C	#	C#
C_1	1	C_1
C_1	1	C_1
C_1	1	C_1
C_1	1	C_1
C_2	n	C_2n
C_1	n	C_1n
C_1	n	C_1n
C_1	n	C_1n
C_1	n	C_1n
C_1	1	C_1
C_1	1	C_1
C_2	n	C_2n
C_2	n^2	C_2n^2
C_3	n^2	C_3n^2
C_4	1	C_4

$\therefore T(n) = 6C_1 + (2C_2 + 4C_1)n + (C_2 + C_3)n^2 + C_4$

$= \theta(n^2)$ (kuadratik)

BAB 5

PENUTUP

5.1 Kesimpulan

Berdasarkan tahap implementasi yang telah dilakukan pada penelitian *lightweight public key cryptography* dengan algoritma Hill-RSA dan pengujian yang dilakukan pada emulator ESP32 berbasis MicroPython, disimpulkan bahwa:

1. Algoritma Hill-RSA terbukti dapat melakukan proses kriptografi, yaitu *key generation*, enkripsi, dan dekripsi dan terbukti dapat melakukan proses enkripsi pada *plaintext* dengan karakter yang panjang.
2. Algoritma Hill-RSA terbukti memiliki waktu komputasi yang lebih efisien karena memanfaatkan konsep matriks *involutory* di mana $K = K^{-1}$, sehingga tidak perlu operasi tambahan untuk matriks *inverse* pada proses dekripsi Hill Cipher.
3. Algoritma Hill-RSA memiliki waktu proses yang cukup baik, tetapi tidak cukup efisien untuk diterapkan pada perangkat IoT, seperti mikrokontroler ESP32 berbasis MicroPython karena ketika rentang bilangan prima adalah 10000, performa kinerja emulator menurun dengan nilai tidak sampai 50% dan dalam waktu eksekusi selama 1 menit lebih tidak menghasilkan *output*.
4. R*eal running time program* akan berpengaruh dengan banyaknya karakter *plaintext* yang dimasukkan. Kinerja emulator ESP32 menunjukkan performa sebesar 70% - 99%, sehingga antara banyaknya *plaintext* yang dimasukkan dengan performa kinerja ESP32 tidak terlalu berpengaruh secara signifikan.
5. Algoritma Hill-RSA memiliki kompleksitas waktu konstan ($\theta(1)$), linear ($\theta(n)$), dan kuadratik ($\theta(n^2)$).
6. Waktu konstan ($\theta(1)$) didapat ketika mengubah elemen yang dibangkitkan ke dalam bentuk matriks, membangkitkan kunci publik e, dan matriks *transpose*.
7. Waktu kuadratik ($\theta(n^2)$; $\theta(\mathrm{m}^2 n)$) didapat ketika membangkitkan elemen *random* untuk matriks kunci, proses enkripsi dan dekripsi dengan Hill Cipher,

dan sisanya memiliki kompleksitas waktu linear $(\theta(n); \theta(rc); \theta(r))$ termasuk proses enkripsi dan dekripsi algoritma RSA.

8. Algoritma Hill-RSA dianggap dapat memiliki ukuran input yang tidak terbatas berdasarkan analisis waktu kompleksitas, tetapi algortima akan lebih efisien jika input yang diberi berukuran sedang dan ukuran input yang besar akan membuat kompleksitas algortima menjadi kurang efisien.

5.2 Saran

Saran yang diberikan untuk penelitian selanjutnya adalah sebagai berikut:

1. Disarankan untuk melakukan penelitian lebih lanjut dengan mengimplementasikan pengujian pada IoT ataupun mikrokontroler secara fisik dan nyata.
2. Disarankan untuk melakukan penelitian lebih lanjut untuk mengetahui mengapa jika digunakan kunci p dan q pada rentang 10000, algoritma Hill-RSA tidak dapat berjalan jika diimplementasikan pada IoT.
3. Disarankan untuk melakukan penelitian dengan menerapkan algoritma *public key* lain, seperti algoritma ECC (Elliptic Curve Cryptography) dan RSA (Rivest-Shamir-Adleman) tanpa modifikasi dengan menggabungkan algoritma lain serta menggunakan kunci dan perhitungan yang lebih sederhana, sehingga sesuai untuk diimplementasikan pada perangkat dengan sumber daya terbatas.

DAFTAR PUSTAKA

Acharya, B., Patra, S. K., & Panda, G. (2009). Involutory, Permuted and Reiterative Key Matrix Generation Methods for Hill Cipher System. *International Journal of Recent Trends in Engineering, 1*(4), 106-108.

Ariyus, D. (2018). *Pengantar Ilmu Kriptografi: Teori, Analisis, dan Implementasi.* Yogyakarta: CV Andi Offset.

Basri. (2016). Kriptografi Simetris dan Asimetris dalam Perspektif Keamanan Data dan Kompleksitas Komputasi. *Jurnal Ilmiah Ilmu Komputer, 2*(2), 16-23.

Bufalo, M., Bufalo, D., & Orlando, G. (2021). A Note on the Computation of the Modular Inverse for Cryptography. *Axioms, 10*(2), 1-10. doi:https://doi.org/10.3390/axioms10020116

Dewi, N. P., Sembiring, D. J., Ginting, R. b., & Ginting, M. B. (2022). PENGAMANAN DATA DENGAN KRIPTOGRAFI HIBRIDA ALGORITMA. *Jurnal SYntax Admiration, 3*(2), 341-361.

Dewi, S. I. (2023). Automorfisma Graf Lengkap dan Graf Tangga. *Buletin Ilmiah Mat. Stat. dan Terapannya (Bimaster), 12*(6), 503-508.

El-Hajj, M., Mousawi, H., & Fadlallah, A. (2023). Analysis of Lightweight Cryptographic Algorithms on IoT Hardware Platform. *Future Internet*, 1-29.

Espressif. (2023, 12 12). *Espressif Systems-ESP32 Series Datasheet.* Retrieved from Espressif Website: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

Gaspar, G., Fabo, P., Kuba, M., Flochova, J., Dudak, J., & Florkova, Z. (2020). Development of IoT applications based on the MicroPython platform for Industry 4.0 implementation. *19th International Conference on Mechatronics - Mechatronika (ME)* (pp. 1-7). San Francisco: IEEE.

Gunawan, H., Budi, A. S., & Primananda, R. (2022). Penerapan Algoritma Diffie Hellman Key Exchange dalam Komunikasi Data Antarnode pada Wireless Sensor Network. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 6*(1), 197-203.

Hasoun, R. K., Khlebus, S. F., & Tayyeh, H. K. (2021). A New Approach of Classical Hill Cipher in Public Key Cryptography. *Internasional Journal of Nonlinear Analysis and Applications, 12*(2), 1071-1082.

Hidayat, A., & Alawiyah, T. (2013). Enkripsi dan Dekripsi Teks menggunakan Algoritma Hill Cipher dengan Kunci Matriks Persegi Panjang. *Jurnal Matematika Integratif, 9*(1), 39-51.

Hidayat, M. I. (2019). p-Grup pada Grup Dihedral. *The 3rd ELPSA Conference 2019* (pp. 1-5). -: INA-Rxiv Papers. doi:10.31227/osf.io/z4xnc

K.S, D., H.R, R., & C, Y. A. (2022). Non-Repudiation based Network Security System using Multiparty Computation. *(IJACSA) International Journal of Advanced Computer Science and Applications, 13*(3), 282-289.

Karatas, Z. Y., Luy, E., & Gonen, B. (2019). A Public Key Cryptosystem based on Matrices. *International Journal of Computer Applications, 182*(42), 47-50.

Khairani, T., Santoso, K. A., & Kamsyakawuni, A. (2021). Pengkodean Monoalphabetic Menggunakan Affine Cipher dengan Kunci Diffie-Hellman. *PRISMA 4, Prosiding Seminar Nasional Matematika* (pp. 553-559). Jember: Jurusan Matematika FMIPA UNNES.

Lee, D.-H., & Lee, I.-Y. (2020). A Lightweight Authentication and Key Agreement Schemes for IoT Environments. *Sensors,* 1-19.

Lie, I. R., & Alamsyah, D. (2023). Penerapan Algortima Diffie-Hellman pada Steganografi Least Significant Bit. *2nd MDP Student Conference (MSC) 2023* (pp. 234-243). Palembang: Universitas Multi Data Palembang.

Mulyanto, Y., Herfandi, & Kirana, R. C. (2022). Analisis Keamanan Wireless Local Area Network (Wlan) Terhadap Serangan Brute Force Dengan Metode Penetration Testing (Studi Kasus: RS H.L. manambai Abdulkadir). *JINTEKS (Jurnal Informatika Teknologi dan Sains), 4*(1), 26-35.

Munir, R. (2019). *Kriptografi: Edisi Kedua.* Bandung: Informatika Bandung.

Pavlyuk, I. I., & Sudoplatov, S. V. (2020). Approximations for Theories of Abelian Groups. *Mathematics and Statistics, 8*(2), 220-224.

Rachmawati, D., & Budiman, M. A. (2020). On Using The First Variant of Dependent RSA Encryption Scheme to Secure Text: A Tutorial. *Journal of Physics: Conference Series*, 1-6.

Rana, M., Mamun, Q., & Islam, R. (2021). Lightweight cryptography in IoT networks: A survey. *Future Generation Computer Systems, 129*, 77-89.

Romindo, & Ferawaty. (2021). Penerapan Algoritma Hybrid RSA Terhadap Pembangkit Kunci Diffie Hellman untuk Sistem Keamanan. *SATIN - Sains Dan Teknologi Informasi, 7*(2), 92-101. doi:https://doi.org/10.33372/stn.v7i2.783

Rosyid, M. F. (2017). *Aljabar Abstrak dalam Fisika.* Yogyakarta: Gadjah Mada University Press.

Samandari, N., Nazari, N. M., Olfat, J. A., Rafi, R., Azizi, Z., Ulfat, W. I., Niazi, M. J. (2023). Applications of Fermat's Little Theorem. *Turkish Journal of Computer and Mathematics Education, 14*(3), 209-215.

Santoso, Y. S. (2021). Message Security Using a Combination of Hill Cipher and RSA Algorithms. *Jurnal Matematika Dan Ilmu Pengetahuan Alam LLDikti Wilayah 1, 1*(1), 20-28.

Saputri, I., Wibowo, P., Ratricia, P., & Ikhwan, A. (2022). Pengamanan Pesan Menggunakan Metode Hill Chiper Dalam Keamanan Informasi. *Bulletin of Information Technology (BIT), 3*(4), 341-349.

Siahaan, M. D., & Siahaan, A. P. (2018). Application of Hill Cipher Algorithm in Securing Text Messages. *IJIRMF (International Journal For Innovative Research in Multidisciplinary Field), 4*(10), 55-58.

Side, S., & Syahrana. (2015). Aplikasi Invers Matriks dalam Pembentukan Pesan Rahasia. *Jurnal Teknosains, 9*(1), 27-39.

Sonea, A., & Cristea, I. (2023). Euler's Totient Function Applied to Complete Hypergroups. *AIMS Mathematics, 8*(4), 7731-7746.

Stallings, W. (2022). *Cryptography and Network Security Principles and Practice.* Pearson.

Strang, G. (2006). *Linear Algebra and Its Applications, 4th Edition.* Boston: Cengage Learning.

Sugondo, B. T., & Budi, A. S. (2021). Implementasi Algoritme Forkskinny pada Pengiriman Data Antara IoT Node dengan IoT Gateway. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer, 5*(4), 1543-1552.

Thakor, V. A., Razzaque, M. A., & Khandaker, M. R. (2021). Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities. *IEEE Access, 9*, 28177-28193.

Tristianto, D., & Anunwembun, B. J. (2022). Penerapan Keamanan Jaringan pada PT. Globalindo Perdana Sejahtera Menggunakan Metode Kriptografi. *Jurnal Sistem Cerdas dan Rekayasa (JSCR), 4*(2), JI 1-7.