BAB II
TINJAUAN PUSTAKA

2.1 Penelitian Sebelumnya

Penulis mengangkat beberapa penelitian serupa yang selanjutnya akan dijadikan acuan dan referensi penelitian dalam memperluas teori yang digunakan dalam penelitian. Berikut beberapa penelitian terdahulu yang peneliti jadikan sebagai referensi berupa jurnal penelitian dengan topik terkait.

Pada penelitian [9] yang berjudul "*Performance Analysis of Cryptographic Algorithms for Selecting Better Utilization on Resource Constraint Devices"* bertujuan untuk melakukan evaluasi kinerja dari algoritma kriptografi kunci simetris dan asimetris yang popular. Penelitian berfokus pada algoritma AES, RC4, Blowfish, CAST, 3DES, dan Twofish sebagai algoritma kinci simetris serta DSA dan ElGamal sebagai algoritma kunci asimetris. Parameter yang digunakan untuk melakukan perbandingan antara lain ukuran kunci, blok data, tipe data, dan kecepatan enkripsi dan dekripsi. Penelitian dilakukan untuk mengetahui efektivitas dari algoritma kriptografi yang berbeda untuk diterapkan dalam kehidupan nyata yang lebih berfokus pada waktu eksekusi yang cepat dan kebutuhan penggunaan memori yang lebuh sedikit.

Implementasi algoritma menggunakan Bahasa pemrograman Python dan perangkat lunak MATLAB untuk memproses data. Perpustakaan Python yang digunakan untuk memfasilitasi program antara lain PyCrypto dan Chilkat dengan masukan berupa file teks, suara, dan video. Perbandingan dilakukan berdasarkan parameter yang telah disebutkan diatas. Penelitian dilakukan dengan memvariasikan ukuran kunci yang berbeda untuk algoritma yang berbeda.

Hasil pengujian dari penelitian ini menunjukan bahwa algoritma DES, Twofish dan RC4 mengungguli algoritma lainnya dalam hal waktu enkripsi serta dekripsi. Ukuran kunci yang lebih besar memakan banyak

waktu komputasi yang mengarah pada penggunaan energi yang lebih tinggi. Penelitian terutama berfokus pada file teks untuk mengukur kinerja dari algoritma.

Pada penelitian [10] yang berjudul "Perbandingan Performa Kriptografi Asimetris pada Proses *Key Exchange*" peneliti melakukan analisis kinerja algoritma kriptografi asimetris dengan cara melakukan pengujian terhadap kecepatan proses pertukaran data dan penggunaan memori (RAM). Penelitian dilakukan untuk mendapatkan algoritma dengan performa terbaik.

Penelitian ini melakukan pengujian terhadap beberapa algoritma asimetris yaitu RSA, El-Gamal, dan Curve25519. Parameter yang digukan dalam penelitian adalah kinerja waktu dan konsumsi memori. Perpustakaan yang digunakan dalam proses pembuatan sistem untuk melakukan pengukuran waktu kinerja adalah modul Python time, dan untuk mengukur konsumsi memori peneliti menggunakan aplikasi Valgrind.

Hasil pengujian dari penelitian ini menunjukan bahwa ElGamal memiliki waktu kinerja yang paling cepat dengan kebutuhan memori yang paling efisien. Diikuti RSA dengan waktu kinerja yang kurang stabil jika dibandingkan dengan algoritma ElGamal dan ECC. Pada pengujian 256 bit RSA dan ElGamal memiliki kinerja yang hampir sama. Algoritma ECC dengan panjang kunci 256 bit memiliki tingkat keamanan yang setara dengan algoritma RSA dan ElGamal dengan panjang kunci 2048 bit.

Pada penelitian [11] yang berjudul *"Performance Evaluation of RSA and Elliptic Curve Cryptography in Wireless Sensor Networks"* yang bertujuan untuk melakukan analisis perbandingan kinerja algoritma kriptografi RSA dan Kurva Eliptik pada perangkat Sensor Jaringan Nirkabel. Penelitian ini menggunakan waktu enkripsi dan dekripsi, ukuran kunci, dan konsumsi energi sebagai parameter perbandingan.

Penelitian ini dilakukan dengan melakukan perbandingan hasil ukuran kunci, waktu enkripsi dan dekripsi serta konsumsi energi dari

masing-masing algoritma. Peneliti menggunakan input data dengan besaran 8, 64, dan 256 bit. Hasil dari penelitian ini menunjukan bahwa ECC unggul dalam hal ukuran kunci, waktu dekripsi, dan konsumsi energi sementara waktu enkripsi jauh lebih baik dengan RSA.

Pada penelitian [12] yang berjudul *"Performance Analysis of RSA and Elliptic Curve Cryptography"* peneliti melakukan analisis perbandingan terhadap algoritma kriptografi asimetris yaitu ECC, RSA, dan dua varian RSA yaitu RSA with CRT dan Multi-Prime RSA. Penelitian dilakukan dengan menggunakan ukuran kunci yang direkomendasikan oleh NIST *(National Institute of Standards and Technology)*. Ukuran kunci yang digunakan yaitu 1024, 2048, dan 3072 bit untuk RSA dan 160, 224, 256 untuk ECC.

Hasil dari penelitian ini menunjukan bahwa ECC mengungguli RSA dan semua varian RSA yaitu RSA with CRT dan Multi-Prime RSA dalam hal efisiensi dan keamanan operasional dengan parameter yang lebih rendah. Penelitian ini menerapkan ECC dengan sistem koordinat affine.

Pada penelitian [13] yang berjudul "Analisis Kompleksitas Waktu Algoritma Kriptografi Elgamal Dan *Data Encryption Standard* (DES)*"* yang bertujuan untuk melakukan perbandingan kompleksitas waktu antara algoritma kriptografi Elgamal dan DES.

Metode yang digunakan adalah analisis kompleksitas waktu algoritma dengan simulasi komputer untuk memperoleh data waktu dari setiap algoritma. Data kompleksitas waktu diperoleh dari perhitungan penjumlahan kompleksitas setiap langkah proses enkripsi dan dekripsi dari *pseudocode* setiap algoritma. Program simulasi komputer yang digunakan untuk melakukan proses enkripsi dan dekripsi dibangun menggunakan Bahasa pascal diatas aplikasi Lazarus. Data yang digunakan berupa plainteks dengan Panjang teks tertentu.

Hasil penelitian menunjukan bahwa nilai kompleksitas algoritma ElGamal adalah $O(x^2)$ dengan x adalah kunci privat. Beberapa hal yang mempengaruhi kompleksitas waktu pada algoritma ElGamal adalah

panjang plaintext dan panjang kunci privatnya. Nilai kompleksitas pada algoritma DES adalah O(n) dengan n adalah panjang plaintext. Nilai kompleksitas waktu pada algoritma DES hanya dipengaruhi oleh panjang plaintextnya.

Tabel 2. 1 State of the Art

No	*Title*	*Comparing*	*Contrasting*	*Synthesize*	*Summarize*
1	*Performance Analysis of Cryptographic Algorithms for Selecting Better Utilization on Resource Constraint Devices* (Md. Enamul Haque , SM Zobaed , Muhammad Usama Islam , and Faaiza Mohammad Areef, 2018)	Melakukan evaluasi kinerja dari algoritma kriptografi kunci simetris dan asimetris yang meliputi AES, RC4, Blowfish, CAST, 3DES, dan Twofish sebagai algoritma kunci simetris serta DSA dan ElGamal.	Parameter yang digunakan untuk melakukan perbandingan antara lain ukuran kunci, blok data, tipe data, dan kecepatan enkripsi dan dekripsi.	Implementasi algoritma menggunakan bahasa pemrograman Python dan perangkat lunak MATLAB untuk memproses data. Perpustakaan Python yang digunakan untuk memfasilitasi program antara lain PyCrypto dan Chilkat dengan masukan berupa file teks, suara, dan video.	Hasil pengujian dari penelitian ini menunjukan bahwa algoritma DES, Twofish dan RC4 mengungguli algoritma lainnya dalam hal waktu enkripsi serta dekripsi. Ukuran kunci yang lebih besar memakan banyak waktu komputasi yang mengarah pada penggunaan energi yang lebih tinggi.
2	Perbandingan Performa Kriptografi Asimetris pada Proses *Key Exchange* (Wahid Miftahul Ashari, 2020)	Melakukan pengujian terhadap kecepatan proses pertukaran data dan penggunaan memori (RAM). Penelitian dilakukan untuk mendapatkan algoritma dengan performa terbaik.	Parameter yang digunakan untuk melakukan pengujian adalah kecepatan dan kebutuhan memori (RAM).	Penelitian ini melakukan pengujian terhadap beberapa algoritma asimetris yaitu RSA, El-Gamal, dan Curve25519. Perpustakaan yang digunakan dalam proses pembuatan sistem untuk melakukan pengukuran waktu kinerja adalah modul Python time, dan untuk	Hasil pengujian dari penelitian ini menunjukan bahwa ElGamal memiliki waktu kinerja yang paling cepat dengan kebutuhan memori yang paling efisien. Diikuti RSA dengan waktu kinerja yang kurang stabil jika

No	*Title*	*Comparing*	*Contrasting*	*Synthesize*	*Summarize*
				mengukur konsumsi memori peneliti menggunakan aplikasi Valgrind.	dibandingkan dengan algoritma ElGamal dan ECC. Pada pengujian 256bit RSA dan ElGamal memiliki kinerja yang hampir sama. Algoritma ECC dengan panjang kunci 256bit memiliki tingkat keamanan yang setara dengan algoritma RSA dan ElGamal dengan panjang kunci 2048 bit.
3	*Performance Evaluation of RSA and Elliptic Curve Cryptography in Wireless Sensor Networks* (Amine Kardi, Dr. Rachid Zagrouba, Dr. Mohammed Alqahtani, 2018)	Melakukan analisis perbandingan kinerja algoritma kriptografi RSA dan Kurva Eliptik pada perangkat Sensor Jaringan Nirkabel.	Penelitian ini menggunakan waktu enkripsi dan dekripsi, ukuran kunci, dan konsumsi energi sebagai parameter perbandingan.	Penelitian ini melakukan perbandingan hasil ukuran kunci, waktu enkripsi dan dekripsi serta konsumsi energi dari setiap algoritma. dengan menggunakan input data dengan besaran 8, 64, dan 256 bit.	Hasil dari penelitian ini menunjukan bahwa ECC unggul dalam hal ukuran kunci, waktu dekripsi, dan konsumsi energi sementara waktu enkripsi jauh lebih baik dengan RSA.
4	*Performance Analysis of RSA and Elliptic Curve*	Melakukan analisis terhadap algoritma	Parameter penelitian untuk melakukan	Penelitian dilakukan dengan menggunakan ukuran kunci	Hasil dari penelitian ini menunjukan bahwa ECC

No	*Title*	*Comparing*	*Contrasting*	*Synthesize*	*Summarize*
	Cryptography (Dindayal Mahto, Dilip Kumar Yadav, 2018)	kriptografi asimetris yaitu ECC, RSA, dan dua varian RSA yaitu RSA with CRT dan Multi-Prime RSA.	perbandingan performa adalah waktu enkripsi, waktu dekripsi, serta waktu total dari proses enkripsi dan dekripsi.	yang direkomendasikan oleh NIST *(National Institute of Standards and Technology)*. Ukuran kunci yang digunakan yaitu 1024, 2048, dan 3072bit untuk RSA dan 160, 224, 256 untuk ECC.	mengungguli RSA dan semua varian RSA yaitu RSA with CRT dan Multi-Prime RSA dalam hal efisiensi dan keamanan operasional dengan parameter yang lebih rendah. Penelitian ini menerapkan ECC dengan sistem koordinat *affine*.
5	Analisis Kompleksitas Waktu Algoritma Kriptografi *Elgamal* Dan *Data Encryption Standard* (Herman Kabetta, 2017)	Melakukan perbandingan kompleksitas watu antara algoritma kriptografi Elgamal dan DES.	Penelitian ini menggunakan metode analisis kompleksitas waktu algoritma berdasarkan waktu (kompleksitas waktu).	Data kompleksitas waktu diperoleh dari perhitungan penjumlahan kompleksitas setiap langkah proses enkripsi dan dekripsi dari pseudocode setiap algoritma. Program simulasi computer yang digunakan melakukan proses enkripsi dan dekripsi dibangun menggunakan Bahasa pascal diatas aplikasi Lazarus. Data yang digunakan berupa	Hasil penelitian menunjukan bahwa nilai kompleksitas algoritma ElGamal adalah $O(x^2)$ dengan x adalah kunci privat. Beberapa hal yang mempengaruhi kompleksitas waktu pada algoritma ElGamal adalah panjang plaintext dan panjang kunci privatnya. Nilai kompleksitas pada algoritma DES adalah O(n) dengan n adalah panjang plaintext. Nilai

No	*Title*	*Comparing*	*Contrasting*	*Synthesize*	*Summarize*
				plainteks dengan Panjang teks tertentu.	kompleksitas waktu pada algoritma DES hanya dipengaruhi oleh panjang plaintextnya.

2.2 Dasar Teori

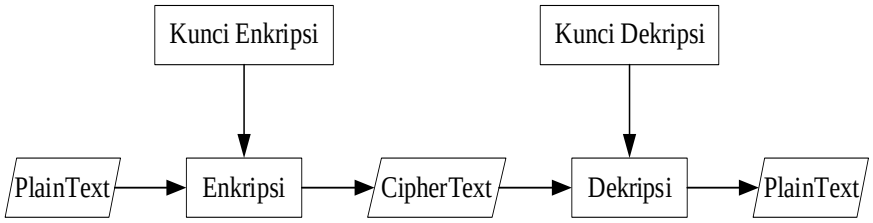
2.2.1 Kriptografi

Kriptografi berasal dari bahasa Yunani, yaitu kata "*Crypto*" dan "*Graphia*" yang berarti penulisan rahasia atau seni mengamankan pesan. Kriptografi merupakan bagian dari cabang ilmu matematika yang disebut kriptologi. Kriptografi dapat diartikan sebagai proses mengubah informasi dari bentuk normalnya, bentuk yang dapat dipahami, menjadi bentuk yang tidak dapat dipahami, sehingga informasi tersebut tidak dapat dibaca tanpa pengetahuan yang rahasia[4]. Definisi yang dikemukakan oleh Bruce Schneier (1996), kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan *(Cryptography is the art and science of keeping messages secure)*[8].

Tujuan mendasar ilmu kriptografi secara lengkap ada empat, yaitu: [14]

a. Kerahasiaan**,** Kerahasiaan adalah layanan yang digunakan untuk menjaga isi dari informasi dari siapa pun kecuali yang memiliki otoritas.
b. Integritas data**,** Integritas data berhubungan dengan penjagaan dari perubahan data secara tidak sah. Untuk menjaga integritas data, sistem harus memiliki kemampuan untuk mendeteksi manipulasi data oleh pihak-pihak yang tidak berhak, antara lain penyisipan, penghapusan, dan pensubsitusian data lain kedalam data yang sebenarnya.
c. Otentikasi**,** Otentikasi berhubungan dengan identifikasi, baik secara kesatuan sistem maupun informasi itu sendiri. Dua pihak yang saling berkomunikasi harus saling memperkenalkan diri. Informasi yang dikirimkan melalui kanal harus diotentikasi keaslian, isi datanya, waktu pengiriman, dan lain-lain.
d. Non-repudiasi**,** Non-repudiasi berarti pengirim tidak dapat menyangkal bahwa ia telah mengirimkan pesan.

Dalam kriptografi terjadi dua proses yaitu proses enkripsi dan proses dekripsi. Enkripsi ialah proses mengamankan suatu informasi dengan membuat informasi tersebut tidak dapat dibaca tanpa bantuan pengetahuan khusus. Dekripsi adalah proses kebalikan dari enkripsi yang bertujuan untuk mengembalikan informasi yang tidak jelas menjadi informasi yang jelas[4]. Proses enkripsi dan dekripsi dapat dilihat dalam Gambar 2.1.



Gambar 2. 1 Proses Enkripsi dan dekripsi

Beberapa istilah yang terdapat dalam proses enkripsi dan dekripsi antara lain:[8]

1. *Plaintext,* Suatu pesan yang tidak disandikan ataupun dapat disebut juga sebagai *cleartext*.
2. *Ciphertext,* pesan yang telah enkripsi.
3. *encryption* atau enkripsi, proses yang dilakukan untuk mengubah *plaintext* ke dalam *ciphertext* atau disebut juga *encipherment*.
4. *Decription* atau dekripsi, proses untuk mengubah *ciphertext* kembali ke *plaintext* atau disebut juga *decipherment*.

Algoritma kriptografi semakin berkembang dan terbagi atas dua bagian yaitu algoritma kriptografi klasik dan modern. Pada kriptografi klasik, kriptografer menggunakan algoritma sederhana, yang memungkinkan ciphertext dapat dipecahkan dengan mudah (melalui penggunaan statistik, terkaan, intuisi, dan sebagainya). Algoritma kriptografi modern dibuat sedemikian kompleks sehingga kriptanalis sangat sulit untuk memecahkan ciphertext tanpa mengetahui kunci. Pengelompokan algoritma juga dilakukan berdasarkan kunci enkripsi – dekripsi yang digunakan, yaitu

symmetric cryptosystem atau simetris (menggunakan kunci yang sama untuk proses enkripsi – dekripsi) dan *Assymmetric cryptosystem* atau asimetris (menggunakan kunci yang berbeda untuk proses enkripsi – dekripsi)[8].

Kriptografi kunci-simetris mengarah kepada metode enkripsi yang mana baik pengirim maupun yang dikirim saling memiliki kunci yang sama(walaupun kebanyakan kunci yang ada sedikit berbeda namun masih berhubungan dalam hal kemudahan perhitungan)[5]. Kriptografi kunci-simetris kadang disebut sebagai Kriptografi simetris merupakan bentuk kriptografi yang lebih tradisional, dimana sebuah kunci tunggal dapat digunakan untuk mengenkripsi dan mendekripsi pesan[5].

Algoritma kriptografi simetris dibagi menjadi dua kategori yaitu algoritma aliran *(Stream Ciphers)* dan algoritma blok *(Block Ciphers)*. Dimana pada algoritma *Stream Ciphers* proses penyandiannya akan beriorientasi pada satu bit/*byte* data. Sedangkan pada algoritma *Block Ciphers*, proses penyandiannya berorientasi pada sekumpulan bit/*byte* data (per blok). Keamanan dari algoritma ini terletak pada kuncinya, jika kunci diberitahukan atau dibocorkan maka siapa saja dapat mengenkrip dan mendekrip data, jadi kunci harus benar-benar rahasia dan aman[6]. Beberapa contoh algoritma kriptografi simetris antara lain Cipher Permutasi, Cipher Substitusi, Cipher Hill, OTP, RC6, Twofish, Magenta, FEAL, SAFER, LOKI, CAST, Rijndael (AES), Blowfish, GOST, A5, Kasumi, DES dan IDEA.

Sistem sandi asimetris atau dikenal juga sebagai sistem sandi kunci publik adalah sistem sandi yang metode menyandi dan membuka sandinya menggunakan kunci yang berbeda. Tidak seperti sistem sandi simetris, sistem sandi ini relatif masih baru[5]. Algoritma sandi jenis ini yang telah terkenal diantaranya Rivest Shamir Adleman (RSA), Knapsack, Rabin, ElGamal, *Digital*

Signature Algorithm (DSA), GOST, Diffie-Hellman, *Elliptic Curve Cryptography* (ECC).

Sistem ini memiliki sepasang kunci yang disebut kunci publik yaitu kunci yang didistribusikan secara umum dan kunci privat yaitu kunci yang dirahasiakan yang hanya dimiliki oleh pihak yang berhak. Umumnya kunci publik digunakan untuk menyandi dan kunci privat digunakan untuk membuka sandi. Sistem sandi asimetris bekerja lebih lambat dari sistem sandi simetris, sehingga sistem sandi ini lebih sering digunakan untuk menyandi data dengan ukuran bit yang kecil. Sistem sandi ini sering pula digunakan untuk mendistribusikan kunci sistem sandi simetris. Penggunaan lain sistem sandi asimetris adalah dalam tanda tangan digital. Tanda tangan digital seperti halnya tanda tangan biasa digunakan untuk membuktikan keaslian dari suatu dokumen yang dikirimkan. Kunci privat digunakan untuk menandatangani, sedangkan kunci publik digunakan untuk membuktikan keaslian tanda tangan itu[5].

Secara umum algoritma kunci asimetris dibagi menjadi tiga, yaitu :[14]

1. Algoritma yang didasarkan pada persoalan faktorisasi integer RSA, algoritma kunci asimetris yang paling banyak digunakan, berdasarkan pada tingkat kesulitan dalam memecahkan persoalan ini. RSA menggunakan perkalian integer untuk *forward operation,* dan faktorisasi sebagai *inverse operation.*
2. Algoritma yang didasarkan pada persoalan algoritma diskrit Protokol pertukaran kunci *Diffie-Hellman* didasarkan pada persoalan ini, begitu juga protokol pertukaran kunci lainnya, seperti *Digital Signature Algorithm* (*DSA*). *Diffie Hellman* menggunakan eksponensiasi diskrit sebagai *forward operation* dan *log* sebagai *inverse operation*.
3. Algoritma yang didasarkan pada persoalan logaritma diskrit kurva eliptik. *Elliptic Curve Cryptography (ECC*) adalah

algoritma kunci asimetris terkini yang semakin banyak digunakan. ECC menggunakan penggandaan poin (*point multiplication*) sebagai *forward operation* dan persoalan logaritma diskrit (*discrete logarithm problem)* sebagai *inverse operation*.

2.2.2 Algoritma Kriptografi RSA *(*Rivest-*Shamir-Adleman)*

Algoritma RSA adalah metode kriptografi yang Ditemukan oleh tiga peneliti dari MIT *(Massachussets Institute of Technology)*, yaitu Ron Rivest, Adi Shamir, dan Len Adleman pada tahun 1977[15]. Kekuatan algoritma RSA terletak pada sulitnya memfaktorkan bilangan besar menjadi faktor-faktor bilangan primanya, sehingga semakin besar bilangan prima yang digunakan semakin baik atau aman. Dalam kriptografi menggunakan algoritma RSA terdapat tiga proses yaitu proses pembangkitan kunci publik dan kunci privat, proses enkripsi, dan proses dekripsi[16].

Pada tahun 1977, Ronald L. Rivest, Adi Shamir, dan Leonard M. Adleman merumuskan algoritma praktis yang mengimplementasikan sistem kriptografi kunci publik yang disebut dengan sistem kriptografi RSA. Sepasang kunci yang dipakai pada kedua proses ini adalah kunci publik (e,n) sebagai kunci enkripsi dan kunci privat d sebagai sebagai kunci dekripsi dimana e, d dan n adalah bilangan bulat positif. Algoritma RSA adalah sebuah block cipher algorithm (algoritma yang bekerja per blok data) yang mengelompokkan plaintext menjadi blok-blok terlebih dahulu sebelum dilakukan enkripsi hingga menjadi ciphertext[16].

Sebagai salah satu skema enkripsi kunci publik pertama yang digunakan secara luas, RSA meletakkan dasar bagi sebagian besar komunikasi yang aman. RSA secara tradisional digunakan dalam *Transport Layer Security* (TLS) dan juga merupakan algoritma asli yang digunakan dalam enkripsi *Pretty Good Privacy* (PGP) . RSA

masih terlihat di berbagai browser web, email, *Virtual Private Network* (VPN), obrolan, dan saluran komunikasi lainnya. RSA juga sering digunakan untuk membuat koneksi aman antara klien VPN dan server VPN. Di bawah protokol seperti OpenVPN , TLS *handshake* dapat menggunakan algoritma RSA untuk bertukar kunci dan membuat saluran yang aman[17].

Pada algoritma RSA terdapat 3 langkah utama yaitu *key generation* (pembangkit kunci), enkripsi dan dekripsi. Kunci pada RSA mencakup dua buah kunci, yaitu *public key* dan *private key*. *Public key* digunakan untuk melakukan enkripsi, dan dapat diketahui oleh orang lain. Sedangkan *private key* tetap dirahasiakan dan digunakan untuk melakukan dekripsi[16].

Keamanan algoritma RSA terletak pada tingkat kesulitan dalam memfaktorkan bilangan non prima menjadi faktor primanya, yang dalam hal ini $n = p \times q$. Sekali n berhasil difaktorkan menjadi p dan q, maka $\phi(n) = (p - 1)(q - 1)$ dapat dihitung. Selanjutnya, karena kunci enkripsi PK diumumkan (tidak rahasia), maka kunci dekripsi SK dapat dihitung dari persamaan $PK \cdot SK \equiv 1 \pmod{\phi(n)}$. Penemu algoritma RSA menyarankan nilai p dan q panjangnya lebih dari 100 digit. Dengan demikian hasil kali $n = p \times q$ akan berukuran lebih dari 200 digit[18].

Menurut Rivest dan kawan-kawan, usaha untuk mencari faktor bilangan 200 digit membutuhkan waktu komputasi selama 4 milyar tahun (dengan asumsi bahwa algoritma pemfaktoran yang digunakan adalah algoritma yang tercepat saat ini dan komputer yang dipakai mempunyai kecepatan 1 milidetik). Belum ditemukannya algoritma yang paling mangkus untuk memfaktorkan bilangan yang besar membuat algoritma RSA tetap dipakai hingga saat ini, sehingga algoritma RSA tetap direkomendasikan untuk menyandikan pesan[18].

Sebagai algoritma Kriptografi Asimetris, pengkodean RSA membutuhkan dua kunci yang berbeda untuk enkripsi dan dekripsi. Bilangan yang dipilih sebagai kunci adalah bilangan prima yang besar, dengan alasan pemfaktoran sebuah bilangan hasil perkalian dari dua bilangan prima yang besar menjadi dua bilangan prima yang sesuai akan sangat sulit, sehingga keamanan dari RSA dapat terjamin. Berikut langkah-langkah proses pembangkitan pasangan kunci pada RSA[18][15]:

1. Pilih dua buah bilangan prima sembarang, p dan q.
2. Hitung $n = p \cdot q$. Sebaiknya $p \neq q$, sebab jika $p = q$ maka $r = p$ sehingga p dapat diperoleh dengan menarik akar pangkat dua dari r.
3. Hitung $\phi(n) = (p - 1)(q - 1)$.
4. Pilih kunci publik, PK, yang relatif prima terhadap $\phi(n)$.
5. Bangkitkan kunci rahasia dengan menggunakan $SK \cdot PK \equiv 1 \pmod{\phi(n)}$.

Proses enkripsi dengan algoritma RSA dilakukan dengan menghitung exponen plaintext dalam operasi *modulo n* (modulo = sisa pembagian) untuk setiap blok pesan atau data sehingga dapat menghasilkan ciphertext. Eksponen yang digunakan adalah *public exponent e*. Operasi ini bisa dituliskan dengan persamaan berikut[15] :

Plaintext: $M < n$

Ciphertext: $C = M^e \bmod n$

Proses deskripsi algoritma RSA dilakukan hampir sama dengan enkripsi tapi eksponen yang digunakan adalah *private exponent d* untuk mengembalikan pesan seperti semula. Operasi ini bisa dituliskan dengan persamaan berikut [15]:

Ciphertext: C

Plaintext: $M = C^d \bmod n$

2.2.3 Algoritma Kriptografi ECC (*Elliptic Curve Cryptography*)

Elliptic Curve Cryptography (ECC) dikembangkan secara independen oleh Victor Miller pada tahun 1986 dan oleh Neil Koblitz pada tahun 1987. Sejak saat itu, ECC telah dievaluasi oleh para pakar matematika dan komputer di seluruh dunia, dan belakangan ini digunakan secara komersial, misalnya untuk keamanan email dan keamanan web[19].

Dibandingkan dengan kriptografi kunci asimetris lainnya, ECC dianggap lebih unggul. Penyebab utamanya adalah karena dengan menggunakan kunci yang jauh lebih kecil atau pendek, ECC tetap dapat memberikan tingkat keamanan yang sama dengan algoritma asimetris lainnya yang menggunakan kunci yang lebih besar. Dengan ukuran kunci yang lebih kecil dan tingkat keamanan yang sama tinggi, implementasi ECC menjadi lebih efisien, yang berarti memungkinkan operasi kriptografi menjadi lebih cepat, dengan prosesor yang lebih rendah atau *software* yang lebih simpel. Ini berarti,tingkat panas yang dihasilkan dan konsumsi tenaga untuk memproses ECC menjadi lebih rendah[19].

Penggunaan ECC di sektor publik dan swasta telah meningkat selama beberapa tahun terakhir. Sementara RSA terus digunakan secara lebih luas dan lebih mudah dipahami dibandingkan dengan ECC, manfaat efisiensi ECC membuatnya menarik untuk banyak kasus penggunaan perusahaan. Ini termasuk mempercepat akses aman ke situs web terenkripsi *Secure Sockets Layer* (SSL) dan *streaming* data terenkripsi dari perangkat *Internet of Things (*IoT) dengan daya komputasi terbatas[20].

Operasi matematika pada ECC merupakan fungsi kurva eliptik $y^2 = x^3 + ax + b$, di mana $4a^3 + 27b^2 \neq 0$. Setiap nilai a dan b yang berbeda dapat menghasilkan bentuk kurva eliptik yang berbeda, dan semua nilai x dan y (titik kordinat) yang memenuhi persamaan di atas akan memotong kurva eliptik tersebut. Kunci publik pada

algoritma ECC adalah sebuah titik pada kurva eliptik dan kunci privatnya adalah sebuah angka acak. Kunci publik diperoleh dengan melakukan operasi perkalian terhadap kunci privat dengan titik generator G pada kurva eliptik.

Keamanan ECC bergantung pada masalah logaritma diskrit yang sulit untuk dipecahkan. Misalnya P dan Q adalah dua titik pada suatu kurva eliptik, sedemikian rupa sehingga kP = Q, di mana k adalah sebuah nilai skalar. Dengan mengetahui nilai P dan Q, tidak mungkin bisa mendapatkan nilai k, jika k adalah nilai yang sangat besar. k adalah logaritma diskrit basis P terhadap Q. Oleh karena itu, operasi utama pada ECC adalah perkalian titik seperti contoh di atas. Pada bagian-bagian selanjutnya akan dijelaskan tiga operasi yang dilakukan pada implementasi algoritma ECC[19].

Dalam kriptografi kunci asimetris, harus ditentukan terlebih dahulu nilai parameter yang akan digunakan dan telah disepakati oleh pihak yang akan berkomunikasi. Parameter yang digunakan dalam ECC yaitu nilai a dan b, bilangan prima p dalam persamaan kurva eliptik bidang terbatas serta titik generator G yang dipilih dari kurva eliptik. Pendekatan enkripsi dengan ECC ini dapat dijelaskan seperti berikut[21]:

1. Pembangkitan Kunci Privat dan Kunci Publik

   Pembangkitan kunci privat n_B dilakukan dengan cara memilih bilangan acak yang nilainya diantara [1, p-1]. Dengan kunci privat tersebut, bangkitkan kunci publik P_B=n_B.G.

2. Enkripsi

   Misalnya pesan yang akan dikirim adalah pesan *m*. Langkah pertama adalah melakukan encode pesan *m* menjadi sebuah titik P_m, dari kurva eliptik. Lalu memilih bilangan acak *k* yang nilai diantara [1, p-1]. Cipherteks akan dihasilkan dari persamaan C_m={(kG), (P_m+kP_B)}.

   dimana G adalah titik generator dan P_B adalah kunci publik.

3. Deskripsi

   Untuk melakukan deskripsi cipherteks C_m, Langkah pertama adalah mengalikan ttik pertama dari cipherteks dengan kunci privat n_B dan kemudian mengurangkan titik kedua dari cipherteks dengan hasil perkalian tersebut.

   P_m+knP_B-n_B(kG)= P_m+k(nB_G)- n_B(kG)= P_m

   Lalu decode P_m menjadi pesan m semula[21].

2.2.4 Algoritma Kriptografi ElGamal

Algoritma Elgamal diciptakan oleh ilmuan mesir, yaitu Taher ElGamal pada tahun 1984. Algoritma ElGamal merupakan algoritma dalam kriptografi yang termasuk dalam kategori algoritma asimetris. Keamanan algoritma ElGamal terletak pada kesulitan perhitungan logaritma diskret pada bilangan modulo prima yang besar sehingga upaya untuk menyelesaikan masalah logaritma ini menjadi sangat sukar. Algoritma ElGamal digunakan dalam perangkat lunak Penjaga Privasi GNU gratis, versi terbaru PGP, dan sistem kripto lainnya[22].

Algoritma ElGamal mempunyai kunci publik yang berupa tiga pasang bilangan dan kunci rahasia berupa satu bilangan. Algoritma ini mempunyai kerugian pada cipherteksnya yang mempunyai panjang dua kali lipat dari plainteksnya. Akan tetapi, algoritma ini mempunyai kelebihan pada enkripsi. Untuk plainteks yang sama, algoritma ini memberikan cipherteks yang berbeda (dengan kepastian yang dekat) setiap kali plainteks di enkripsi[22].

Algoritma ElGamal terdiri dari tiga proses, yaitu proses pembentukan kunci, proses enkripsi dan proses dekripsi. Sama halnya dengan algoritma RSA, algoritma ElGamal juga merupakan cipher blok, yaitu melakukan proses enkripsi pada blok-blok plainteks dan menghasilkan blok-blok cipherteks yang kemudian dilakukan proses dekripsi dan hasilnya digabungkan[23].

Besaran-besaran yang digunakan dalam pembangkitan kunci publik algoritma ElGamal[23] :

1. Bilangan prima, p (tidak rahasia) β.
2. Bilangan acak α sebagai akar primitif (α < p) (tidak rahasia).
3. Bilangan acak a (a < p) (rahasia).
4. Blok plainteks M (plainteks) (rahasia).
5. y dan c (cipherteks) (tidak rahasia).

Pembentukan kunci pada algoritma ElGamal terdiri atas pembentukan kunci publik dan kunci rahasia. Pada proses ini dibutuhkan sebuah bilangan prima (p) yang digunakan untuk membentuk grup Zp*, elemen primitif α dan sembarang a { 0,1,…., p-2}. Kunci publik algoritma ElGamal terdiri atas pasangan 3 bilangan (p, α, β) di mana[23]:

$$\beta = \alpha^a \bmod p$$

Sedangkan kunci rahasianya adalah bilangan a tersebut. Proses pembentukan kunci untuk algoritma ElGamal terdiri atas[23] :

1. Penentuan bilangan prima aman yang bernilai besar.
2. Penentuan elemen primitif.
3. Pembentukan kunci berdasarkan bilangan prima aman dan elemen primitif.

Tujuan penentuan bilangan prima aman ini adalah untuk mempermudah dalam penentuan elemen primitif. Digunakan bilangan prima p sehingga:

$$p = 2.\ q + 1$$

dengan q adalah bilangan prima sehingga nilai minimal p adalah 5 dan q adalah 2. Bilangan prima p tersebut disebut sebagai bilangan prima aman. Langkah penentuan bilangan prima tersebut dinyatakan sebagai berikut[23]:

1. Tentukan bilangan prima p ≥ 5.
2. Hitung q dengan persamaan p = 2.q +1.

3. Jika q merupakan bilangan prima, maka p merupakan bilangan prima aman. Jika q bukan merupakan bilangan prima, maka p bukan merupakan bilangan prima aman.

Untuk menguji keprimaan suatu bilangan, digunakan suatu metode yang disebut Teorema Fermat yaitu jika x adalah bilangan prima dan y adalah bilangan bulat yang tidak habis dibagi dengan x, yaitu PBB(y,x) = 1, maka[23]:

$y^{x-1} = 1 \pmod{p}$

Proses enkripsi algoritma ElGamal menggunakan kunci publik (p,α,β) dan sebuah bilangan integer acak k (k ϵ {0,1,…,p-1}) yang dijaga kerahasiannya oleh penerima yang mengenkripsi pesan. Untuk setiap karakter dalam pesan dienkripsi dengan menggunakan bilangan k yang berbeda-beda. Satu karakter yang direpresentasikan dengan menggunakan bilangan bulat ASCII akan menghasilkan kode dalam bentuk blok yang terdiri atas dua nilai (r, t)[23].

Langkah-langkah dalam proses enkripsi, adalah sebagai berikut[23]:

1. Ambil sebuah karakter dalam pesan yang akan dienkripsi dan ditransformasi karakter tersebut ke dalam kode ASCII sehingga diperoleh bilangan bulat M.
2. Hitung nilai r dan t dengan persamaan berikut:
   $r = \alpha^k \pmod{p}$
   $t = \beta^k M \pmod{p}$
3. Diperoleh cipherteks untuk karakter M tersebut dalam blok (r, t).
4. Lakukan proses di atas untuk seluruh karakter dalam pesan termasuk karakter spasi.

Dekripsi algoritma ElGamal dari cipherteks ke plainteks menggunakan kunci rahasia a yang disimpan kerahasiaannya oleh penerima pesan. Diberikan (p, α, β) sebagai kunci publik dan a sebagai kunci rahasia pada algoritma ElGamal. Jika diberikan cipherteks (r, t), maka:

$M = t\,(r^a)^{-1} \bmod p$

Dengan M adalah plainteks. Di mana nila: $(r^a)^{-1} = r^{-a} = r^{p-1-a} \bmod p$.

Langkah-langkah proses dekripsi algoritma ElGamal, adalah sebagai berikut[23]:

1. Ambil sebuah blok cipherteks dari pesan yang telah dienkripsikan pengirim.
2. Dengan menggunakan a yang dirahasiakan oleh penerima, hitung nilai plainteks dengan menggunakan persamaan
   $M = t\,(r^a)^{-1} \bmod p$ dan
   $(r^a)^{-1} = r^{-a} = r^{p-1-a} \bmod p$.

2.2.5 Tandatangan Digital

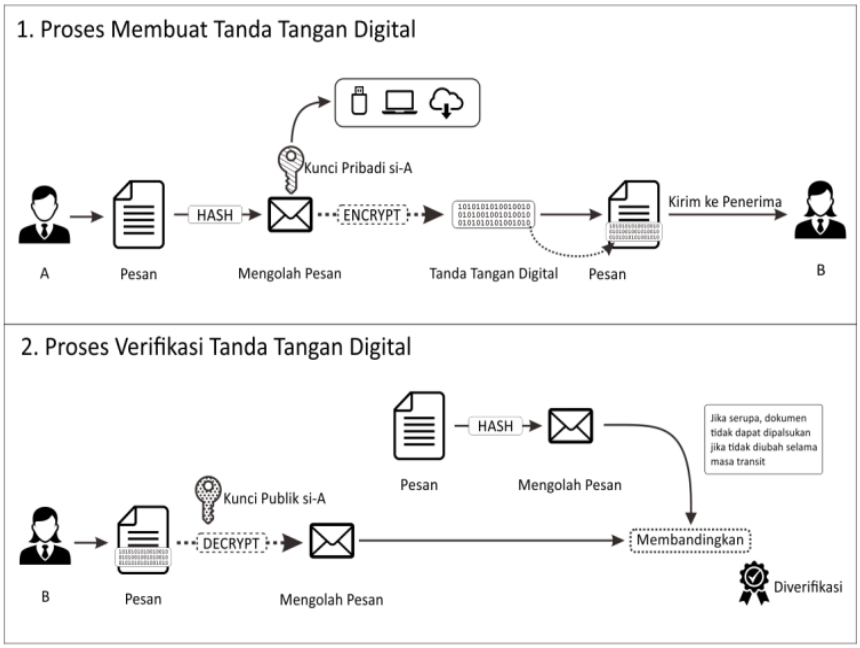
Konsep tandatangan digital (TTD) sendiri sudah ada sejak dari awal perkembangan teknologi informasi dan komunikasi. Ide awal TTD sendiri dimulai sejak 1976 melalui sebuah tulisan *New Direction in Cryptography.* Dalam artikel tersebut menerangkah bahwa konfidensial sebuah dokumen yang dikirimkan dalam bentuk digital adalah sebuah tantangan. Pada prinsipnya, TTD adalah sebuah kombinasi uni dari fungsi hash dan enkripsi dengan metode asimetris (Schneier, 1995), Untuk dapat menandatangani sebuah dokumen elektronik, dokumen tersebut akan dijadikan sebagai masukan pada fungsi hash[24].

Teknik umum yang digunakan untuk membuat TTD adalah dengan fungsi M, ditransformasikan oleh fungsi hash (H), menjadi pesan singkat (h). Pesan singkat (h) tersebut lalu di enkripsi (S) dengan Kunci Pribadi (*Private Key* / PK) pengirim pesan (Munir, 2005). Teknik tersebut dapat dirumuskan sebagai berikut[24]:

$S = E_{SK}\,(h)$

Hasil dari enkripsi (S) tersebut yang disebut dengan TTD. Oleh karena itu, setiap perbedaan pada satu bit pada konten dokumen yang

dihasilkan memiliki nilai hash yang berbeda juga. Proses tandatangan digital dapat dilihat pada gambar dibawah ini.



Gambar 2. 2 Proses Pembuatan Tandatangan Digital[24]

2.2.6 Kompleksitas Algoritma

Setiap komputer dengan arsitektur berbeda mempunyai bahasa mesin yang berbeda sehingga waktu setiap operasi antara satu komputer dengan komputer lain tidak sama. *Compiler* bahasa pemrograman yang berbeda menghasilkan kode mesin yang berbeda sehingga waktu setiap operasi antara *compiler* dengan *compiler* lain tidak sama. Model abstrak pengukuran waktu/ruang harus independen dari pertimbangan mesin dan *compiler* apapun. Besaran yang dipakai untuk menerangkan model abstrak pengukuran waktu/ruang ini adalah kompleksitas algoritma. Ada dua macam kompleksitas algoritma, yaitu: kompleksitas waktu dan kompleksitas ruang[25].

Kompleksitas waktu, T(n), diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n. Kompleksitas ruang, S(n), diukur dari memori yang digunakan oleh struktur data yang terdapat di dalam

algoritma sebagai fungsi dari ukuran masukan n. laju peningkatan waktu (ruang) yang diperlukan algoritma dapat ditentukan menggunakan besaran kompleksitas waktu/ruang algoritma dengan meningkatnya ukuran masukan n[25].

Hal-hal yang mempengaruhi kompleksitas waktu ialah sebagai berikut [25]:

1. Jumlah masukan data untuk suatu algoritma (n).
2. Waktu yang dibutuhkan untuk menjalankan algoritma tersebut.
3. Ruang memori yang dibutuhkan untuk menjalankan algoritma yang berkaitan dengan struktur data dari program.

Kompleksitas mempengaruhi performa dan kinerja dari suatu algoritma. Kompleksitas waktu dibagi menjadi tiga jenis, yaitu *worst case*, *best case*, dan *average case*. Masing-masing jenis kompleksitas ini menunjukkan kecepatan atau waktu yang dibutuhkan algoritma untuk mengeksekusi sejumlah kode[25].

1. $T_{max}(n)$: kompleksitas waktu untuk kasus terburuk *(worst case)*, kebutuhan waktu maksimum.
2. $T_{min}(n)$: kompleksitas waktu untuk kasus terbaik *(best case)*, kebutuhan waktu minimum.
3. $T_{avg}(n)$: kompleksitas waktu untuk kasus rata-rata *(average case)*, kebutuhan waktu secara rata-rata.

Pengukuran kinerja kualitatif algoritma biasanya dilakukan dengan menyatakan kinerja sebagai salah satu persamaan sederhana yang menunjukkan hubungan antara input dan kinerja. Cara tradisional untuk menyatakan kinerja algoritma adalah dengan menggunakan notasi *Big-O (O)*[25].

Notasi “O” disebut notasi “O-Besar” (*Big-O)* yang merupakan notasi kompleksitas waktu asimptotik. Notasi *Big-O* berguna untuk membandingkan beberapa algoritma dari persoalan yang sama untuk menentukan yang terbaik. Berikut merupakan Pengelompokan Algoritma Berdasarkan Notasi O-Besar[25].

Tabel 2. 1 Pengelompokan Algoritma Berdasarkan Notasi O-Besar[18]

Kelompok Algoritma	**Nama**	**Spektrum Kompleksitas Waktu**
O(1)	konstan	algoritma polinomial
O(log n)	logaritmik	
O(n)	lanjar	
O(n log n)	n log n	
$O(n^2)$	Kuadratik	
$O(n^3)$	Kubik	
$O(2^n)$	Eksponensial	algoritma eksponensial
O(n!)	Factorial	

Penjelasan masing-masing kelompok algoritma adalah sebagai berikut[25]:

1. O(1), Kompleksitas O(1) berarti waktu pelaksanaan algoritma adalah tetap, tidak bergantung pada ukuran masukan.
2. O(*log n*) Kompleksitas waktu logaritmik berarti laju pertumbuhan waktunya berjalan lebih lambat daripada pertumbuhan *n*. Di sini basis algoritma tidak terlalu penting sebab bila *n* dinaikkan dua kali semula, misalnya, *log n* meningkat sebesar sejumlah tetapan.
3. O(*n*) Algoritma yang waktu pelaksanaannya lanjar umumnya terdapat pada kasus yang setiap elemen masukannya dikenai proses yang sama,. Bila *n* dijadikan dua kali semula, maka waktu pelaksanaan algoritma juga dua kali semula.
4. O(*n log n*) Waktu pelaksanaan yang *n log n* terdapat pada algoritma yang memecahkan persoalan menjadi beberapa persoalan yang lebih kecil, menyelesaikan tiap persoalan secara independen, dan menggabung solusi masing- masing persoalan algoritma.

5. O(n^2) Algoritma yang waktu pelaksanaannya kuadratik hanya praktis digunakan untuk persoalana yang berukuran kecil. Umumnya. Bila *n* dinaikkan menjadi dua kali semula, maka waktu pelaksanaan algoritma meningkat menjadi empat kali semula.
6. O(n^3) Seperti halnya algoritma kuadratik, algoritma kubik memproses setiap masukan dalam tiga buah kalang bersarang, misalnya algoritma perkalian matriks. Bila *n* dinaikkan menjadi dua kali semula, waktu pelaksanan algoritma meningkat menjadi delapan kali semula.
7. O(2^n) Algoritma yang tergolong kelompok ini mencari solusi persoalan secara "*brute force*". Bila *n* dijadikan dua kali semula, waktu pelaksanaan menjadi kuadrat kali semula.
8. O(*n!*) Seperti halnya pada algoritma eksponensial, algoritma jenis ini memproses setiap masukan dan menghubungkannya dengan *n* - 1 masukan lainnya. Bila *n* dijadikan dua kali semula, maka waktu pelaksanaan algoritma menjadi faktorial dari 2*n*.

2.2.7 Code::Blocks

Code::Blocks adalah suatu program *free, non-profit, open-source* dan *cross-platform integrated development environment*. Program ditulis dalam C++ beserta wxWidgets untuk GUI-nya yang dapat digunakan bersama dengan berbagai macam kompilator, seperti GCC dan Visual C++. Sekarang ini, Code::Blocks tersedia sebagai perangkat pengembangan dalam bahasa C dan C++, walaupun program ini juga bisa disesuaikan, dan mungkin akan membutuhkan pemasangan tambahan, untuk pengembangan perangkat lunak ARM, AVR, DirectX, FLTK, Fortran, GLFW, GLUT, GTK+, Irrlicht, Lightfeather, MATLAB, OGRE, OpenGL, Qt, SDL, SFML, STL, SmartWin dan wx. Code::Blocks tersedia di sistem operasi Windows, Linux, Mac OS X dan FreeBSD[26].

2.2.8 NTL *(Number Theory* Library*)*

NTL *(Number Theory Library)* adalah pustaka C++ portabel berperforma tinggi yang menyediakan struktur data dan algoritma untuk memanipulasi bilangan bulat panjang bertanda tangan, dan untuk vektor, matriks, dan polinomial di atas bilangan bulat dan di atas bidang terbatas. NTL adalah perangkat lunak bebas , dan dapat digunakan sesuai dengan ketentuan *GNU Lesser General Public License* versi 2.1 atau yang lebih baru. NTL ditulis dan dikelola oleh Victor Shoup dengan beberapa kontribusi yang dibuat oleh orang lain[27].

NTL menyediakan implementasi algoritma canggih berkualitas tinggi untuk[27]:

1. Aritmatika bilangan bulat panjang sembarang dan aritmatika titik mengambang presisi sembarang.
2. Aritmatika polinomial atas bilangan bulat dan bidang hingga termasuk aritmatika dasar, faktorisasi polinomial, pengujian ireduksi, perhitungan polinomial minimal, jejak, norma, dan banyak lagi.
3. Pengurangan basis kisi, termasuk implementasi Schnorr-Euchner yang sangat kuat dan cepat, pengurangan blok Korkin-Zolotarev, dan heuristik pemangkasan Schnorr-Horner baru untuk blok Korkin-Zolotarev.
4. Aljabar linier dasar atas bilangan bulat, bidang terbatas, dan bilangan titik mengambang presisi sembarang.