BAB III
METODOLOGI PENELITIAN

Pada penelitian ini, dilakukan pengembangan model kriptografi menggunakan dua algoritma yang kemudian diimplementasikan menggunakan bahasa pemrograman *Python* dengan GUI. Langkah-langkah diuraikan secara rinci sebagai berikut:

3.1 Identifikasi Masalah

Pertukaran data atau informasi merupakan hal yang umum dilakukan pada era digital. Pihak pengirim akan mengirimkan sebuah data kepada penerima dengan maksud agar penerima memperoleh informasi yang diberikan oleh pengirim. Namun, pada proses pengiriman informasi melalui *server* publik, data rentan untuk dimanfaatkan oleh pihak lain ketika data yang dikirim merupakan data asli yang dapat langsung dipahami isi pesannya. Sehingga keamanan data merupakan hal yang sangat penting dalam pengiriman dan pertukaran data.

Salah satu data yang penting untuk diamankan ialah *coding file*. Sebuah *coding file* yang akan dikirim perlu diberikan keamanan agar hanya pihak yang memiliki kewenangan saja yang dapat memakai *coding file* tersebut. Salah satu nya dengan mengaplikasikan ilmu kriptografi pada data yang akan diamankan sehingga, pihak lain yang tidak memiliki kewenangan tidak dapat memperoleh informasi pada data tersebut tanpa menggunakan kunci.

Terdapat berbagai jenis algoritma kriptografi yang dapat digunakan untuk mengamankan *coding file*. Berdasarkan literatur - literatur yang ada, diambil algoritma OTP CLCG dan RSA-CRT untuk mengamankan *file* yang akan dikirim. Materi yang digunakan untuk mengkonstruksi sistem keamanan pesan ini yaitu berupa dasar - dasar ilmu matematika yang dipakai dalam algoritma kriptografi serta tahapan pada algoritma OTP CLCG dan RSA-CRT.

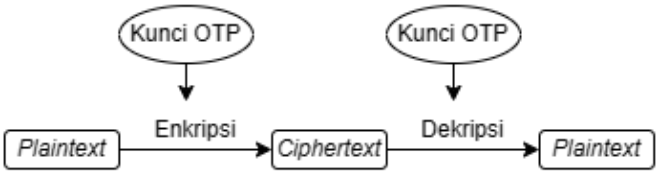
3.2 Model Dasar

Model dasar yang digunakan pada penelitian ini adalah algoritma *One Time Pad* (OTP) dan Rivest Shamir Adleman (RSA).

3.2.1 Algoritma OTP

Algoritma OTP merupakan algoritma kriptpografi simetris yang menggunakan operasi *Exclusive OR* dalam bentuk modulo untuk proses enkripsi

dan dekripsi. Prinsip pengamanan data menggunakan algortima ini adalah dengan mengkombinasikan masing-masing karakter pada *plaintext* dengan satu karakter pada kunci. Berdasarkan penjelasan proses enkripsi dan dekripsi RSA pada bab 2, skema kriptografi *One Time Pad* ditunjukkan pada Gambar 3.1.

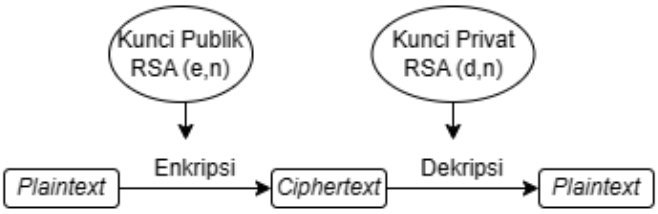


Gambar 3.1 Skema Kriptografi OTP

Proses enkripsi dan dekripsi data pada *coding file* dilakukan sama seperti pada pengamanan teks biasa. Isi data akan dibaca per-satuan karakter lalu dienkripsi maupun dekripsi menggunakan kunci yang ada melalui tahapan yang telah dijelaskan pada sub bab 2.2.4.

3.2.2 Algoritma RSA

RSA (Rivest Shamir Adleman) adalah sebuah algoritma kriptografi asimetris yang mendasarkan proses enkripsi dan dekripsinya pada konsep bilangan prima dan aritmatika modulo. Kunci publik dipakai untuk mengenkripsi sedangkan kunci privat digunakan untuk mendekripsi. Kunci publik RSA dibangkitkan dengan mengambil dua bilangan prima. Dengan menggunakan rumus pembangkitan kunci RSA akan dihasilkan kunci publik (*e*,*n*) dan kunci privat (*d*,*n*) dicari menggunakan rumus $(d \times e)$ mod $\phi(n) = 1$ atau $d = \frac{1 + \mathrm{k} \times \phi(n)}{e}$ di mana $d \in \mathbb{Z}$. Berdasarkan penjelasan proses enkripsi dan dekripsi RSA pada bab 2, skema kriptografi RSA ditunjukkan pada Gambar 3.2.

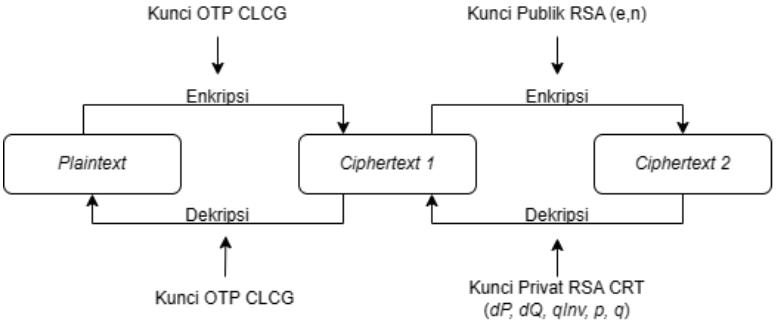


Gambar 3.2 Skema Kriptografi RSA

Skema kriptografi RSA diawali dengan membaca *coding file* perkarakter sebagai *plaintext* dan dilakukan proses enkripsi dan dekripsi sesuai dengan tahapan di sub bab 2.2.5.

3.3 Pengembangan Model

Penelitian ini menggunakan dua algoritma yang dimodifikasi yaitu algoritma OTP CLCG dengan RSA-CRT sehingga perlu adanya perancangan skema enkripsi dan dekripsi. Skema proses enkripsi dekripsi *file* ditunjukkan pada Gambar 3.3.



Gambar 3.3 Skema enkripsi dan dekripsi model pengembangan

Skema enkripsi diawali dengan membaca *file* yang akan dienkripsi. Lalu *file* yang berupa *plaintext* dienkripsi menggunakan algoritma OTP modifikasi CLCG menghasilkan *ciphertext* pertama. Selanjutnya, *ciphertext* pertama dienkripsi menggunakan kunci publik algoritma RSA sehingga menghasilkan *ciphertext* kedua. *Ciphertext* kedua merupakan *file* terenkripsi yang akan dikirim ke penerima. Ketika penerima ingin mengembalikan *file* enkripsi menjadi *file* asli, maka dilakukan proses dekripsi. Dekripsi diawali dengan membaca *file* yang telah terenkripsi. Lalu *file* yang berupa *ciphertext* didekripsi menggunakan kunci privat algoritma RSA-CRT menghasilkan *cipherteks* 1. Selanjutnya, *cipherteks* 1 didekripsi menggunakan kunci OTP modifikasi CLCG sehingga menghasilkan *plaintext* yang sama dengan *plainteks* awal.

3.4 Konstruksi Aplikasi

Program aplikasi dibentuk dengan menggunakan bahasa pemrograman *Python*. Terdapat beberapa *library Python* yang dipakai untuk menunjang penelitian, baik pada proses enkripsi dekripsi ataupun pembuatan aplikasi. Berikut beberapa library yang digunakan:

1. *Math*

   Math adalah *library Python* yang digunakan untuk perhitungan ilmiah dan matematika yang kompleks. *Library* ini memiliki fungsi math seperti

mengevaluasi operasi matematika biasa, operasi modulo, operasi trigonometri, operasi logaritma, dan lain sebagainya.

2. Tkinter

   Tkinter digunakan untuk membentuk sebuah aplikasi antarmuka (GUI). Cotohnya untuk membuat komponen-komponen pada aplikasi seperti *button*, *textbox*, *label* atau *frame* yang mana ini sangat mendukung dalam penciptaan aplikasi GUI .

3.4.1 *Input* dan *Output*

Rancangan aplikasi ini dibagi menjadi tiga bagian yaitu pada bagian pembangkitan kunci RSA-CRT, enkripsi dan dekripsi. Proses pembangkitan kunci memerlukan input p, q dan e untuk menghasilkan output berupa kunci publik dan privat RSA-CRT yang akan dipakai pada proses enkripsi dan dekripsi. Terdapat tombol petunjuk pengisian angka untuk membuka menu tambahan berupa keterangan tipe data yang harus diinput pada menu pembangkitan kunci. *Coding file*, kunci OTP CLCG dan kunci publik RSA-CRT diinput pada proses enkripsi untuk menghasilkan *coding file* yang tersamarkan (*ciphertext*). Proses pengembalian *file* asli memerlukan input berupa *ciphertext*, kunci OTP CLCG dan kunci privat RSA-CRT pada bagian dekripsi. Untuk lebih jelas disajikan tabel input dan output pada Tabel 3.1.

Tabel 3.1 *Input* dan *output* pada rancangan Aplikasi

	Pembangkitan Kunci	Enkripsi	Dekripsi
Input	*p* *q* *e*	*Coding file* *(Plaintext)* Kunci OTP CLCG (x_0, y_0, a, b, r, c) Kunci publik RSA (*e*,*n*)	*Coding file* *(Ciphertext*) Kunci OTP CLCG (x_0, y_0, a, b, r, c) Kunci privat RSA-CRT (*dP*, *dQ*, *qInv*, *p*, *q*)
Output	Kunci Publik RSA dan Privat RSA-CRT	*Coding file* (*Ciphertext*)	*Coding file* (*Plaintext*)

3.4.2 Algoritma OTP menggunakan CLCG

Penelitian ini menggunakan CLCG untuk membangun kunci algoritma OTP sepanjang data. Jika pada algoritma OTP biasa dibutuhkan kunci sepanjang pesan, OTP modifikasi CLCG hanya membutuhkan 6 variabel untuk pembentukan kunci. Variabel yang dibutuhkan pada metode CLCG yaitu x_0, y_0, a, b, r dan c. Variabel r dan c digunakan untuk membentuk matriks pemilihan kunci sedangkan x_0, y_0, a dan b digunakan untuk memilih kunci berdasarkan posisi elemen matriks. Pembentukan matriks memiliki batasan dikarenakan algoritma OTP CLCG menggunakan operasi XOR dalam 1 *byte* sehingga elemen pada matrik tidak boleh lebih dari 255 (1 byte). Contoh pemilihan variabel r dan c yaitu 9 dan 14, maka matriks yang terbentuk tersaji pada Tabel 3.2.

Tabel 3.2 Matriks Pemilihan Kunci OTP CLCG

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	15	16	17	18	19	20	21	22	23	24	25	26	27	28
2	29	30	31	32	33	34	35	36	37	38	39	40	41	42
3	43	44	45	46	47	48	49	50	51	52	53	54	55	56
4	57	58	59	60	61	62	63	64	65	66	67	68	69	70
5	71	72	73	74	75	76	77	78	79	80	81	82	83	84
6	85	86	87	88	89	90	91	92	93	94	95	96	97	98
7	99	100	101	102	103	104	105	106	107	108	109	110	111	112
8	113	114	115	116	117	118	119	120	121	122	123	124	125	126

Untuk rumus pembentukan bilangan acak menggunakan CLCG dijabarkan sebagai berikut:

1. Memilih sembarang bilangan $x_0, y_0, a, b, r, c \in \mathbb{Z}$
2. Hitung $x_1 = ax_0 + b \; mod \; m$ dan $y_1 = ay_0 + b \bmod m$ dengan $m = r \times c$.
3. Hitung $M_{(I,0)} = x_1 \; mod \; r$ dan $M_{(0,J)} = y_1 \bmod c$ dengan r adalah 9 dan c adalah 14.
4. Hitung K_1 dengan cara mencari nilai $M_{(I,J)} = M[x_1 \bmod r][y_1 \bmod c]$ pada matriks yang telah dibuat. K_1 merupakan kunci yang terpilih pada urutan ke-1.
5. Karena x_1 dan y_1 telah diketahui, ulangi langkah 2 sampai 4 untuk mencari kunci ke-*i*.

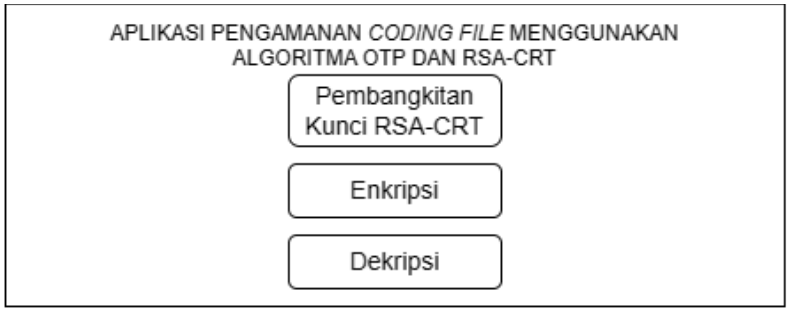
Rumus enkripsi pada algoritma OTP CLCG yaitu $C_i = (P_i \oplus K_i)$ dengan K_i merupakan kunci yang telah dibentuk menggunakan metode CLCG dan telah diubah kedalam bentuk biner sepanjang 1 *byte*. Proses dekripsi *file* dilakukan dengan rumus $P_i = (C_i \oplus K_i)$ di mana kunci yang digunakan sama dengan kunci pada proses enkripsi.

3.4.3 Algoritma RSA-CRT

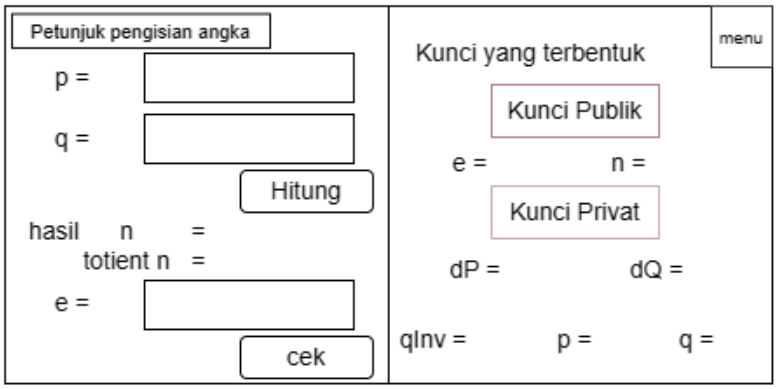
Algoritma RSA membutuhkan waktu yang lama dalam proses enkripsi dan dekripsinya. Oleh sebab itu, dilakukan modifikasi algoritma menggunakan CRT agar proses dekripsi dapat berjalan lebih cepat. Terdapat perubahan variabel kunci yang digunakan pada proses dekripsi. Kunci privat RSA (*d*,*n*) dimasukkan pada perhitungan metode CRT pada persamaan (2.9), (2.10) dan (2.11) sehingga diperoleh variabel *dP, dQ, qInv*. Kunci privat modifikasi yang digunakan ialah $\mathcal{K}$(*dP, dQ, qInv, p, q*). Proses dekripsi dilakukan dengan cara menghitung $P = m_2 + h.q$ di mana nilai m_2, h dan q diperoleh menggunakan rumus pada persamaan (2.12), (2.13) dan (2.14).

3.5 Rancangan Tampilan

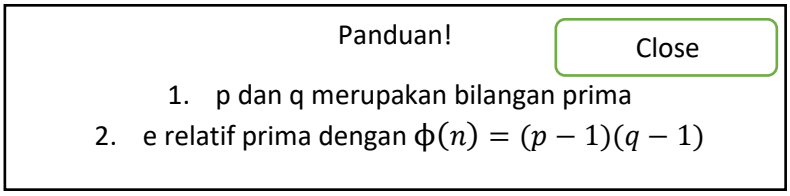
Berikut merupakan rancangan tampilan dari aplikasi yang akan dibuat peneliti.



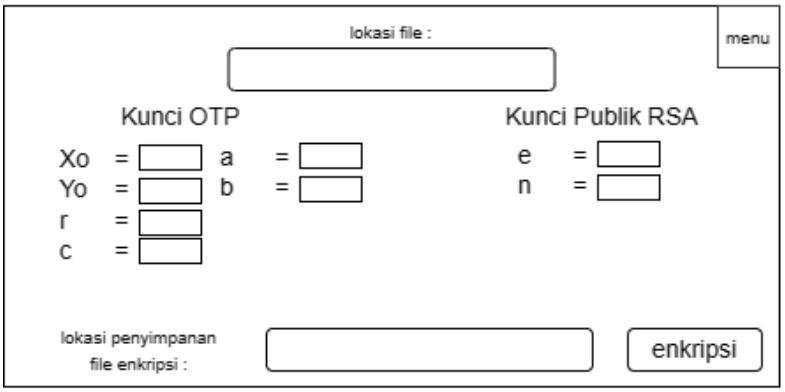
Gambar 3.4 Rancangan Tampilan Menu Utama Aplikasi



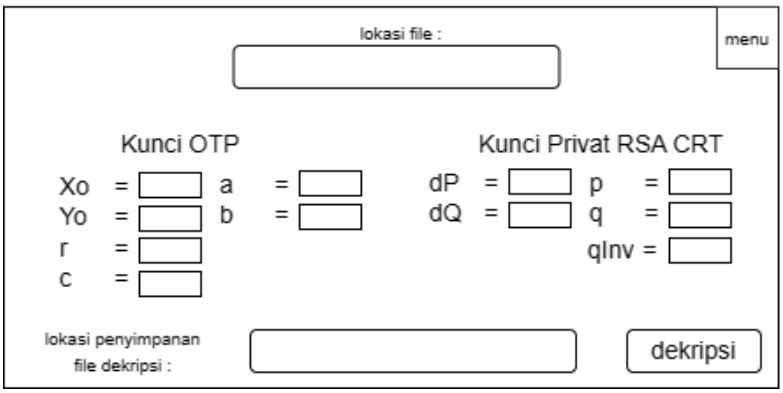
Gambar 3.5 Rancangan Tampilan Menu Pembangkitan Kunci RSA-CRT



Gambar 3.6 Rancangan Tampilan Menu Tipe Data untuk Input Data Pembangkitan Kunci RSA-CRT



Gambar 3.7 Rancangan Tampilan Menu Enkripsi *File*



Gambar 3.8 Rancangan Tampilan Menu Dekripsi *File*

3.6 Validasi

Validasi dilakukan dengan cara menguji *coding file* hasil dekripsi. Hasil *running coding file* asli dibandingkan dengan *running file* hasil dekripsi. Proses mengujian dilakukan berkali-kali menggunakan *coding file* yang berbeda-beda.