Implementasi Rabin *Public Key Cryptosystem* dalam Skema Enkripsi Dekripsi Pesan dengan Telegram Bot

Fadel Azzahra[1)], Girinoto[2)], Faizal Gani Setyawan[3)]
1) Badan Siber dan Sandi Negara, fadel.azzahra@bssn.go.id
2) Politeknik Siber dan Sandi Negara, girinoto@poltekssn.ac.id
3) Badan Siber dan Sandi Negara, faizal.gani@bssn.go.id

Abstrak

Berbagai skema public-key cryptosystem, seperti RSA dan Rabin cryptosystem, didasarkan pada prinsip kesulitan dalam memfaktorkan bilangan bulat besar. Kesulitan ini menjadikan sistem kriptografi tersebut aman terhadap serangan komputasi. Rabin cryptosystem, meskipun memiliki tingkat keamanan yang sebanding dan dikembangkan lebih awal dibandingkan dengan RSA, kurang populer karena proses dekripsinya yang menghasilkan empat kemungkinan pesan, sehingga memerlukan langkah tambahan untuk menentukan pesan yang benar. Penelitian ini bertujuan untuk mengimplementasikan skema dasar Rabin cryptosystem pada Telegram Bot guna mengenkripsi pesan yang dikirim melalui media digital. Implementasi dilakukan dengan memanfaatkan fleksibilitas dan kemampuan kustomisasi Telegram Bot sebagai platform pengujian. Hasil yang diharapkan adalah terwujudnya sistem pengiriman pesan dengan enkripsi Rabin yang berfungsi secara efektif dalam lingkungan Telegram, sekaligus menyediakan wawasan terhadap potensi aplikasinya dalam komunikasi digital.

Kata kunci : Public-Key Cryptosystem, Rabin cryptosystem, Telegram bot

Abstract

Various public-key cryptosystem schemes, such as RSA and Rabin cryptosystems, rely on the difficulty of factoring large integers, a computationally complex problem that ensures their cryptographic security. Although Rabin cryptosystem offers comparable security and was developed earlier than RSA, it is less popular due to its decryption process, which produces four possible messages, necessitating additional steps to identify the correct one. This study aims to implement the basic Rabin cryptosystem scheme in a Telegram Bot to encrypt messages sent through digital media. The implementation utilizes the flexibility and customizability of Telegram Bot as a testing platform. The expected outcome is the development of a functional Rabin encryption-based messaging system within the Telegram environment, providing insights into its potential application in digital communication.

Keywords : Public-Key Cryptosystem, Rabin cryptosystem, Telegram bot

1. PENDAHULUAN

Michael Rabin mengembangkan sistem kriptografi kunci publik Rabin pada periode yang sama dengan pengembangan sistem RSA oleh Rivest, Shamir, dan Adleman [1]. Sama seperti RSA, sistem kripto Rabin didasarkan pada kesulitan masalah faktorisasi bilangan bulat besar, menjadikannya secara teoritis lebih kuat [2]. Namun, RSA menjadi lebih populer karena kemudahannya dalam penggunaan dengan algoritma yang hampir sekuat Rabin. Meski begitu, Rabin masih memiliki keunggulan tersendiri terutama dalam keamanannya yang dianggap lebih kuat secara teoritis [3]. Hal inilah yang menjadi alasan mengapa algoritma Rabin dipilih untuk implementasi aplikasi ini, terutama untuk kebutuhan enkripsi pesan atau file yang lebih aman.

Penelitian mengenai penggunaan algoritma Rabin menunjukkan bahwa, meskipun tidak sepopuler RSA, algoritma ini tetap relevan dalam skenario di mana keamanan tingkat tinggi dibutuhkan, seperti pada aplikasi perpesanan. Salah satu platform yang mendukung implementasi sistem kripto dengan fleksibilitas tinggi adalah Telegram dengan Telegram Bot [4]. Penggunaan Telegram Bot [5] memungkinkan aplikasi kriptografi Rabin dapat dijalankan tanpa terikat pada platform khusus, menjadikannya pilihan yang tepat untuk enkripsi digital.

Penelitian pada aplikasi ini dikembangkan untuk mengimplementasikan sistem kripto Rabin pada Telegram Bot [6], dengan tujuan menciptakan platform enkripsi yang lebih aman dan efisien dalam mengelola pesan atau file.

2. LANDASAN TEORI

2.1 Rabin *Public-Key Cryptosystem*

Rabin *cryptosystem* adalah salah satu *public-key cryptosystem* yang ditemukan oleh Michael Rabin [7]. Rabin masih menjadi salah satu *asymmetric cryptosystem* yang memiliki keunikan properti tersendiri, seperti *low-cost encryption*, enkripsi yang relatif cepat dibandingkan dengan RSA, dan algoritma yang telah terbukti sulit sebagaimana problem faktorisasi bilangan [2]. Meski demikian, terdapat 'kekurangan' dari Rabin, yaitu munculnya empat kemungkinan pesan yang mungkin, dengan hanya satu pesan saja yang benar. Konfigurasi dekripsi Rabin dapat menyebabkan gagalnya dekripsi karena tidak ada indikator bawaan khusus yang menentukan pesan mana yang benar.

Sebagaimana *asymmetric cryptosystem* yang lain, Rabin menggunakan *public key* dan *private key [8]*. Dalam skema asli yang dibuat oleh Michael Rabin, skema pembangkitan kunci (*key generation*) untuk *private key* untuk A adalah p dan q dengan p dan q adalah dua bilangan prima berbeda berukuran besar, saling koprima, memiliki ukuran yang cenderung sama, sedangkan untuk *public key* untuk A adalah n dengan n = pq.

Dimisalkan B hendak mengenkripsi pesan m kepada A, maka B melakukan langkah-langkah berikut:

a. Merepresentasikan pesan m menjadi integer dalam *range* {0, 1, …, n-1}
b. Mendapatkan *public key* A
c. Menghitung $c = m^2 \mod n$

Selanjutnya, B dapat mengirimkan pesan rahasia *c* yang telah terenkripsi.

Untuk mendekripsi pesan rahasia *c*, A menghitung $m = \sqrt{c} \mod n$ yang akan menghasilkan empat kemungkinan untuk m berupa m_1, m_2, m_3, dan m_4, selanjutnya A menentukan pesan *m* yang tepat menggunakan *Chinese Remainder Theorem*. Untuk $p \equiv q \equiv 3 \pmod 4$, kemungkinan pesan *m* dapat dicari dengan cara berikut:

a. Mencari terlebih dahulu nilai *a* dan *b* yang memenuhi persamaan $ab + pq = 1$ menggunakan algoritma Extended Euclidean
b. Hitung $r = c^{(p+1)/4} \mod p$
c. Hitung $s = c^{(q+1)/4} \mod q$
d. Hitung $x = (aps + bqr) \mod n$
e. Hitung $y = (aps - bqr) \mod n$
f. Kemungkinan pesan (yang merupakan akar kuadrat modulo *c*) adalah *x*, $-x \mod n$, *y*, dan $-y \mod n$

Untuk mengatasi ambiguitas dalam pemilihan pesan *m* yang tepat, diterapkan *redundancy padding*. *Redundancy padding* adalah penambahan bit tertentu pada belakang pesan *m* yang hendak dienkripsi yang bertujuan untuk memberikan tanda bahwa pesan *m* tersebut adalah pesan yang benar jika dan hanya jika terdapat *redundancy pad* yang memenuhi kriteria tertentu. Biasanya *redundancy pad* adalah perulangan beberapa bit belakang dari *plaintext*. Dalam praktik skema *Rabin* pada umumnya, biasanya *redundancy pad* mengambil 64-bit terakhir dari pesan *m* sebelum di enkripsi [9]. Nantinya, apabila ternyata dari keempat kemungkinan *m* tidak ada satupun yang memenuhi syarat *redundancy padding*, maka B dapat menolak *ciphertext c* dan menganggap *c* tidak valid.

2.2 Telegram Bot

Telegram Bot adalah jenis akun dalam aplikasi pesan instan Telegram yang beroperasi sebagai aplikasi pihak ketiga yang terintegrasi dalam platform Telegram itu sendiri. Ini memungkinkan pengembang untuk membuat aplikasi interaktif yang dapat diakses oleh semua pengguna Telegram [6]. Bot ini tidak hanya mudah digunakan namun juga berfungsi seperti kontak pesan biasa dalam aplikasi tetapi juga mudah untuk dikembangkan berkat dukungan Telegram terhadap antarmuka pemrograman aplikasi Bot atau *Application Programming Interface* (API) yang khusus. API ini memberikan alat kepada pengembang untuk merancang, membangun, dan menyesuaikan bot secara efektif, memungkinkan mereka untuk mengotomatiskan tugas, mengirim dan menerima pesan, mengintegrasikan dengan sistem eksternal, dan membangun fungsionalitas kompleks yang meningkatkan pengalaman pengguna tanpa perlu platform terpisah. Fleksibilitas dan kemudahan pengembangan menjadikan Bot Telegram serbaguna untuk tujuan pribadi dan bisnis, memungkinkan berbagai kemungkinan interaktif langsung dalam pesan.

3. METODE PENELITIAN

Penelitian ini menggunakan metode pengembangan perangkat lunak berbasis prototipe untuk mengimplementasikan skema Rabin *Public Key Cryptosystem* pada Telegram Bot. Desain sistem dimulai dengan melakukan analisis kebutuhan untuk menentukan fungsi utama yang diperlukan, termasuk enkripsi, dekripsi, dan pengelolaan kunci. Proses desain mencakup pemodelan algoritma Rabin untuk memastikan keamanan data melalui langkah-langkah matematis seperti pembangkitan kunci, enkripsi pesan, dan dekripsi dengan verifikasi redundansi.

Implementasi dilakukan menggunakan Python sebagai bahasa pemrograman dengan memanfaatkan Telegram Bot API untuk membangun interaksi pengguna. Tahap pengujian dilakukan secara iteratif, melibatkan pengujian kinerja algoritma pada berbagai jenis pesan digital, seperti teks dan gambar. Evaluasi dilakukan dengan membandingkan hasil implementasi terhadap teori dan mengukur keefektifan sistem dalam mengenkripsi dan mendekripsi pesan, termasuk verifikasi akurasi pesan hasil dekripsi.

Hasil pengujian dianalisis untuk menentukan keandalan algoritma Rabin dalam lingkungan Telegram Bot, serta mengidentifikasi kelebihan dan keterbatasannya dalam aplikasi komunikasi digital.

4. HASIL DAN PEMBAHASAN

4.1 Algoritma Pembangkitan Kunci

Tabel 1 menjelaskan tahapan-tahapan dalam algoritma pembangkitan kunci untuk sistem enkripsi Rabin. Setiap langkah dalam tabel ini merinci proses *input*, *output*, dan prosedur spesifik yang diperlukan untuk membangkitkan *public key* dan *private key* yang valid.

Tabel 1. Algoritma Pembangkitan Kunci

No.	Proses	Deskripsi
1	*Input*	Ukuran bit untuk *private key*
2	*Output*	*Public key* (*n*), *private key* (*p* dan *q*)
3	Langkah	1. Bangkitkan dua bilangan ganjil acak *p* dengan ukuran bit yang ditentukan 2. Bangkitkan dua bilangan ganjil acak *q* dengan ukuran bit yang ditentukan 3. Jika $q = p$, maka ulangi langkah 2 4. Jika $q > p$, maka tukar nilai *p* dan *q* agar *private key* yang dibangkitkan lebih besar 5. Cek apakah *p* dan *q* saling prima dan $p \equiv q \equiv 3 \pmod 4$. Jika tidak, ulangi langkah 2 6. Hitung $n = pq$

4.2 Algoritma Enkripsi

Tabel 2 menjelaskan proses awal algoritma enkripsi untuk mengonversi *plaintext* menjadi bentuk integer yang siap dienkripsi. Tahapan ini meliputi konversi *plaintext* ke biner, pembagian ke dalam blok, dan penambahan *redundancy padding* untuk memastikan keutuhan data saat enkripsi.

Tabel 2. Algoritma Enkripsi

No.	Proses	Deskripsi
1	*Input*	*Plaintext* (bisa berupa teks atau *file*), *public key* (*n*), ukuran *redundancy pad*
2	*Output*	Bentuk integer *padded plaintext*
3	Langkah	1. Konversikan bentuk plaintext dengan merepresentasikan tiap karakter pada teks dalam heksadesimal (untuk *plaintext* berupa teks) atau membaca isi *file* dalam bentuk heksadesimal (untuk *plaintext* berupa *file*) 2. Konversikan heksadesimal *plaintext* menjadi biner (bilangan basis 2) 3. Gabungkan seluruh biner *plaintext* 4. Bentuk *splitted plaintext* dengan membagi biner *plaintext* menjadi blok-blok biner berukuran ((panjang bit *public key*/2) – ukuran redundancy pad) 5. Jika panjang bit pada blok *splitted plaintext* terakhir tidak memenuhi ukuran, tambahkan bit 0 (*trailing bit*) hingga ukuran memenuhi 6. Bentuk biner *padded plaintext* dengan menambahkan *redundancy pad* sejumlah (ukuran *redundancy pad*) bit dari biner belakang tiap blok pada *splitted plaintext* 7. Konversikan *padded plaintext* menjadi integer

Tabel 3 melanjutkan proses enkripsi dengan langkah-langkah utama untuk menghasilkan *ciphertext*. Pada tahap ini, *padded plaintext* dalam bentuk integer dienkripsi menggunakan *public key* (*n*), menghasilkan *ciphertext* yang siap untuk dideskripsi di tahap berikutnya.

Tabel 3. Proses Enkripsi

No.	Proses	Deskripsi
1	*Input*	Bentuk integer *padded plaintext* (*m*), *public key* (*n*)
2	*Output*	*Ciphertext* dalam bentuk integer (*c*) dengan ukuran bit *c* = ukuran bit *n*
3	Langkah	1. Hitung $c = m^2 \bmod n$

Tabel 4 menjelaskan proses penulisan *ciphertext* ke dalam *file* setelah tahap enkripsi selesai. *Ciphertext* dalam bentuk integer dikonversi ke format biner, dibagi menjadi blok-blok berukuran 8 bit, dan diubah ke heksadesimal. Hasil akhirnya disimpan sebagai *file* terenkripsi yang siap digunakan atau disimpan untuk kebutuhan dekripsi di kemudian hari.

Tabel 4. Penulisan *Ciphertext*

No.	Proses	Deskripsi
1	*Input*	*Ciphertext* dalam bentuk integer (*c*)
2	*Output*	*File ciphertext*
3	Langkah	1. Konversikan *ciphertext* menjadi biner 2. Bentuk *splitted ciphertext* dengan membagi biner *ciphertext* menjadi blok-blok biner berukuran 8 bit 3. Konversikan *splitted ciphertext* menjadi heksadesimal 4. Tulis heksadesimal dari *splitted ciphertext* menjadi *encrypted file*

4.3 Algoritma Dekripsi

Tabel 5 menggambarkan tahap awal dekripsi, yaitu inisiasi *ciphertext*. Pada tahap ini, *ciphertext* yang berasal dari *file* dikonversi menjadi bentuk integer. Proses ini melibatkan pembacaan *ciphertext* dalam format heksadesimal, pengubahan ke format biner, penggabungan blok-blok biner, dan akhirnya konversi ke integer yang siap digunakan dalam proses dekripsi selanjutnya.

Tabel 5. Inisiasi *Ciphertext*

No.	Proses	Deskripsi
1	*Input*	*Ciphertext* berupa *file*, *private key* (*p* dan *q*)
2	*Output*	Bentuk integer *ciphertext*
3	Langkah	1. Konversikan bentuk *ciphertext* dengan membaca isi *file* dalam bentuk heksadesimal 2. Konversikan heksadesimal *ciphertext* menjadi biner, maka akan terbentuk blok-blok biner *ciphertext* 3. Gabungkan ((ukuran bit *pq*)/8)-blok biner *ciphertext* sehingga terbentuk blok-blok biner *concated ciphertext* dengan ukuran bit *pq* 4. Konversikan *concated ciphertext* menjadi integer

Tabel 6 menjelaskan langkah-langkah utama dalam proses dekripsi untuk mengembalikan *ciphertext* ke bentuk *plaintext*. Dalam tahap ini, *ciphertext* dalam bentuk integer didekripsi menggunakan *private key* (*p* dan *q*) dan *redundancy padding* untuk memastikan integritas data. Proses meliputi perhitungan nilai *r*, *s*, *x*, dan *y* serta verifikasi

padded plaintext yang benar melalui pengecekan *redundancy padding*, sehingga diperoleh hasil akhir yang sesuai dengan *plaintext* asli.

Tabel 6. Proses Dekripsi

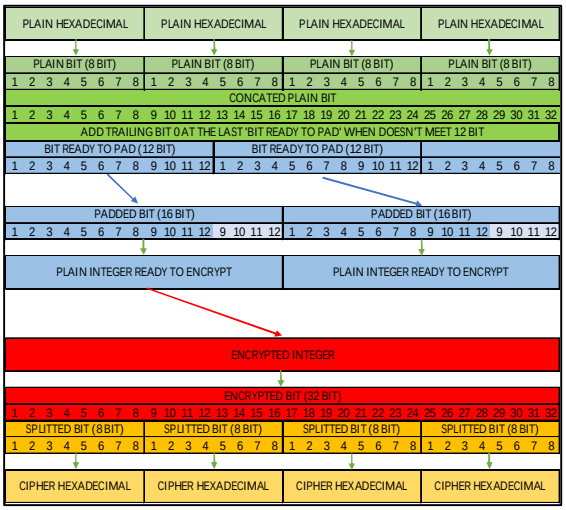
No.	Proses	Deskripsi
1	*Input*	Bentuk integer *concated ciphertext* (*c*), *private key* (*p* dan *q*), ukuran *redundancy pad*
2	*Output*	*Plaintext* yang sesuai dalam bentuk integer (*m*)
3	Langkah	1. Cari *a* dan *b* yang memenuhi persamaan $ab + pq = 1$ 2. Hitung $r = c^{(p+1)/4} \bmod p$ 3. Hitung $s = c^{(q+1)/4} \bmod q$ 4. Hitung $x = (aps + bqr) \bmod n$ 5. Hitung $y = (aps - bqr) \bmod n$ 6. Tetapkan nilai *m* yang mungkin: *x*, *-x* mod *n*, *y*, *-y* mod *n* 7. Cek nilai *padded plaintext* (*m*) yang benar dengan mengonversikan masing-masing nilai *m* yang mungkin ke bentuk biner, kemudian cek adanya *redundancy pad* dengan mengecek (*redundancy pad**2)-bit dari belakang. Apabila ukuran biner *m* sama dengan (ukuran bit *n*)/2 dan terdapat perulangan bit (ditemukan *redundancy*), maka *m* dipilih menjadi *padded plaintext* yang sesuai

Tabel 7 menguraikan langkah akhir dari algoritma dekripsi, yaitu mengembalikan *padded plaintext* yang telah didekripsi ke dalam bentuk awalnya sebagai *plaintext* asli. Pada tahap ini, integer hasil dekripsi dikonversi dari bentuk biner ke teks atau format file asli, menghilangkan *padding* yang ditambahkan saat enkripsi. Proses ini memastikan bahwa data yang didekripsi sesuai dengan *plaintext* awal sebelum dienkripsi, sehingga dapat dibaca atau digunakan sesuai tujuan aslinya.

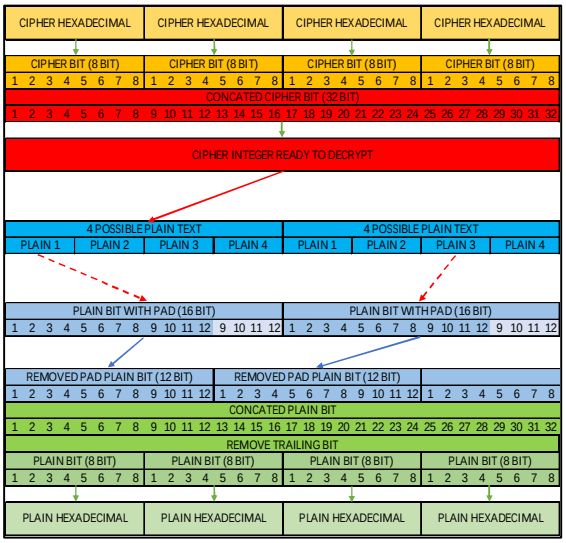
Tabel 7. Penulisan *Plaintext*

No.	Proses	Deskripsi
1	*Input*	*Padded plaintext* dalam bentuk integer (*c*), ukuran bit *private key*, ukuran *redundancy pad*
2	*Output*	*Plaintext* (bisa berupa teks atau file)
3	Langkah	1. Konversi integer *plaintext* menjadi biner 2. Hapus *redundancy pad*: hapus (ukuran *redundancy pad*)-bit *plaintext* dari belakang 3. Gabungkan semua biner *plaintext* menjadi *concated plaintext* 4. Hapus bit 0 (*trailing bit*) dari *concated plaintext* bagian belakang sejumlah ((panjang *concated plaintext*) (mod ukuran bit *private key*)) 5. Bentuk *splitted plaintext* dengan membagi biner menjadi blok-blok dengan ukuran masing-masing blok adalah 8 bit (1 byte) 6. Konversikan *splitted plaintext* menjadi heksadesimal 7. Tulis heksadesimal dari *splitted plaintext* menjadi *decrypted file* dengan menambahkan format *file* yang sesuai

Berikut adalah penggambaran secara visual untuk algoritma enkripsi dan dekripsi



Gambar 1. Algoritma Enkripsi



Gambar 2. Algoritma Dekripsi

Dalam algoritma tersebut, algoritma *redundancy padding* yang digunakan adalah menambahkan *pad* pada blok-blok bit *plaintext* dengan ukuran tiap bloknya ((ukuran *public key*/2)-ukuran *redundancy pad*) dengan tujuan untuk mempertahankan ukuran blok-blok bit *plaintext* tetap sesuai dengan ukuran *public key*/2 setelah *pad* ditambahkan. Selain itu, sifat difusi *plaintext* akan muncul karena blok-blok bit *plaintext* yang akan ditambahkan *pad* tidak hanya berasal dari satu blok *plaintext* saja, namun juga berasal dari potongan blok *plaintext* yang lain. Dengan demikian, penambahan ukuran *plaintext* setelah dienkripsi sama dengan (panjang bit keseluruhan *plaintext*) / ((ukuran *public key*/2)-ukuran *redundancy pad*) * (ukuran *pad*). Selanjutnya keseluruhan algoritma tersebut oleh penulis diaplikasikan pada Telegram Bot.

4.4 Penulisan *Plaintext*

Pada tahap ini, hasil dekripsi berupa *padded plaintext* dalam bentuk integer dikonversi kembali ke

bentuk awalnya sebagai *plaintext* asli. Proses ini melibatkan penghapusan *redundancy padding*, konversi data ke format biner, dan pengembalian ke bentuk teks atau file asli sesuai dengan format yang digunakan sebelum enkripsi. Langkah-langkah ini memastikan bahwa pesan yang diterima setelah dekripsi identik dengan pesan awal sebelum dienkripsi.

4.5 *Testing*

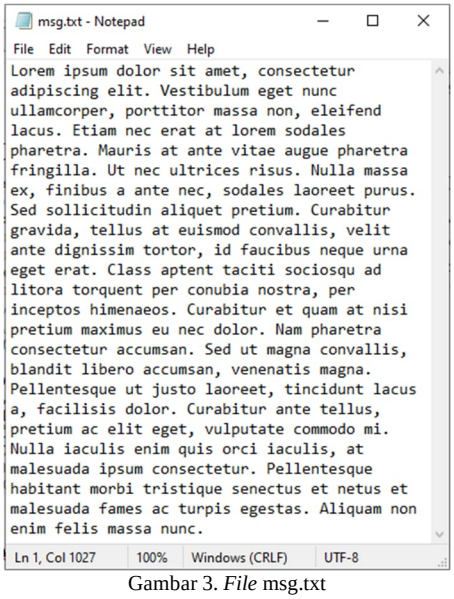
Pengujian aplikasi Rabin pada Telegram Bot akan dilakukan dengan membangkitkan kunci dan menjalankan proses enkripsi-dekripsi sebanyak dua kali, yaitu pada *file* teks dan file gambar. *Private key* yang digunakan memiliki ukuran 16 bit. Berikut ini adalah langkah-langkah setiap pengujian yang dilakukan melalui Telegram Bot, yang hasilnya akan ditampilkan di konsol. *Output* dari Telegram Bot akan ditandai dengan simbol >>.

Pembangkitan Kunci oleh A

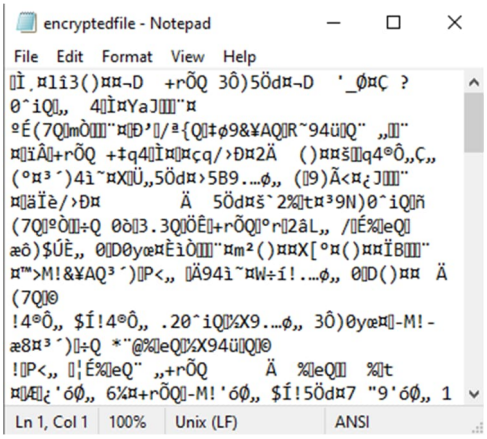
>> "Masukkan ukuran private key"
16
>> "Kunci 16 bit berhasil dibangkitkan"
>> "p: 63671"
>> "q: 54583"
>> "n: 3475354193"
>> "p dan q adalah private key anda"
>> "n adalah public key anda"

Enkripsi 1 oleh B

>> "Masukkan public key penerima"
3475354193
>> "Ukuran public key: 32 bit"
>> "Kirimkan pesan/file yang ingin dienkripsi"
<file msg.txt>
>> "Mendapatkan file dengan tipe text/plain"
>> "Enkripsi berhasil"
>> <file encryptedfile>
>> "Ukuran file awal: 1024 bytes"
>> "Ukuran file terenkripsi 1368 bytes"



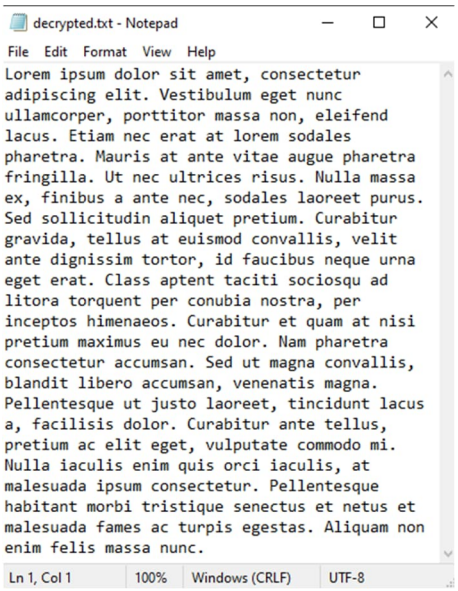
Gambar 3. *File* msg.txt



Gambar 4. *File Encryptedfile*

Dekripsi 1 oleh A

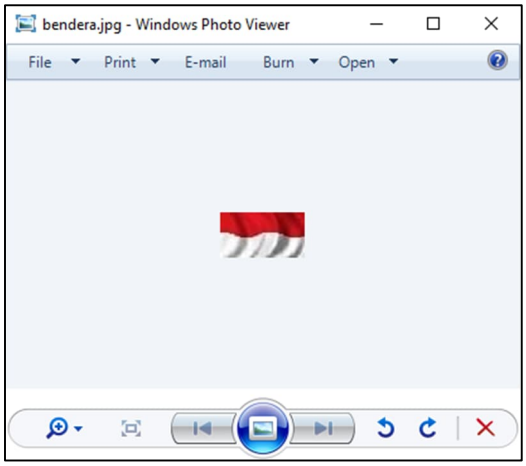
>> "Masukkan private key dengan format: p q"
63671 54583
>> "Ukuran private key: 16 bit"
>> "Kirimkan file yang ingin didekripsi"
<file encrypted.txt>
>> "Mendapatkan file terenkripsi"
>> "Dekripsi berhasil"
>> "Mendapatkan pesan/file dengan tipe text/plain"
>> <file decrypted.txt>
>> "Ukuran file terenkripsi: 1368 bytes"
>> "Ukuran file didekripsi 1024 bytes"



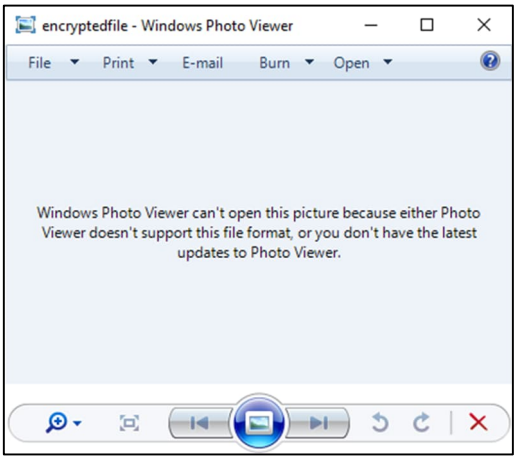
Gambar 5. *File* decrypted.txt

Enkripsi 2 oleh B

>> "Masukkan public key penerima"
3475354193
>> "Ukuran public key: 32 bit"
>> "Kirimkan pesan/file yang ingin dienkripsi"
<file bendera.jpg>
>> "Mendapatkan file dengan tipe image/jpeg"
>> "Enkripsi berhasil"
>> <file encryptedfile>
>> "Ukuran file awal: 879 bytes"
>> "Ukuran file terenkripsi 1172 bytes"



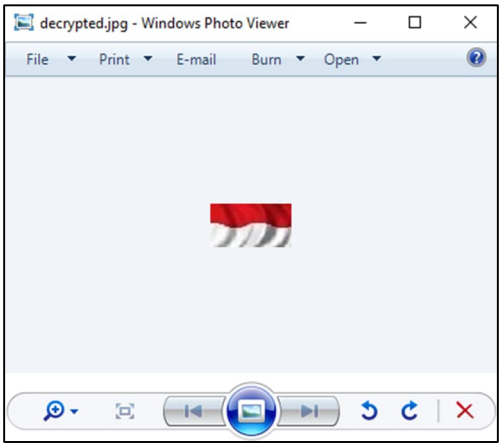
Gambar 6. *File* bendera.jpg



Gambar 7. *File Encryptedfile*

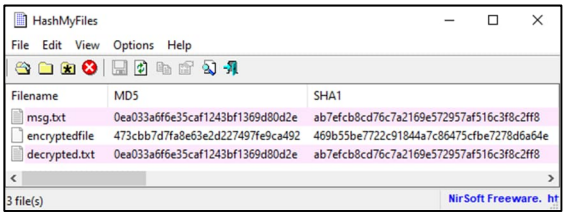
Dekripsi 2 oleh A

>> "Masukkan private key dengan format: p q"
63671 54583
>> "Ukuran private key: 16 bit"
>> "Kirimkan file yang ingin didekripsi"
<file encrypted.jpg>
>> "Mendapatkan file terenkripsi"
>> "Dekripsi berhasil"
>> "Mendapatkan pesan/file dengan tipe image/jpeg"
>> <file decrypted.jpg>
>> "Ukuran file terenkripsi: 1172 byte"
>> "Ukuran file didekripsi 879 byte"

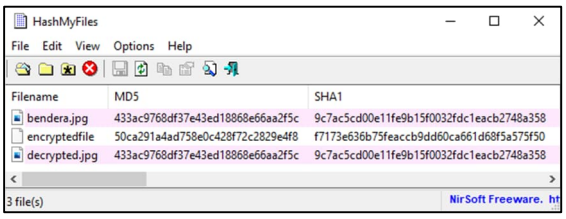


Gambar 8. *File* decrypted.jpg

Berikut dilampirkan hasil penghitungan nilai *hash* untuk mengecek isi dari *file* sebelum dienkripsi, setelah dienkripsi, dan setelah didekripsi



Gambar 9. Hasil Penghitungan Nilai *Hash File* Hasil Proses Enkripsi-Dekripsi 1



Gambar 10. Hasil Penghitungan Nilai *Hash File* Hasil Proses Enkripsi-Dekripsi 2

Dari hasil di atas, dapat disimpulkan bahwa Rabin mampu mengenkripsi dan mendekripsi pesan dengan baik (tidak ada perubahan dalam *plaintext*) meskipun Rabin 'memiliki kelemahan' dalam munculnya empat kemungkinan *plaintext* yang mungkin. Hal ini menandakan bahwa *redundancy padding* berhasil menghilangkan ambiguitas pada proses dekripsi.

5. KESIMPULAN

Meskipun tidak sepopuler algoritma RSA, Rabin *cryptosystem* tetap memiliki keunikan dan potensi yang signifikan sebagai salah satu *asymmetric cryptosystem* yang dapat diterapkan dalam skema enkripsi-dekripsi *plaintext*. Keunggulan dari Rabin terletak pada strukturnya yang sederhana dan kemampuannya untuk menyediakan keamanan yang kuat. Salah satu tantangan dalam algoritma dekripsi ini adalah adanya potensi untuk menghasilkan lebih dari satu solusi. Namun, masalah ini dapat diatasi dengan penerapan teknik *redundancy padding*, yang membantu memastikan bahwa hasil dekripsi dapat diandalkan dan sesuai dengan input awal.

Rabin *cryptosystem* juga dapat diaplikasikan pada platform sederhana, seperti Telegram Bot, terutama untuk tujuan pendidikan dan penelitian. Hal ini membuka peluang bagi pengembang dan peneliti untuk mengeksplorasi dan memahami konsep kriptografi dengan lebih baik. Namun, dalam penerapan algoritma dalam bentuk program aplikatif, keefektifan dan efisiensi algoritma perlu diperhatikan dengan serius. Layaknya algoritma RSA, Rabin melakukan penghitungan matematis dengan angka yang relatif besar, yang dapat mempengaruhi kinerja aplikasi jika tidak dioptimalkan. Kedepannya diharapkan dilakukan penelitian dan pengujian untuk mengeksplorasi dan meningkatkan performa Rabin *cryptosystem*.

REFERENSI

[1] R. Thombre and B. Jajodia, "Experimental Analysis of Attacks on RSA & Rabin Cryptosystems using Quantum Shor's Algorithm," in *Proceedings of International Conference on Women Researchers in Electronics and Computing*, AIJR Publisher, Sep. 2021, pp. 587–596. doi: 10.21467/proceedings.114.74.

[2] M. A. Budiman, Handrizal, and S. Azzahra, "An implementation of Rabin-p cryptosystem and affine cipher in a hybrid scheme to secure text," in *Journal of Physics: Conference Series*, IOP Publishing Ltd, Jun. 2021. doi: 10.1088/1742-6596/1898/1/012042.

[3] Y. Salami, V. Khajehvand, and E. Zeinali, "Cryptographic Algorithms: A Review of the Literature, Weaknesses and Open Challenges," *Journal of Computer & Robotics*, vol. 16, no. 2, pp. 63–115, 2023.

[4] D. Ismawati and I. Prasetyo, "The Development of Telegram BOT Through Short Story," in *Proceedings of the Brawijaya International Conference on Multidisciplinary Sciences and Technology (BICMST 2020)*, Paris, France: Atlantis Press, 2020. doi: 10.2991/assehr.k.201021.049.

[5] N. Hafiz, O. C. Briliyant, D. F. Priambodo, M. Hasbi, and S. Siswanti, "Remote Penetration Testing with Telegram Bot," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 7, no. 3, pp. 705–714, Jun. 2023, doi: 10.29207/resti.v7i3.4870.

[6] D. Ismawati and I. Prasetyo, "The Development of Telegram BOT Through Short Story," in *Proceedings of the Brawijaya International Conference on Multidisciplinary Sciences and Technology (BICMST 2020)*, Paris, France: Atlantis Press, 2020. doi: 10.2991/assehr.k.201021.049.

[7] X. Zhao and D. Li, "A Lightweight User Authentication Scheme for Multi-Gateway Based Wireless Sensor Networks Using Rabin Cryptosystem," *IEEE Access*, vol. 11, pp. 79874–79889, 2023, doi: 10.1109/ACCESS.2023.3300440.

[8] R. K. Ramesh, R. Dodmane, S. Shetty, G. Aithal, M. Sahu, and A. K. Sahu, "A Novel and Secure Fake-Modulus Based Rabin-3 Cryptosystem," *Cryptography*, vol. 7, no. 3, Sep. 2023, doi: 10.3390/cryptography7030044.

[9] M. A. Budiman and T. A. Irvida, "Securing text using Rabin-p public key cryptosystem and Spritz algorithm in a hybrid cryptosystem: a tutorial," in *Journal of Physics: Conference Series*, Institute of Physics, 2023. doi: 10.1088/1742-6596/2421/1/012010.